Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models

Sunita Sarawagi
IIT Bombay

Soumen Chakrabarti
IIT Bombay

ABSTRACT

Over 40% of columns in hundreds of millions of Web tables contain numeric quantities. Tables are a richer source of structured knowledge than free text. We harness Web tables to answer queries whose target is a quantity with natural variation, such as `net worth of zuckerburg`, `battery life of ipad`, `half life of plutonium`, and `calories in pizza`. Our goal is to respond to such queries with a ranked list of quantity distributions, suitably represented. Apart from the challenges of informal schema and noisy extractions, which have been known since tables were used for non-quantity information extraction, we face additional problems of noisy number formats, as well as unit specifications that are often contextual and ambiguous.

Early "hardening" of extraction decisions at a table level leads to poor accuracy. Instead, we use a probabilistic context free grammar (PCFG) based unit extractor on the tables, and retain several top-scoring extractions of quantity and numerals. Then we inject these into a new collective inference framework that makes global decisions about the relevance of candidate table snippets, the interpretation of the query's target quantity type, the value distributions to be ranked and presented, and the degree of consensus that can be built to support the proposed quantity distributions. Experiments with over 25 million Web tables and 350 diverse queries show robust, large benefits from our quantity catalog, unit extractor, and collective inference.

1. INTRODUCTION

Web search engines enhance "organic" search results with data from structured knowledge bases (KBs), curated from diverse sources using information extraction [13] and entity annotation [5] techniques. With very few exceptions [10, 19, 3, 1], a vast majority of work on extracting typed text segments, entities, attributes and relations involve discrete symbols and not *measurable quantities*[1]. And yet, the Web is as rich a source of quantities as it is of symbolic knowledge. There are hundreds of millions of information-rich HTML tables on the Web. In our sample, a full 40% of their columns exclusively contain quantities. On the other hand, *quantity-seeking* Web queries are well-served only in verticals like shopping, food and travel where structured databases contain columns that are matched against the query, aggregated, and presented as quantities throughout. With one mild exception [20], the emerging Web tables literature [17, 4, 7, 11] does not regard quantity extractions as any different from discrete symbolic value extractions.

1.1 Our goal

Our goal is to bridge the gap between the noisy presentation of quantities in Web tables and open-domain, one-off quantity-seeking queries. Asking for an attribute of an entity is a common case, e.g., `average revenue of microsoft`, `half-life of plutonium` and `mass of pluto`. Note that we are primarily interested in seeking quantities with *uncertainty* and *imprecision*, unlike "cosmic truths" like the number of primes under 1000, or the value of π. For this reason, our queries cannot be answered from Wikipedia Infoboxes, which mostly offer point answers, and without directly accumulating evidence from the Web. To this end, we present a new, robust, open-domain, quantity search system QEWT.

QEWT is a large system with very many details. Here we distill three major contributions: a quantity catalog and a new table column unit annotator based on a probabilistic context free grammar (PCFG), a query response model based on ranked value distributions, and a new algorithm for collective consensus inference. Our unit catalog, query workloads, labeled data, and unit annotator are publicly available[2].

1.2 Our contributions

Table column unit annotator. Quantity columns in web tables are notoriously challenging to extract. Unit expressions may be ambiguous ("m.") or even missing, syntactic clues though present might be noisy, and units may not follow clear-cut representations in table headers. We compiled a quantity catalog QuTree, and designed a new unit annotator for table columns based on probabilistic context free grammars (PCFGs). The PCFG technique exploits a diverse set of clues, including co-occurrence statistics between quantity types, units and phrases mined from an *unlabeled* corpus of table headers.

Quantity response model. In IR, the response is a list of URLs or documents. In expert or entity search [2], it is a list of entities. Queries that seek uncertain quantities require a very different treatment. In particular, extracted quantities can be distinct from each other for extraneous reasons, or there may be systematic variations (which isotope of Plutonium, or Microsoft revenue in which year). A more intuitive uniform output representation is a *distribution over quantities.* We describe techniques to represent, score and rank such distributions h, given the query.

[1] "When you can measure what you are speaking about, and express it in numbers, you know something about it." — Lord Kelvin.

[2] `http://www.cse.iitb.ac.in/~sunita/wwt`

http://dx.doi.org/10.1145/2623330.2623749 .

Earlier work [3, "QCQ"] is a restrictive special case of our framework. Each response in QCQ is a single *interval*, and a model was trained to score these using labeled data. Here we can create distributions and intervals in a query-driven manner without labeled data. Since our target is open-domain query answering, we ensure that our response model can handle data at arbitrary scales and from arbitrary distributions via a data-driven distance metric.

Collective extraction framework. The centerpiece of our approach is a collective model for extracting quantities from raw Web tables. Instead of "hardening" extraction decisions on quantity columns eagerly, we keep around multiple uncertain values and units, which are finally collectively resolved over multiple tables at query time. We depend on a consensus model that defines a joint distribution h over all candidate extractions to assign collectively resolved probabilities on the relevance of each snippet and the extracted unit and value. Another valuable input is from a classifier that provides a distribution over the target quantity type. We train the classifier via a novel method of tapping our huge unlabeled table corpus.

Experimental evaluation. We report on experiments with 350 queries spanning three query benchmarks compiled from QCQ [3], InfoGather [20] and World Bank data [18]. The base corpus tables is accessed through the Google API at research.google.com/tables and our corpus of 25 million tables extracted [11] from a commercial Web crawl of 500 million pages. Our experimental results can be summarized as:

- Our PCFG-based column annotator is considerably more accurate than rule-based (82% vs 40%) or simpler baselines (74%). However, it is not nearly perfect.
- As a consequence, choosing locally best extractions from each table independently gives poor accuracy. Collective judgment of snippet relevance and extractions improves answer precision from 28% to 42%.
- Smooth notions of consensus between contributing quantities is important; accumulating counts for point estimates (collapsing distribution h to a histogram over discrete values) causes a 6% drop in precision.
- The quantity type classifier trained from the unlabeled table corpus provides a 8% boost in precision.

2. SYSTEM OVERVIEW AND ROADMAP

We present an overview of our system QEWT in Figure 1. The user submits a quantity query q which has two parts. The first part a_q is a sequence of word/s that are a verbal description of the response quantity type (e.g., distance from sun, speed, annual revenue). The second part e_q is a free-form sequence of words that indicate an entity for which a quantity attribute is being sought (e.g. Pluto, Concorde, Microsoft). The query words are submitted to indices over Web table corpora and a set of tables retrieved. Figure 3 shows sample tables for the query `co2 emissions of china` with $e_q =$ `china` and $a_q =$ `co2 emissions`. A snippet generator module processes the retrieved tables for potential answer snippets and attaches a relevance score to each. Next, a number/unit extractor parses the noisy snippets to output an uncertain list of values and units based on a unit ontology. The uncertain extractions from all tables are collectively resolved to get a distribution over the target quantity. Finally the response as a continuous distribution or ranked quantity intervals is output to the user. We next present an overview of the main components.

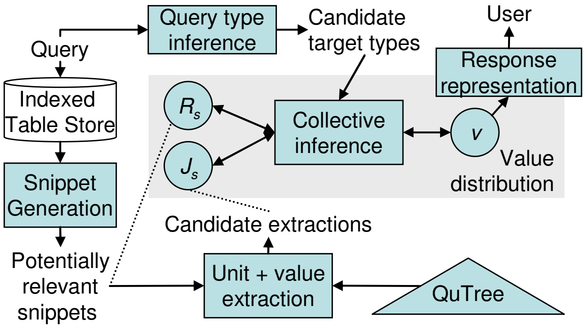


Figure 1: System sketch of QEWT.

