Curating and Searching the Annotated Web

Amit Singh, Sayali Kulkarni, Somnath Banerjee, Ganesh Ramakrishnan, Soumen Chakrabarti
IIT Bombay
ganesh@cse.iitb.ac.in, soumen@cse.iitb.ac.in

ABSTRACT

We demonstrate CSAW, a system for Curating and Searching the Annotated Web. CSAW annotates named entities and quantities in Web-scale text corpora, and, where confident, connects these annotations with entries in an entity and type catalog such as Wikipedia. The semistructured catalog, together with the unstructured corpus, forms a composite database that CSAW can then search using powerful reachability, proximity and aggregation primitives. Specifically, we can look for snippets with mentions of specific entities, entities of a specified type, quantities with specified types or units, find unions and intersections of snippet sets, and then aggregate evidence from snippet sets into ranked responses. Responses are not page URLs as in standard Web search, but ranked tables where the cells can be entity references, quantities, or token snippets. We will show a subset of CSAW's capabilities, and describe the beginnings of a next-generation Web search API that significantly extends the capabilities of APIs provided by popular search engines today.

1. INTRODUCTION

Despite immense progress in crawling and indexing the Web as well as ranking technology, the query experience has remained exactly the same as fifteen years back: type some string tokens into a text box, and get a ranked list of URLs as response. Several notable research prototypes [5, 4, 6] have sought to break this "syntax barrier" [3] on small corpora and specific tasks, but these have not become mainstream at Web scale.

Substantial advances have been made in two related technologies in the last decade: named entity recognition (NER) and entity resolution or disambiguation. Collectively, we call these information extraction [11]. Web search engines already annotate their enormous corpus with several named entities of coarse-grained types, such as persons, organizations, locations, dates, prices, etc. These annotations are exploited extensively by the ranking logic. E.g., `94720` in the query `pizza 94720` may be interpreted as a zip code. Responses to queries on (ambiguous) person names are diversified so as not to be dominated by a few people.

Notwithstanding these advances, the connection between the world of text and the semantic world of entities and relations is still tenuous in mainstream Web search [2].

- As far as publicly known, Web search engines annotate named entities from a few dozen types, but they do not attach token spans to entities in a suitable entity catalog.
- The lowest common denominator query language gives the user very little control over expressing queries that are in part *placeholders* for entities of given types and in part strings and other predicates to match e.g., *weight* of Nikon FM2N.
- The response is page-oriented rather than entity and relation oriented, which are often the responses desired. The onus is on the user to click through and inspect the page for the answer. (The response snippets often do not serve up the answer.)
- There is no precise, flexible and open policy of evidence aggregation across token segments rich in matched query words.

In this demo, we present an early preview of CSAW (Curating and Searching the Annotated Web), a system that addresses the issues listed above.

1.1 Data model

Ordinary IR systems index token strings and severely restrict query expressiveness, whereas relational databases demand intimate schema knowledge. Information extraction systems analyze unstructured text and populate structured databases that can be queried independent of the original corpus. Our premise is that the data model for next-generation search has to be somewhere in between: IR indices augmented with linkages to structured catalogs of entities and relationships. In our current architecture of CSAW,

- A document is modeled as a sequence of tokens.
- Some token segments are identified as potential *mentions* of named entities.
- One special kind of named entities is a *quantity*, possibly associated with a unit of measurement.
- Some tokens segments are associated with an *entity node ID* in an entity catalog, currently, Wikipedia. In a companion paper [9] we describe new algorithms for adding such annotations accurately.
- Wikipedia also provides a *category hierarchy*; each entity node can be attached with one or more categories.

For concreteness we will refer to Wikipedia in the rest of this paper, while making it clear that other entity catalogs may be used.

1.2 Query capabilities

This composite representation allows us to combine circumstantial evidence of association (textual proximity) with relatively clean structured data (inside the catalog) in potent ways. At a high level, queries are built out of the following primitives:

- Instantiating documents or snippets (collectively called **contexts**) where query tokens occur close together.
- Ditto for mentions of a given entity, or entities reachable from a given category.
- Intersecting context sets where query tokens or entity mentions co-occur.
- Extracting entities and quantities from qualifying con-

texts.