Table corpus. We use two sources of tables: a commercial crawl of 500 million Web pages, from which we extracted and indexed 25 million non-decorative HTML tables, and tables collected per-query via the API provided by research.google.com/tables. We extract from each table zero or more top rows as the header for each of its columns, and selected text from the page embedding the table as the context as described in [11]. A type interpreter labels table columns as numeric or textual; in our corpus of 25 million tables we recognized 40% of the table columns as numeric.

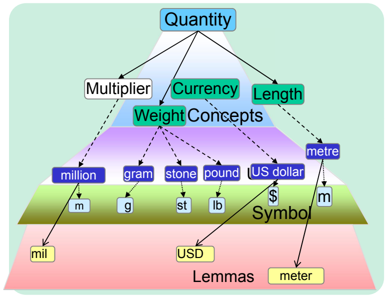
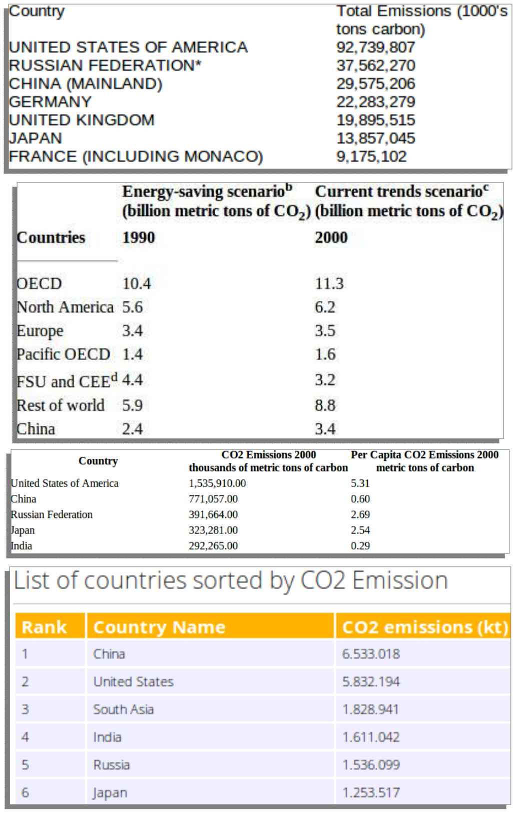


Figure 2: Fragment of QuTree.

Unit catalog QuTree. Starting from Category:Units_of_measurement in Wikipedia, we created a unit catalog we call QuTree. It has 44 quantity types such as *Length*, *Area*, *Speed*, and 750 units. Each quantity type has a *canonical* unit, and other units come with conversion factors to/from the canonical unit. A fragment of QuTree is shown in Figure 2. Each catalog unit is associated with one or more full names (e.g., kilometre per hour), one or more symbols (e.g., kmph, km/h) and an optional list of lemmas to account for the common variant names of a unit, (e.g., meter-metre, kmph-kilometre per hour). QuTree also includes the concept of a "multiplier" to denote dimensionless quantities, with unit instances like thousand, million and billion, which capture scales of measurements. QuTree can be downloaded from goo.gl/542L2Y. It is to quantities what YAGO [14] is to discrete entities.

Query target type. In the user query, a_q is a textual hint at a quantity type t from the quantity catalog. We must build an estimate of the distribution $\Pr(t|q) = \Pr(t|a_q)$. For example, for attribute `co2 emissions` $\Pr(t|a_q)$ is expected to assign high probability to the quantity type *mass*. In Section 3.1 we show how to train $\Pr(t|a_q)$ via an innovative use



Country	Total Emissions (1000's tons carbon)
UNITED STATES OF AMERICA	92,739,807
RUSSIAN FEDERATION*	37,562,270
CHINA (MAINLAND)	29,575,206
GERMANY	22,283,279
UNITED KINGDOM	19,895,515
JAPAN	13,857,045
FRANCE (INCLUDING MONACO)	9,175,102

	Energy-saving scenario[b] (billion metric tons of CO_2)	Current trends scenario[c] (billion metric tons of CO_2)
Countries	1990	2000
OECD	10.4	11.3
North America	5.6	6.2
Europe	3.4	3.5
Pacific OECD	1.4	1.6
FSU and CEE[d]	4.4	3.2
Rest of world	5.9	8.8
China	2.4	3.4

Country	CO2 Emissions 2000 thousands of metric tons of carbon	Per Capita CO2 Emissions 2000 metric tons of carbon
United States of America	1,535,910.00	5.31
China	771,057.00	0.60
Russian Federation	391,664.00	2.69
Japan	323,281.00	2.54
India	292,265.00	0.29

List of countries sorted by CO2 Emission

Rank	Country Name	CO2 emissions (kt)
1	China	6.533.018
2	United States	5.832.194
3	South Asia	1.828.941
4	India	1.611.042
5	Russia	1.536.099
6	Japan	1.253.517

Figure 3: Sample tables responding to query e_q = `china` and a_q = `co2 emissions`.

of the table corpus, and without requiring tedious manual labeling. We will see in Section 6.4 that this signal is very useful to the collective extractor in finding relevant answers.

Snippet relevance. The index search returns a set of candidate tables that match the query keywords q in the header, context, or body. On each candidate table, we use a snippet generation module to match e_q to a row header, a_q to a column header, and identify candidate cells for quantity extraction. Observe from Figures 3 and 4 the challenges of identifying potentially relevant rows, e.g., "China" vs. "China (mainland)", and relevant columns, e.g. multiple columns might be relevant and column headers may not match a_q at all as shown in Figure 4. Therefore, relevance of a snippet is uncertain. We use a random variable R_s to denote the uncertainty in the relevance of a snippet s and associate a score $r_s \in [0,1]$ to denote $\Pr(R_s = 1)$. We will describe snippet generation and relevance score assignment further in Section 3.2.

Number and unit extraction. From each snippet s, we extract a float value from the table cell, and annotate it with a unit from QuTree based on signals in the column header and the table cell. Note that a unit node in QuTree belongs to exactly one quantity type, so a unit annotation also gives its quantity type. Both these tasks are highly challenging because of the extreme diversity of writing down quantities on the web: multipliers expressed separately in the column header, locale-dependent use of commas, periods, and spaces, scientific notation, etc. can confound any simplistic local extractor. Unit extraction from headers has its own challenges of dealing with drastic formatting differences in both column headers and cell quantities E.g., one column header may say "Total Emissions (1000's tons carbon)" whereas another may say "(billion metric tons of CO_2)" — these are different, whereas yet another table saying "CO2 emissions (kt)" is comparable to the second case. In Section 5 we present our design of a parser based on a context free grammar and a rich feature set. The output of this step is a weighted list of possible extractions of values and units from each snippet.

Collective inference. The candidate extractions are inputs to the centerpiece of QEWT: a novel collective inference procedure that simultaneously estimates the relevance of each snippet and each of its possible extractions, and the target quantity type basing its judgement on a global estimate of the distribution of values around the quantity. We describe this in Section 3.5.

Answer representation. As challenging as robustness to input variation is the issue of answer representation. The vast body of work on search has to rank discrete items such as URLs (pages) [8] or entities [2]. Here our true response is an uncertain quantity, but how best is this shown to the user? Rather than take an inflexible stand, QEWT supports three different types of answer formats each of which is suited for a different kind of quantitative queries. QEWT's internal response representation is a quantity distribution, but QEWT can provide simplified digests in the form of ranked value intervals [3], or degenerated to ranked point values (in case the query has little or no uncertainty, such as physical constants, or are categorical queries in disguise, such as the number of USB ports in a laptop).

3. COLLECTIVE CONSENSUS INFERENCE

As shown in Figure 1, the collective inference module, which we describe in this section, mediates between the type distribution, snippet relevance uncertainty, and the uncertainty of snippet unit and value extractions. Section 3.1 is about estimating the target type of the query. Section 3.2 discusses scoring snippets wrt the query. Section 3.3 describes how a single snippet can lead to many possible extractions of a unit and value, and how to score these extractions. Section 3.4 deals with the design of distributions for consensus inference. Finally, Section 3.5 describes the consensus inference algorithm itself.

3.1 Query target quantity type $\Pr(t|q)$

Given query $q = (a_q, e_q)$, its target quantity type, which is uncertain, is modeled as a distribution $\Pr(t|a_q)$. A good estimate of $\Pr(t|a_q)$ is vital for generating relevant responses.

3.1.1 Dictionary match

A baseline method may look for matches of the attribute words with type and unit names/lemmas in our quantity catalog. It may correctly find the target type of queries like `length of nile` or `year of crash`. But many queries like `usa co2 emissions`, `walton net worth`, `miami rainfall`, and `ebay revenue` have no match in the catalog, leading to loss of recall. Surprisingly, there is also precision loss. E.g., query `fan speed` which matches well the type *speed* in QuTree but misses the correct target type *frequency*.

3.1.2 Data-driven approach

Ideally, we would like to recognize that query word `revenue` targets type *money amount* and that `distance`, `length`, `height` and `width` all target the type *length* (dimension).

This form of association between (potential query) words and quantity types is evident in a fraction (but still a large absolute number) of tables, with column headers like "Width in inches", "co2 emissions in kiloton", "Average rainfall in inches", "Fan speed (rpm)", and "Annual revenue ($ million)". To take advantage of these headers, they must be mapped to types, which is, in general, a highly nontrivial job (see Section 5). However, starting from over 25 million tables, we could find 1.1 million headers which could be mapped with high precision (99%) to types, using simple rules discussed in Section 5.1, and restricting to headers with an exact and unique match with a unit in QuTree.

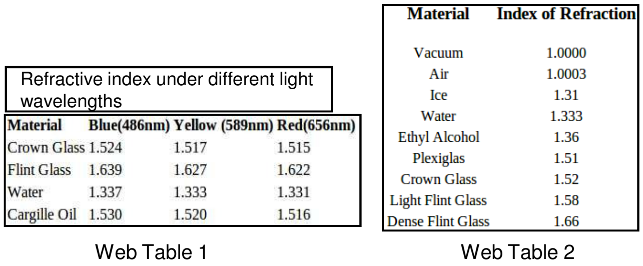
We thus get (automatically) labeled instances with observed features consisting of the words $\mathbf{x}_k$ in the header not included in the unit, and a quantity type t_k derived by generalizing its extracted unit. E.g., from the header text "Annual revenue ($ million)" we extract an instance with bag of words $\mathbf{x} = \{annual, revenue\}$ and type *money amount*. These labeled instances are used to train a logistic regression classifier for $\Pr(t|q)$. 3-fold cross validation gave 94% accuracy, higher than using PMI models [16]. If the posterior entropy was large (e.g., for a query on `refractive index`), we prefered the "dimensionless" type in favor of other types.

3.2 Snippet match with query

Each table retrieved from the index generates one or more snippets. We represent the uncertainty of relevance of a snippet s to q by $R_{s,q}$, or R_s if q is fixed. The uncertainty of relevance is reflected in a score r_{sq} or $r_s \in [0, 1]$. Based on query q, the local evidence in favor of relevance ($R_s = 1$) is r_s, and the local evidence in favor of irrelevance ($R_s = 0$) is $1 - r_{qs}$.

Given query $q = (a_q, e_q)$ and a candidate table T, we describe how we match these two parts of the query to generate one or more snippets from T. We view web tables as vertical stores of entities and their attributes[3]. Accordingly, a candidate snippet is generated by matching e_q to a row r in a column c_e and matching a_q to a different column c_a and generating the snippet from cell (r, c_a). For most queries, we find a single snippet per table but sometimes the entity e_q could match multiple rows of a table, or a_q could match multiple columns of T. For example, in Figure 3 the first three web tables generate a single snippet for query `co2 emissions of china`. In contrast, for the query `refractive index of flint glass`, in Figure 4, the second table matches the entity "flint glass" in the last two rows whereas the first web table provides the attribute "Refractive index" in three different forms over columns 2, 3, and 4. We next elaborate on our method for generating such snippets.

The candidate tables are required to match all high-IDF terms in the query in either the body, header, or context. This ensures that our candidate snippets are minimally relevant. Thereafter, we separately measure the similarity score $\text{sim}(e_q, r, c_e, T)$ of entity e_q to cell (r, c_e) and similarity score $\text{sim}(a_q, c_a, T)$ of attribute string a_q to column c_a's header. These match scores are custom designed to depend on matches beyond the immediate cells to include T's context, title, and other parts of its body. In [11] we presented the design of a segmented similarity function that shows how to measure the relevance of a table's column to a query keyword while combining column-specific match with matches from the rest of the table. We use this segmented similarity function as the match function: $\text{sim}(e_q, r, c_e, T)$ and $\text{sim}(a_q, c_a, T)$.



Refractive index under different light wavelengths

Material	Blue(486nm)	Yellow (589nm)	Red(656nm)
Crown Glass	1.524	1.517	1.515
Flint Glass	1.639	1.627	1.622
Water	1.337	1.333	1.331
Cargille Oil	1.530	1.520	1.516

Web Table 1

Material	Index of Refraction
Vacuum	1.0000
Air	1.0003
Ice	1.31
Water	1.333
Ethyl Alcohol	1.36
Plexiglas	1.51
Crown Glass	1.52
Light Flint Glass	1.58
Dense Flint Glass	1.66

Web Table 2

Figure 4: Snippet match with query.

The snippets from a table are generated as follows. We first find the (r, c) with the best value of $\text{sim}(e_q, r, c, T)$. Call it (r^*, c_e^*). Fix c_e^* as the entity column. We then find the column c that is numeric and has maximum similarity $\text{sim}(a_q, c, T)$. Call it c_a^*. Next, as snippets we select those cells r_s, c_s for which:

$$\begin{aligned}\text{sim}(e_q, r_s, c_e^*, T) &\geq \max(\text{sim}(e_q, r^*, c_e^*, T) - \epsilon, \sigma)\\ \text{sim}(a_q, c_s, T) &\geq \max(\text{sim}(a_q, c_a^*, T) - \epsilon, 0)\}\end{aligned}$$

This criteria makes sure that we select all snippets $s = (r_s, c_s)$ whose match is close enough (within ϵ) of the best possible match from T, provided the entity match is at least σ. In our experiments we used 0.2 for both ϵ and σ. The reason we have a minimum match threshold for entities and not for attributes is because the number of numeric columns in a table is typically much smaller than the number of rows. We depend on the global consensus model to prefer the columns that are correct. We use the entity and attribute match scores to assign the relevance score of a snippet as $r_s = (\text{sim}(a_q, c_s, T) + \text{sim}(e_q, r_s, c_e^*, T))/2$.

3.3 Possible extractions from snippet

We model each snippet s as extracting exactly one value and one unit (therefore, type). However, given the noise in extraction, we model the extracted value and type as uncertain, but drawn from a finite, usually small set of alternative extractions. This set of possible extractions is indexed by j. For example, consider the web table in Figure 1 obtained in response to query `height of washington monument`.

Height (m)	Year	Building
168,7	1884	Washington Monument
95,8	1899	Old Post Office
91,5	1990	Washington National Cathedral
87,6	1892	United States Capitol
39,2	1943	Jefferson Memorial

Table 1: An example web table for query `height of washington monument`.