- Scoring and ranking entities, quantities and quantity intervals based on the number and quality of contexts where they occur.

Detailed examples follow in Section 2, suggesting a preliminary sketch of a query API.

As with SQL or XQuery, we do not expect typical end users to type queries in such a language. We expect query mining and NLP applications to "compile" information needs into our intermediate form [10, 8].

1.3 Response

Another broad departure from typical Web search is that the CSAW user expects not a list of URLs, but entities, quantities, or tables of entities, quantities and text fields. In case of select-like point queries, users may ask for the typical battery life of a laptop, or the phone number of a business. Search engines are beginning to appreciate the utility of simple attribute compilation interfaces, such as Google Squared. CSAW goes further; we can ask about the typical driving time between Paris and Nice, which is not an attribute of either entity. In addition, CSAW aggregates evidence or consensus over contexts to rank quantity intervals [1], and we are adding capabilities to aggregate evidence for discrete entities as well [7]. As just one example of what this can enable, we can get as response a table of physicists and musical instruments, sorted in decreasing order of evidence of association. More concrete examples are shown in Section 2.

2. USER EXPERIENCE

2.1 Basic notions

A query is either a simple query or a join query. A **simple query** has two parts:

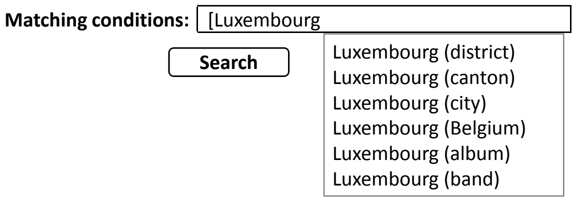
Target entities: A target entity is either a Wikipedia **category** or a quantity type (abbreviated **qtype**). Qtypes may be in a hierarchy, e.g. *Qtype:foot* is a refinement of *Qtype:LinearDistance*. Target entities are represented by a free variable of the form `?v`.

Match conditions: A match condition is a set of items drawn from three kinds of items.

Tokens: Single tokens or phrases represented as strings, e.g., `academy awards`.

Entities: References to entities, such as *Castro (city)* or *Jordan_algebra*. Although we write them as strings for clarity, they are internally represented as Wikipedia node IDs.

Quantities: A numeric literal, associated with a unit, e.g., *1250 INR*. Although written as a string, this is internally represented as a number and a standard unit, enabling approximate numeric matching and possibly unit conversion.

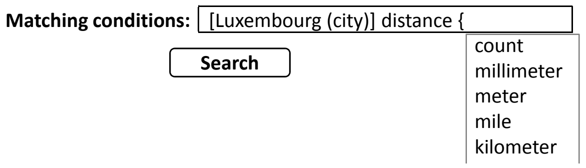

Matching conditions: [Luxembourg

Search

Luxembourg (district)
Luxembourg (canton)
Luxembourg (city)
Luxembourg (Belgium)
Luxembourg (album)
Luxembourg (band)

Figure 1: Wikipedia entity suggestion.

The purpose of a simple query is to extract contexts that have an instance of each target entity, and have high similarity or proximity to the match conditions. These contexts are then suitably aggregated to emit a sequence of response tuples. A **join query** is a collection of simple queries related through shared target entities. Examples follow shortly.


Matching conditions: [Luxembourg (city)] distance {

Search

count
millimeter
meter
mile
kilometer

Figure 2: Qtype suggestion.

2.2 Query hardening

CSAW does not assume that the user has intimate knowledge of Wikipedia entities or categories or quantity types. Instead, the query process proceeds in two steps. In the first step, the query is *hardened* through an autocompletion (also called query suggestion). Typing "[" triggers autocompletion to a Wikipedia entity or category via user selection from a drop down list. This facilitates the user to harden the keyword `Luxembourg` to the Wikipedia entity *Luxembourg (city)* (in the sense of a city) as shown in Figure 1. Similarly, typing "{" triggers autocompletion to a qtype, such as *meter, USD, foot* etc., as shown in Figure 2. In the examples that follow, Wikipedia entities will be marked within "[]" and qtypes marked with "{}".