We get one snippet from this table at the cell (1,1), but we have uncertainty over its value: does 168,7 equal 168.7, or a list of two numbers 168 and 7. This gives rise to two possible values: 168.7 and 168? Also, from the header Height (m), the unit parser extracts three possible units: meter, million, and mile. So, we consider all six combinations of unit and value as possibilities for j. Each j is attached with a score g_{sj} that we generate as follows: First, parse the value in the cell based on different locale specific parsers and get the set of successful parses of the value $v_{s1}, \ldots, v_{sn}$. Second, use the CFG unit parser described in Section 5 to get a list of

[3] We omit a discussion of horizontal tables for simplicity [6].

units along with their scores: $(u_{s1}, w_1), \dots, (u_{sm}, w_{sm})$. The set j consists of the mn possible cross product of value,unit combination where each j is associated with: a value v_{sj}, a unit u_{sj} and therefore its type t_{sj}, and a confidence score $g_{sj} = \frac{w_{sj}}{n} \geq 0$. (Note that a specific extraction j has an extracted unit, and each unit maps to a unique quantity type. For convenience, we can convert a specific extraction to some canonical unit that we associate with the quantity type.) Summarizing, for each snippet s, there are two hidden random variables:

- $R_s \in \{0, 1\}$, the snippet relevance bit.
- J_s, an extraction index. If $R_s = 0$, then $J_s = \perp$ is undefined. Otherwise, J_s tells us what value and unit/type s contributes to the global consensus.

3.4 Value distribution h

In abstract terms, a quantity query should be answered with a type (unit) and a value distribution h. For each potential response type τ, we will build a distribution $h_\tau(v)$, with the usual constraint $\int_\bullet h_\tau(\bullet) d\bullet = 1$.

h_τ will be estimated from a set of values $\{v_i\}$, but, based on the previous sections, each value v_i is associated with a probability π_i that the value should actually contribute to h_τ. This is expressed with the notation $h_\tau(\bullet | \{(v_i, \pi_i)\})$.

Simple parametric distributions are not a good choice for h, given that we have to deal with numbers of arbitrary scale, coming from arbitrary domains, with a high level of extraction noise. Therefore, we focus on non-parametric distributions. We briefly mention two standard forms of h_τ for the sake of completeness.

3.4.1 Kernel density

A natural option is a kernel density estimate:

$$h_\tau(\bullet; \{v_i, \pi_i\}) = \frac{1}{Z} \sum_i \frac{\pi_i}{\sqrt{2\pi}\sigma_i} e^{(\bullet - v_i)^2 / 2\sigma_i^2}, \qquad (1)$$

where σ_i is a kernel width parameter that may be the same for all i as a special case, and $Z = \sum_i \pi_i$. A problem that we faced with a fixed width is that they do not adapt well to numbers of arbitrary scales. Consider the query that seeks to find the half-life of Plutonium. Plutonium has many isotopes with extremely diverse half-lives: 14, 88, 6560, 24100, and 376000 years. Naturally these are stated approximately, with errors that are commensurate with the magnitude of the quantity being expressed. We associated a different width σ_i around each data point that increases with the scale of the point v_i as max(10% of v_i, minimum non-zero gap between numbers).

3.4.2 Wavelet

Another popular non-parametric density estimator is based on Wavelets. Wavelets are believed to outperform kernel density estimators at representing discontinuities and local variations, features rampant in our data. We use the Haar wavelet as our basis function. Due to lack of space, we refer the reader to standard textbooks on the topic and omit further details.

3.5 Consensus inference algorithm

3.5.1 Intuition

Now we are in a position to verbally express how collective consensus is formed, given a query. We will state that various scores "should be large", with the understanding that, in a collective scheme, we want some sort of aggregate (say their product) to be large. We will then translate this description into a formal model.

- If τ is a top-scoring response quantity type, then $\Pr(\tau|q)$ should be large.
- Suppose we choose specific values for all snippet relevance variables R_s. Then the score of that configuration is
$$\prod_{s:R_s=1} r_s \prod_{s:R_s=0} (1 - r_s), \qquad (2)$$
and this should be large.
- If a snippet is irrelevant, no specific extraction needs to be chosen for that snippet. If snippet s is relevant, we need to pick one extraction $J_s = j$, for which $t_{sj} = \tau$ (i.e., the extracted type matches the query target type), and g_{sj} is large.
- Consider now all relevant snippets s and their chosen extractions J_s. These induce a multiset of values $\{v\}$. If we are to propose value distribution h to the user, we want the density at $\{v\}$ to be large. Assuming iid extraction events[4], this can be written as $\prod_{s:R_s=1} h(v_{s,J_s})$.
- If h were an arbitrary distribution of unlimited complexity, it could support an arbitrary set of extracted values, no matter how disparate from each other. Therefore h needs to be *regularized*, and there needs to be consensus among the values claimed to be contributing to h.

In the rest of this section, we fill in the details of the above framework.

```
Inputs: $\Pr(t|q)$; for all snippets $s$, $r_s$ and $t_{sj}, v_{sj}, g_{sj}$ for all candidate extractions $j$.
Evolving variables: $\check{r}_s, \check{g}_{sj}$
initialize hidden variables $\check{g}_{sj} \leftarrow g_{sj}\, r_s\, \Pr(t_{sj}|q)$
for iterations $i = 1, 2, \dots$ do
  for each snippet $s$ do
    let $h_\tau^{\backslash s}(\bullet)$ be a value distribution estimated from all snippets except $s$, using weights $\check{g}_{sj}$
    for each candidate extraction $j$ do
      $\phi_{sj} \leftarrow g_{sj}\, r_s\, \Pr(t_{sj}|q)\, h_{t_{sj}}^{\backslash s}(v_{sj})$ {consensus}
    end for
    $\phi_{s\perp} \leftarrow 1 - r_s$
    $D \leftarrow \phi_{s\perp} + \sum_j \phi_{sj}$
    $\check{r}_s \leftarrow (1/D) \sum_j \phi_{sj}$
    for each $j$ do
      $\check{g}_{sj} \leftarrow \phi_{sj}/D$
    end for
  end for
end for
```

Figure 5: Collective inference pseudocode.

3.5.2 Iterative update algorithm

Each snippet s that potentially contributes information to the response of a query has the associated hidden variables R_s and J_s. The collective inference algorithm will build estimates of the posterior distributions over R_s, J_s through the procedure shown in Figure 5. Specifically, let $\check{r}_s \in [0, 1]$ be the posterior probability of relevance of snippet s. If the snippet is relevant, then the probability of extraction j being valid is $\check{g}_{sj}$, with $\sum_j \check{g}_{sj} = 1$ for each s. We also maintain, for

[4] Multiple extractions from a table are not iid. QEWT handles them, but we skip discussing them for simplicity.

all possible response types τ, the value distributions denoted $h_\tau(\bullet)$.

The consensus update $\phi_{sj} \leftarrow g_{sj}\, r_s\, \Pr(t_{sj}|q)\, h^{\setminus s}_{t_{sj}}(v_{sj})$ can also be interpreted as a "leave-one-out" validation of v_{sj} in the backdrop of other contributing values, as explained below. (Think of index i here as $\langle s, j\rangle$.) If v_i is dropped while making the estimate h_τ, we will write it as $h^{\setminus i}_\tau(\bullet|\{(v_i, \pi_i)\})$. A natural notion of consensus among $\{(v_i, \pi_i)\}$ can be obtained by dropping each (v_i, π_i) in turn, forming the estimate $h^{\setminus i}_\tau(\bullet|\{(v_i, \pi_i)\})$, and evaluating $h^{\setminus i}(v_i, \{(v_i, \pi_i)\})$. It tells us how strongly v_i itself is supported by the rest of the observed values.

The lines after the consensus are to update $\check{r}_s$ and $\check{g}_{sj}$ to normalized posterior probabilities. After (sufficient) convergence, we are left with these posterior probabilities, from which we construct h_τ on all values (with their posterior probabilities). The resulting h_τ and/or values with their posterior probabilities are then sent to the answer interval representation module, described next.

4. INTERVAL REPRESENTATION

Expressive densities described in the previous section give us a great deal of power to fit the extracted values. However, the user may wish to view something simpler than such a general density. Following QCQ [3], we propose approaches to present ranked *intervals* as the query response. We associate each interval I with a lower limit ℓ_I and upper limit u_I and a p_I that denotes the probability that the answer lies within $[\ell_I, u_I]$. The final answer is a set $\mathcal{I}$ of non-overlapping intervals $I_1, \ldots, I_k$ such that $\sum_{I \in \mathcal{I}} p_I = 1$. A baseline approach would be to cluster the relevant values, but the relevance of a value is not known for sure, and the number of clusters is also unknown. Therefore, we need more sophisticated methods.

For this part snippet boundaries are not relevant, so we denote our input as a set of (v_i, π_i) pairs where a i corresponds to some sj and $\pi_i = \check{g}_{sj}$.

4.1 Uniform density mixture

$\mathcal{I}$ can also be thought of as a mixture of uniform distribution with density

$$h_{\mathcal{I}}(v) = \int_v \sum_{I \in \mathcal{I}} \frac{p_I}{u_I - \ell_I} \delta(v \in [\ell_I, u_I]) dv \qquad (3)$$

Thus, one method of obtaining $\mathcal{I}$ is to approximate a more expressive density (Section 3.4) to a mixture of uniform distributions. Let $h(\bullet|\{(v_i, \pi_i)\})$ be a density (such as a Kernel density). We find our desired intervals $I_1, \ldots, I_k$ such that within each $I = [\ell_I, u_I]$, the error of approximating density h by a constant is within a tolerance ϵ while minimizing the number of such intervals. We measure error in terms of KL divergence between h and $h_{\mathcal{I}}$; this makes the error per interval as $\int_{v=\ell_I}^{u_I} h(v) \log h(v)\, dv - p_I \log \frac{p_I}{u_I - \ell_I}$ where $p_I = \int_{v=\ell_I}^{u_I} h(v)\, dv$. We allow intervals to be single points $\ell_I = u_I$, and assign an error of 0 for such intervals.

Since we restrict the interval boundaries to values in V, we can easily find the optimal set of intervals in $O(|V|)^2$ time using a simple range segmentation algorithm.

4.2 MDL-intervals

One problem with the above method is the difficulty of finding one tolerance ϵ to fit all query types. We next present a method based on the minimum description length (MDL) principle [12] that is more adaptive to per-query variations in value distributions.

Assume we have a set S_q of values sampled from all rows of the snippet columns for query q. For example, in Figure 3 the set S_q will be sampled from all distinct co2 emission values over all countries. Our goal is to find intervals $\mathcal{I} = I_1, \ldots, I_k$ to represent the point answers $V = \{(v_i, \pi_i)\}$. Note that each $v_i \in S_q$. Using MDL, we interpret this as a compression problem from a hypothetical sender of V to a receiver, with the intervals serving as a compression model for V. We assume S_q is known to both parties. The cost of sending V has two parts: the cost of the model, which in our case is the set of intervals, and the cost of sending the data V given model.

The cost of sending the data V given intervals $\mathcal{I} = I_1, \ldots I_k$ is the sum of the cost of data in each interval. Let S_I denote the set of numbers from S_q that lie in interval $I = [\ell_I, u_I], p_I$. As per $\mathcal{I}$, all values within $[\ell_I, u_I]$ have probability $r_I = \frac{p_I}{|S_I|}$ of being relevant. Each entry v_i in V is relevant with probability π_i, thus the expected number of bits for sending v_i is: $-\pi_i \log r_I - (1 - \pi_i) \log(1 - r_I)$. Other values in S_I are irrelevant and require $-\log(1 - r_I)$ bits each. Summing up, data cost over all intervals is $-\sum_{I \in I_1,\ldots I_k} \sum_{v_i \in I} -\pi_i \log r_I + \pi_i \log(1 - r_I) - |S_I| \log(1 - r_I)$.

The model cost is the cost of sending the parameters r_I and the boundary $[\ell_I, u_I]$. Following MDL, we encode r_I by finding a good fit distribution $p(r_I)$, and setting the cost as $-\log p(r_I)$. Since r_I is between 0 and 1, a natural choice is the Beta distribution whose density is $\Pr(p; m, n) = \frac{1}{B(m,n)} p^{m-1}(1-p)^{n-1}$ where $B(m, n)$ is the Beta function and m, n are parameters. We need to choose m, n so as to minimize the cost of sending r_Is over all intervals and all queries. Since S_q is typically larger than V, we choose the prior parameters as $m = 1, n = 2$ so that smaller values of r_I get lower cost. The set of intervals that minimize the sum of data and model cost can be found in $O(|V|^2)$ time using the same segmentation algorithm as in Section 4.1.

5. UNIT EXTRACTOR

So far we have abstracted the role of the extractor that finds likely units from table columns. Given the noise in tables bearing quantities, the module that annotates table columns with units needs to be fairly sophisticated.

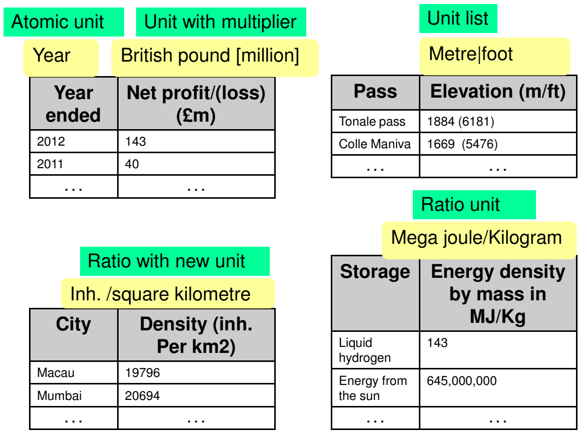
Problem statement. Given a numeric table column with a textual header $\mathbf{x}$ our goal is to extract the units (if any) from $\mathbf{x}$ that associate with the numbers in the column. Figure 6 shows some example unit annotations to table columns. As shown, we handle four types of unit occurrences:

- atomic units like year and meter that have exact match with a unit node in our quantity ontology QuTree,
- units with multipliers (e.g. million pounds),
- compound units formed by taking a ratio or product of two units (e.g. mega joule/kilogram), and
- a list of units (e.g. metre|foot).

5.1 An initial rule-based extractor

We initially attempted to extract based on a set of intuitive rules capturing various lexical clues and matches with the unit catalog, since all previous work on quantities have relied on rule-based unit extractors [3, 20].