2.3 Supported predicates and operators

CSAW currently supports the predicates IsA, SubCat, HasCat, HasUnit and InContext. The first three involve Wikipedia's entity and category hierarchy. IsA(e, C) means entity e is a direct instance of category C, i.e., $e \in C$. SubCat(C_1, C_2) means $C_1 \subseteq C_2$. HasCat$(e, C) \equiv e \in C_1 \subseteq \cdots \subseteq C$, shorthanded as $e \in^+ C$. InContext takes one context argument and a variable number of entities, quantities, tokens and phrases, and returns true if the latter all appear in the context. The width of a context is usually a certain window of tokens, a sentence, or a table cell etc. E.g., InContext$(s; \texttt{elephant}, \texttt{?h}) \wedge$ HasUnit$(\texttt{?h}, \textit{Qtype:foot})$ is true if the string literal "elephant" appears in a context s together with a quantity with unit *foot*. Note that *QType:foot* is already disambiguated apart from the elephant's foot. We will abbreviate this query as InContext$(s; \texttt{elephant}, \texttt{?h} \in \textit{QType:foot})$.

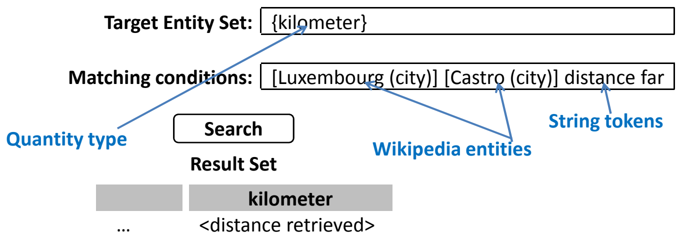
The most interesting operator in CSAW is Cons, which aggregates evidence ("consensus") over qualifying contexts. Suppose we wish to list scientists that are most likely to be associated with musical instruments. We first write

InContext$(\texttt{?c}; \texttt{?s} \in^+ \textit{Category:Scientists},$
$\texttt{?m} \in^+ \textit{Category:Musical instruments}, \texttt{played})$

to collect contexts, and then write down the aggregation Cons$(\texttt{?c})$ to collect tuples $\langle \texttt{?s}, \texttt{?m} \rangle$ across contexts `?c`, sorted in decreasing order of evidence. Consensus for quantities [1] may be achieved differently from collecting consensus for discrete entities [7].

2.4 Example queries

We describe some more queries supported by our system to illustrate our design.



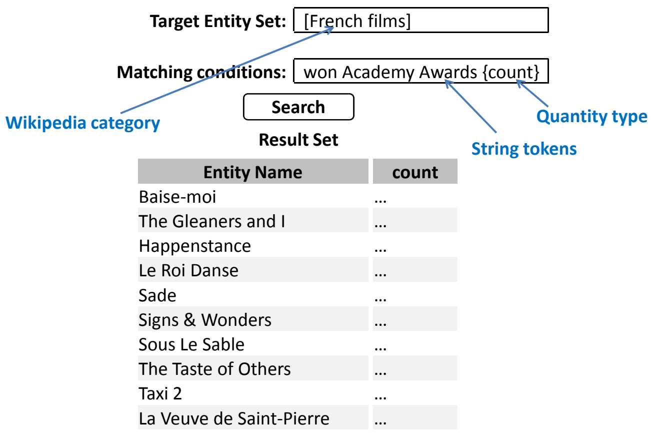
Target Entity Set: {kilometer}

Matching conditions: [Luxembourg (city)] [Castro (city)] distance far

Result Set

	kilometer
...	<distance retrieved>

Figure 3: Searching for a quantity.



Target Entity Set: [French films]

Matching conditions: won Academy Awards {count}

Result Set

Entity Name	count
Baise-moi	...
The Gleaners and I	...
Happenstance	...
Le Roi Danse	...
Sade	...
Signs & Wonders	...
Sous Le Sable	...
The Taste of Others	...
Taxi 2	...
La Veuve de Saint-Pierre	...