Let the term MATCH refer to the longest sequence of tokens in a header $\mathbf{x}$ that matches a unit name, symbol, or


Atomic unit
Year
Unit with multiplier
British pound [million]
Unit list
Metre|foot
Ratio with new unit
Inh. /square kilometre
Ratio unit
Mega joule/Kilogram

Year ended	Net profit/(loss) (£m)
2012	143
2011	40
…	…

Pass	Elevation (m/ft)
Tonale pass	1884 (6181)
Colle Maniva	1669 (5476)
…	…

City	Density (inh. Per km2)
Macau	19796
Mumbai	20694
…	…

Storage	Energy density by mass in MJ/Kg
Liquid hydrogen	143
Energy from the sun	645,000,000
…	…

Figure 6: Example unit annotations (in yellow) on table columns.

lemma in QuTree. Annotating a unit based purely on a MATCH leads to many false positives because there are several words like “in”, “at”, “last”, “s”, “stone”, “point” that are unit names/symbols but are commonly used as non-unit words. Therefore, we defined rules that require additional evidences for a MATCH in $\mathbf{x}$ to be tagged a unit:

1. AFTER-IN: A MATCH after IN is a unit e.g. “Price in \$”, “Distance in km”, “Wind velocity in miles per hour”.
2. BRACKETED: A MATCH within brackets is a unit e.g “Net profit (\$)”, “CO2 emissions (kiloton)”
3. TYPE-NAME: A MATCH tied to a unit U whose parent type name appears in the header is unit U. E.g. “Length of race, m” is annotated U=`Metre` and “Cruise Speed, mph.” is annotated `Miles per hour`

On deploying these rules on our table corpus we were surprised by their large number of errors. For example, the rule AFTER-IN wrongly labels the header “Scores in last match” as unit `Last` (a unit of Volume), and cannot disambiguate between units `Carat, Knot,` and `Kiloton` in header “Capacity in kt” since “kt” is a symbol for each of these three units. The rule BRACKETED wrongly labels word “dec” in header “Population (Dec 2006)” as unit Decade. Also, it cannot differentiate between units in headers “duration(s)” and “year(s)” where the unit annotation of the first should be `Seconds` but the second should have no unit. The rule TYPE-NAME wrongly annotates header “Length of song(m:s)” as `Meter` whereas the correct annotation for “m” is `Minute`. The rule is particularly bad for compound units, for example in “Energy density by volume (MJ/L)”, volume helps annotate L as `Liter`. Rules for compound units are not easy because they require simultaneous labeling of many different parts of the headers. Finally, these set of rules have poor recall, for example, the header “CO2 Emissions 2000 thousands of metric tons of carbon” from the third table in Figure 3 is not covered by any of the three rules.

We therefore explored alternative models that combine multiple soft evidence from additional resources and are more expressive in their modeling of compound units and other unit patterns like multipliers. The rampant ambiguity of a mention with language words and a unit and with other units, implied that just depending on clues derived from the header string may not be adequate. Further, since the system of derived units is based on a well-defined grammar, it seemed natural to use a grammar to drive the extraction. Feature-based context free grammars seemed perfect for the task. Regular grammars cannot capture patterns that encourage alternative units to be of the same type, as in “Meters per second (m/s)”.

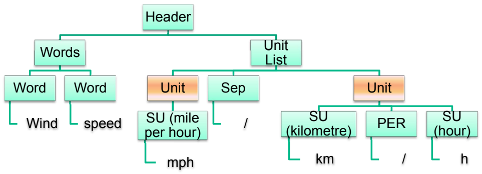


Figure 7: An example parse for the column header `Wind␣speed␣(mph␣/␣km/h)`. SimpleUnit is abbreviated to SU, and unary nodes like CUnit and AtomUnit have been deleted for compactness.

Header	::=	Words? Unit-List Words?
Unit	::=	CUnit | Multiplier Msep CUnit | CUnit Msep Multiplier | Msep Multiplier
Msep	::=	Empty | OF | IN
CUnit	::=	SimpleUnit | SimpleUnit UnitOp SimpleUnit
UnitOp	::=	Empty | PER | ’/’ | ×
SimpleUnit	::=	AtomUnit | Multiplier AtomUnit
AtomUnit	::=	Unit_in_QuTree | New_word
Multiplier	::=	Multiplier_Unit_in_QuTree | Number

Figure 8: Grammar used by the CFG

5.2 Feature-based context free grammar

In this approach we use a discriminative Context Free Grammar (CFG) with scores attached to each possible production in the grammar [15]. In Table 8 we show the CFG (without scores) for unit extraction from table headers. The grammar supports all four different types of compound units illustrated in Figure 6. In Figure 7 we present an example parse tree for the header: “Wind speed (mph / km/h)” that this grammar supports.

In general, the grammar allows many possible parses of a header $\mathbf{x}$. Each parse-tree has a score that is additive over each of the productions in the tree. A production P of the form: $R ::= R_1 R_2$ is scored as:

$$\text{score}(P) = \mathbf{w}.\mathbf{f}(P, \mathbf{x}, i, j, k) \tag{4}$$

where (i, j) and $(j + 1, k)$ are the text spans in $\mathbf{x}$ that R_1 and R_2 cover, respectively. When R_2 is empty ($j = k$), we have a unary production. The feature vector $\mathbf{f}$ can be used to capture various clues that help identify units. We list the set of features we used in sections 5.2.1,. . . 5.2.5. The weight vector $\mathbf{w}$ corresponding to the features could be trained using the discriminative framework of [15], but since the number of features in our case was small (seven) and easily interpretable, we fixed their values manually.

The task of annotating units in an input $\mathbf{x}$ reduces to the task of finding the tree of production with the highest sum of scores, and outputing the list of units under the “Unit” nodes in the tree. Since we have scores, as against hard rules, we can output multiple extractions each weighted with a score. The well-known Inside-outside algorithm [13, 9] for inference in PCFGs can be easily extended to output the top-K highest scoring extractions in polynomial time.

We next list the features used in $\mathbf{f}(P, \mathbf{x}, i, j, k)$. We first list the features for leaf-level productions where R is a unit name U which could be either a Multiplier or an AtomicUnit, and finally in Section 5.2.5 present features for other internal nodes of the parse tree. We will use the short form $\mathbf{x}_{ij}$ to denote the token subsequence $x_i \ldots, x_j$ of $\mathbf{x}$.

5.2.1 Matches with the Unit Catalog

The TF-IDF similarity of $\mathbf{x}_{ij}$ to various parts of unit $U's$ entry in QuTree, including U's name, lemmas, and symbol is an important feature for productions that tag $\mathbf{x}_{ij}$ as a unit name. Another feature analogous to the TYPE-NAME rule, is the TF-IDF similarity of the words in the parent quantity type of U and tokens in $\mathbf{x}$ *excluding* $\mathbf{x}_{ij}$. For example, in the header "Length (m)", this feature will apply for production `Meter :=` $\mathbf{x}_{22}$ since the word "Length" matches the parent type of unit Meter. However, it does not fire for production `million :=` $\mathbf{x}_{22}$.

5.2.2 Lexical clues

When each of the AFTER-IN and BRACKETED rules apply on a $\mathbf{x}_{ij}$ we fire a feature when R is the unit state "Unit". To allow for occasional extraneous tokens, (e.g. length in approx meter), we also fire these tokens with a tolerance of one token.

5.2.3 Relative frequency

We exploit ontologies such as WordNet that provide relative frequency of word usage to get disambiguation clues between units and non-units and between different symbols of the same unit. WordNet provides relative frequency of various senses of nouns in its ontology. Each noun sense s is associated with a list of word-forms and a frequency $f(s)$. Let $S = \{s_1, \ldots, s_m\}$ be the set of senses whose word-forms match $\mathbf{x}_{ij}$. If one of these senses, say s, is a descendant of the Quantity type in Wordnet and matches the base name of the unit U in QuTree, we fire a feature with value $1 - \frac{f(s)}{\sum_{t \in S} f(t)}$. For example, with $\mathbf{x}_{ij}$="last", we found eight matching senses in Wordnet, with more than 99% frequency on the non-unit meaning of last as "finish" or "end". This provided strong evidence to not tag "last" as a unit.