Figure 4: Compiling a single quantity related to elements of a specified category.

Single target quantity. Cons(?c) where
InContext(?c; ?d ∈ +*Qtype:LinearDistance*,
Castro (city), *Luxembourg (city)*, `distance`, `far`)
finds the distance between cities Castro and Luxembourg. See Figure 3.

Multiple targets expressed as category. Suppose we want to compile a table with a row for every French film, with the title and number of Academy Awards won by that film as the two columns. The clauses of our query are

- ?f $\in^+$ *Category:French Films*,
- ?a ∈ *Qtype:Number*,
- InContext(?c; ?f, ?a, `academy awards`, `won`),

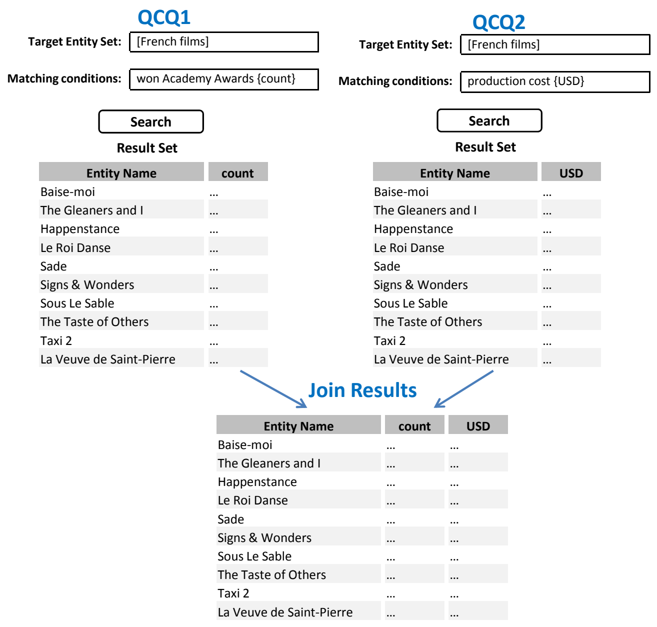
followed by the "group-by" Cons(?c), resulting in a table with two columns. See Figure 4.

Subqueries and joins. We may want to compile the production cost and number of academy awards for French films. In general these quantities may come from different context, even different pages. Effectively, two subqueries are issued with shared free variables ?f, ?a, ?p, the clauses being

- ?f $\in^+$ *Category:French Film*,
- ?a ∈ *Qtype:Number*, ?p ∈ *Qtype:MoneyAmount*,
- InContext(?c1; ?f, ?a, `academy awards`, `won`),
- InContext(?c2; ?f, ?p, `production cost`, `budget`),

ending with Cons(?c1, ?c2), to emit a sequence of ⟨?f, ?a, ?p⟩ tuples. See Figure 5.

Since we will be extracting and aggregating information from largely unstructured sources, in general, the output is a probabilistic or uncertain database, i.e., tuples should be accompanied with some notion of confidence, so that, e.g., they can be materialized and reused in other queries. Uncertain databases is a major area of contemporary database



QCQ1 — Target Entity Set: [French films]; Matching conditions: won Academy Awards {count}; Result Set

Entity Name	count
Baise-moi	...
The Gleaners and I	...
Happenstance	...
Le Roi Danse	...
Sade	...
Signs & Wonders	...
Sous Le Sable	...
The Taste of Others	...
Taxi 2	...
La Veuve de Saint-Pierre	...

QCQ2 — Target Entity Set: [French films]; Matching conditions: production cost {USD}; Result Set

Entity Name	USD
Baise-moi	...
The Gleaners and I	...
Happenstance	...
Le Roi Danse	...
Sade	...
Signs & Wonders	...
Sous Le Sable	...
The Taste of Others	...
Taxi 2	...
La Veuve de Saint-Pierre	...

Join Results

Entity Name	count	USD
Baise-moi	...	...
The Gleaners and I	...	...
Happenstance	...	...
Le Roi Danse	...	...
Sade	...	...
Signs & Wonders	...	...
Sous Le Sable	...	...
The Taste of Others	...	...
Taxi 2	...	...
La Veuve de Saint-Pierre	...	...