5.2.4 Co-occurrence statistics from table corpus

Another strong clue for correctly assigning a unit U to $\mathbf{x}$ is obtained from the presence of strongly co-occurring words in $\mathbf{x}$ outside unit words $\mathbf{x}_{ij}$. For example, for $\mathbf{x}$ = "width in m", and unit U = `metre` for token x_{22}, we can exploit the fact that "width" often co-occurs with `Length` units. We use the unlabeled corpus of table headers to train a query type classifier as described in Section 3.1.2. Using this, we assign a feature with value $\Pr(t_U|\mathbf{x})$ where t_U is the parent type of unit U.

5.2.5 Compound, Multiple, Multiplier units

We next add a set of features to handle compound units. For each compound unit type: unit-multiplier pair (e.g. dollar [million]), and ratio/product of two units (MJ/L) we add bias terms so that atomic units are preferentially related via these operations instead of being treated as a list of unrelated units. Finally, when two units belong to the same type, we add a bias term so that they are treated as alternatives (e.g. metre|feet).

6. EXPERIMENTS

After describing our testbed in Section 6.1, we perform controlled studies on collective extraction (§6.2), choice of value distribution h (§6.3), query type inference (§6.4), response interval generation approaches (§6.5), and the effect of various unit extractors (§6.6).

6.1 Testbed

Our corpus of tables was collected from two sources: a commercial Web crawl with 500M pages similar to ClueWeb09[5] from which we extracted 25 million tables [11], and tables returned by research.google.com/tables in response to queries.

We used three sources of queries:

QCQ: 28 diverse queries used in [3].

WorldBank: 172 queries on four quantity attributes of countries: forest area, co2 emissions, land area, and population. Ground truth is from World Bank documents [18].

InfoGather: 146 queries used in InfoGather [20], on three different types of quantities: population of 50 large cities of the world and revenue and profit of 34 large corporations. Land area of countries was also part of this workload, but since we had already included in WorldBank, we dropped it from this set.

Queries with ground truth are at goo.gl/542L2Y.

As in QCQ [3], we need to design how the performance of QEWT is measured. Most commonly, ground truth G is available as a set of values, or one or a few ranges. Similar to QCQ, we assume that the query specifies a multiplicative confidence band ϵ. I.e., if a true point value is v, the user would be satisfied with a value in $[(1-\epsilon)v, (1+\epsilon)v]$ (Our results are with $\epsilon = 0.02$). Meanwhile, QEWT presents a distribution h. The *probabilistic precision* of h is

$$\int_v h(v)\delta\Big([(1-\epsilon)v, (1+\epsilon)v] \in G\Big) dv, \tag{5}$$

which is the total area matching a ground truth value band. Recall is defined as the fraction of G supported by h[6]. The probabilistic F1 score is the harmonic mean of probabilistic precision and recall. Given the total area under h is always 1, any attempt to recall all ground values by "smearing out" h will result in large swathes of h never being touched by a ground truth band. Since the form of h shown to the user is a set probability weighted set intervals, we use $h_{\mathcal{I}}$ (Equation 3) for h.

6.2 Benefits of collective answer extraction

QEWT potentially improves upon two simpler baselines. In **CollectiveHard**, for each snippet s, we greedily and locally choose the extraction with the best scoring unit and value. I.e., J_s is pinned. The only uncertainty is in relevance R_s, for which h is used. For $\Pr(t|q)$ we use the hard scores from Section 3.1.1 which is 1 when a_q matches the catalog, and uniform otherwise. In **Independent**, h is missing but the rest of the setting is exactly the same as in **Collective**. The default representation for h is variable width kernel density. Figure 9 shows that the full power of QEWT's collective extraction is vital. In workload InfoGather, **CollectiveHard** suffers particularly severely because of the hard method of query type assignment on the `corporate tax rate` queries.

6.3 Effect of choices of h

With the value of h established, in Figure 10 we compare choices for h: variable and fixed width kernel density, wavelets, and a degenerate distribution with impulse probabilities only at point values seen in the snippets, essentially treating quantities as discrete symbolic extractions.

[5] lemurproject.org/clueweb09

[6] Note, an alternative definition of probabilistic recall is precision times the fraction of G supported by h. But that causes precision to be counted twice in F1 and is redundant.

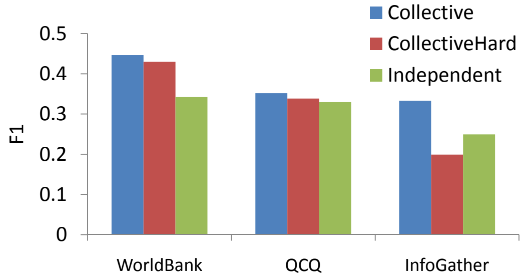


Figure 9: Comparing collective extraction models.

Whereas there is no clear winner among the continuous distributions, they are all better than point histograms, which are bad for queries like net worth of a celebrity, or revenue of a company that are often stated approximately; it gets no benefit of consensus from values that are close by (e.g. $ 111.5 billion versus $ 111.05 billion).

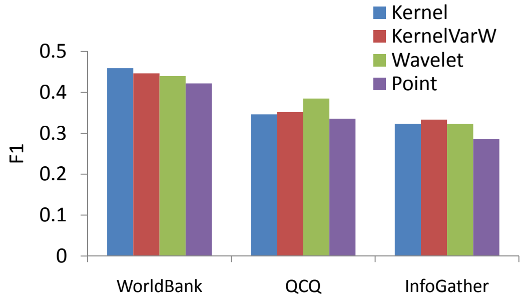


Figure 10: Comparing different models for h(.)

6.4 Effect of query type prediction $\Pr(t|q)$

We compare the dictionary match approach (§3.1.1) with the data-driven approach (§3.1.2). Out of the 35 distinct attribute names spanning the three workloads, dictionary match correctly identified only ten of the query types. In contrast, our data-driven approach correctly classified 34 of them with confidence at least 0.66. Some example attributes and their type and score from this method appear in Figure 11. When the data-driven unit extractor is plugged into the collective answer extractor, the result is a significant boost to end-to-end accuracy, as shown in Figure 12.

6.5 Interval generation methods compared

Figure 13 compares MDL-intervals (§4.2) vs. uniform mixture approximations (§4.1). The x-axis is proportional to the (assigned) marginal cost of an additional segment (one uniform distribution). The y-axis shows F1. At low segment cost, responses are over-fragmented, losing recall. At high segment cost, precision is lost.

Part of the reason for MDL's superiority is the guidance from the reference value distribution. Figure 14 shows an example, for the query "corporate tax rate of united states". There are 12 distinct correct answers in the tight interval $[39.05, 39.34]$ (%), shown as green crosses. The candidate extracted values $V = \{(v_i, p_i)\}$ are shown as red squares (some wide off the mark). About 100 reference values forming S_q, sampled from tables on corporate tax rates of several countries are between 20% and 40%, shown as blue diamonds. Because of the large reference density in $[35, 45]$, the intervals (with probabilities) created by MDL are **[39.2,39.4]: 0.61**, [35,35]: 0.1, [40,40]: 0.09, and [39.0,39.1]: 0.09. The most likely one is shown as a green box, containing most true values. In contrast, the top response from UniformMix is the low-precision interval [35.0,45]: 0.91, because it has no query-specific method of measuring relative distances between points.

Attribute	Type	$\Pr(\tau\|a_q)$
annual rainfall	Length	0.92
revenue	Currency	0.71
weight	Mass	0.96
CO2 emissions	Mass	0.87
depth	Length	0.97
distance to sun	Length	0.97
net worth	Currency	0.83
population	Multiples	0.73
half life	Time	0.67

Figure 11: Query attributes a_q, labeled type τ and $\Pr(\tau|a_q)$ via logistic classifier.

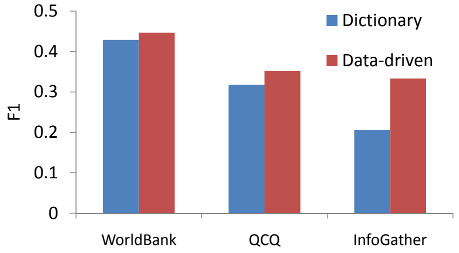


Figure 12: Query F1 accuracy under different query-type classification models.

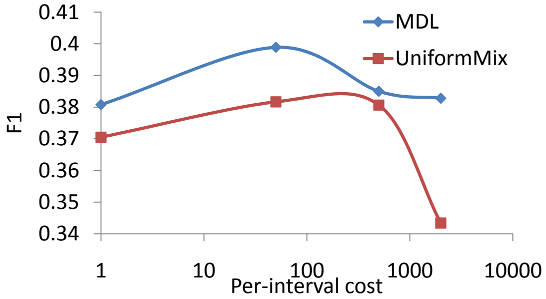


Figure 13: Comparing the MDL and UniformMix models for generating final intervals in the answer for increasing Segment cost.

6.6 Benefits of PCFG-based unit extractor

Here we evaluate different parsers for extracting units discussed in Section 5 and also study their impact on end-to-end query processing.

In order to estimate the extraction accuracy of unit parsers, we created a dataset of 617 table headers from our corpus that have been manually labeled with the correct units. In Figure 15 we plot the extraction accuracy of different parsers under varying settings. **Rule** is a rule-based parser that in-

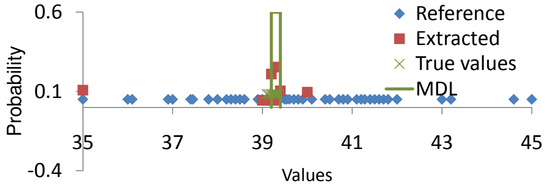


Figure 14: An example showing how a reference set of numbers helps the MDL approach select the correct intervals.

cludes among others the rules of Section 5.1 refined to label only unambiguous matches. This has an accuracy of only 40%, and almost all of this error is due to poor recall — the precision of the rule-based parser is 99%. In contrast, our proposed CFG-based parser that we call **QuantCFG** achieves an accuracy of 82%. We created another parser called **Sequence** that uses all the features of **QuantCFG** but not its grammar, and chooses the unit with highest score **w.f** among all word sequences that match QuTree. The drop in accuracy to 74% establishes the importance of the grammar. The next three bars establish the importance of features in Sections 5.2.2, 5.2.3, 5.2.4 by reporting accuracy of **QuantCFG** without various subsets of them.

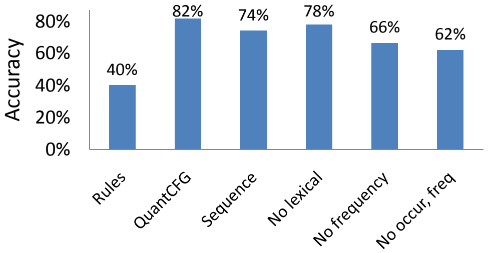


Figure 15: Comparing different parsers.

We next assess the impact of the unit extraction accuracy on the final answer quality in the end-to-end system. In Figure 16, we compare F1 scores of the answer on the three workloads on three different parsers described above: the rule-based parser, the feature-based sequence parser, and our QuantCFG parser. We observe that it is necessary to use parsers both with high precision and high recall for getting quality answers.

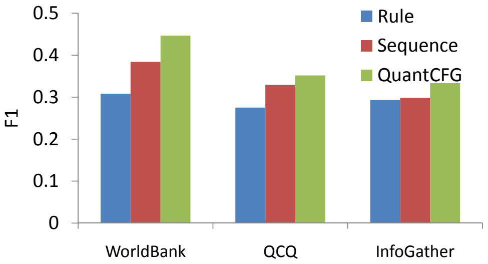


Figure 16: Comparing different parsers on their impact on the answer quality in the end to end system.

7. RELATED WORK

Moriceau [10] was among the earliest to formalize quantity search, and provide some initial notions of temporal trends and aggregation of values. A more extensive system was built by Wu and Marian [19, "W&M"]. Banerjee *et al.* [3, "QCQ"] proposed the quantity interval ranking problem. None of these exploited source HTML tables, none delayed per-snippet extraction decisions, and none proposed an inference procedure that collectively estimated snippet relevance, snippet extractions, and the value distribution. SCAD [1] collected quantities while satisfying domain-guided numeric constraints between them (e.g., a laptop screen is wider than it is tall). But SCAD did not use a unit extractor or consensus inference as in QEWT. Zhang *et al.* [20, "InfoGather"] also extract units and values from Web tables. They focus on identifying correspondences among tables based on column types. They do not model uncertain value distributions. Their aggregation/consensus is based on exact match of values, which we demonstrate as weaker than QEWT's value distribution model. InfoGather extracts column units using rules, which cannot handle compound units and noisy headers, unlike our PCFG unit extractor. Their "query" resembles a table completion task, with ∼100 entities, units, and scales explicitly provided. In contrast, QEWT is a robust, open-domain system for ad-hoc, single-entity queries.

Acknowledgments. This work was partly supported by research grants from the Indo-German Max Planck Centre for Computer Science (IMPECS) and from Yahoo! Research.

8. REFERENCES

[1] A. Bakalov, A. Fuxman, P. P. Talukdar, and S. Chakrabarti. SCAD: collective discovery of attribute values. In *WWW Conference*, pages 447–456, 2011.

[2] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Information Processing and Management*, 45(1):1–19, 2009.

[3] S. Banerjee, S. Chakrabarti, and G. Ramakrishnan. Learning to rank for quantity consensus queries. In *SIGIR Conference*, 2009.

[4] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1), 2009.

[5] M. Cornolti, P. Ferragina, and M. Ciaramita. A framework for benchmarking entity-annotation systems. In *WWW Conference*, pages 249–260, Rio de Janeiro, Brazil, 2013.

[6] E. Crestan and P. Pantel. Web-scale table census and classification. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, 2011.

[7] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. In *VLDB*, 2010.

[8] T.-Y. Liu. Learning to rank for information retrieval. In *Foundations and Trends in Information Retrieval*, volume 3, pages 225–331. Now Publishers, 2009.

[9] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[10] V. Moriceau. Numerical data integration for cooperative question-answering. In *EACL Workshop on Knowledge and Reasoning for Language Processing*, pages 42–49, 2006.

[11] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. In *Proc. of the 38th Int'l Conference on Very Large Databases (VLDB)*, 2012.

[12] J. Rissanen. Stochastic complexity in statistical inquiry. In *World Scientific Series in Computer Science*, volume 15. World Scientific, Singapore, 1989.

[13] S. Sarawagi. Information extraction. *FnT Databases*, 1(3), 2008.

[14] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge unifying WordNet and Wikipedia. In *WWW Conference*, pages 697–706. ACM Press, 2007.

[15] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *EMNLP*, July 2004.

[16] P. D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. In *ECML*, 2001.

[17] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9), 2011.

[18] Worldbank. http://data.worldbank.org/, 2014.

[19] M. Wu and A. Marian. Corroborating answers from multiple web sources. In *WebDB: Tenth International Workshop on the Web and Databases*, 2007.

[20] M. Zhang and K. Chakrabarti. Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD Conference*, 2013.