Figure 5: Subqueries and joins.

research and we plan to draw on this area to further refine our system.

3. SYSTEM DESCRIPTION

The pages crawled from the Web are annotated using the quantity and Wikipedia annotators. The two annotation engines generate separate Lucene indices which are merged and stored in a distributed manner using Katta (`http://katta.sourceforge.net/`). The search API uses this distributed index to filter pages based on the Wikipedia entities appearing in the query keywords. Filtered results are then fed to the QCQ component that returns the final ranked interval list. See Figure 6.

3.1 Annotation Process

3.1.1 *Quantity Annotator*

The quantity annotator detects quantity mentions and classifies them into one of several quantity types such as count, date, mile, hour, dollar etc. Quantity mentions occur in diverse forms like \$10, USD 10, 000, 10–20 feet, 10^6 km, ten million litres. The system uses a set of pre-defined rules for each quantity type to annotate token spans. This annotation process can be done at the rate of 900 documents/second on one 4-core Intel Xeon CPU (2.50GHz).

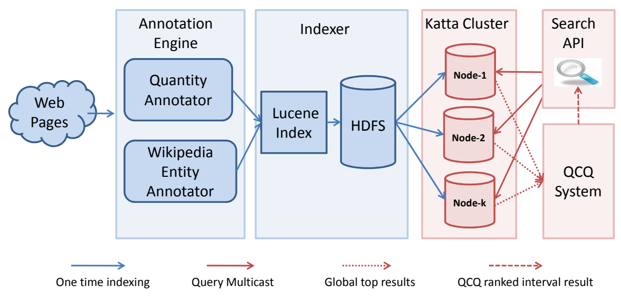


Figure 6: System Architecture

3.1.2 Wikipedia Entity Annotator

The entity annotation component consists of tagging the Wikipedia entities appearing in the plain text. There are two main stages in this process: spotting and tagging.

Spotting phase. The Wikipedia version we use contains about 2.3 million Wikipedia entities. These entities are used to build a trie (prefix tree) generated using the Webgraph[1] framework. Documents to be annotated are tokenized and token sequences that maximally match an entity in the trie are identified. These token sequences are termed as *spots*. For our demonstration, we restrict our entity set to a subset of the entities picked from a few Wikipedia categories.

Tagging phase. Once all the candidate spots on a page are identified, a label is chosen for each spot independent of the others, without any collective information, using the LOCAL algorithm [9]. Some of the identified spots could be dropped in the annotation phase for the lack of annotation confidence. We plan to use the collective inference techniques [9] in the future. The LOCAL algorithm was reported to yield a maximum F1 measure of 60.49 at a precision of 59.24, when evaluated on about a hundred news articles with about 20,000 spots to annotate. Precision can be gained at the cost of recall to achieve a precision of 70.22% at a recall of 41.75% and 79.57% at a recall of 14.63%. In our annotation process, we decided to be conservative about the annotations we make in order to improve our precision at the cost of low recall. The entire annotation process using LOCAL can be done at the rate of 10 documents/second on one 4 core Intel Xeon CPU (2.50GHz). The main bottleneck in the processing pipeline is random disk I/O involved in accessing Wikipedia feature vectors (~10GB) since they cannot entirely fit in RAM.

3.1.3 Indexing

Quantity and entity annotations generated by the annotation components are stored as Lucene indexes distributed across multiple machines. We have adapted the Katta framework to answer the queries over this distributed index.

3.2 Query Processing

The query is first decomposed into simple subqueries. For each simple query, the target entities and quantities define a set of contexts to inspect. These contexts are constructed by scanning posting lists for entity and quantity types together with posting lists for ordinary words [4].

In CSAW, the posting scan is complicated by the distributed nature of the Katta index. Candidate lists are built on each Katta node and these are merged to a central list.

Contexts in the high-quality list are scored and then aggregated to develop a confidence score for every candidate tuple. If needed, these are then joined across subqueries to produce the final answer.

Acknowledgment. Thanks to D. Grosvenor, K. Ramanathan, S. Sarawagi and S. Sudarshan for helpful discussions.

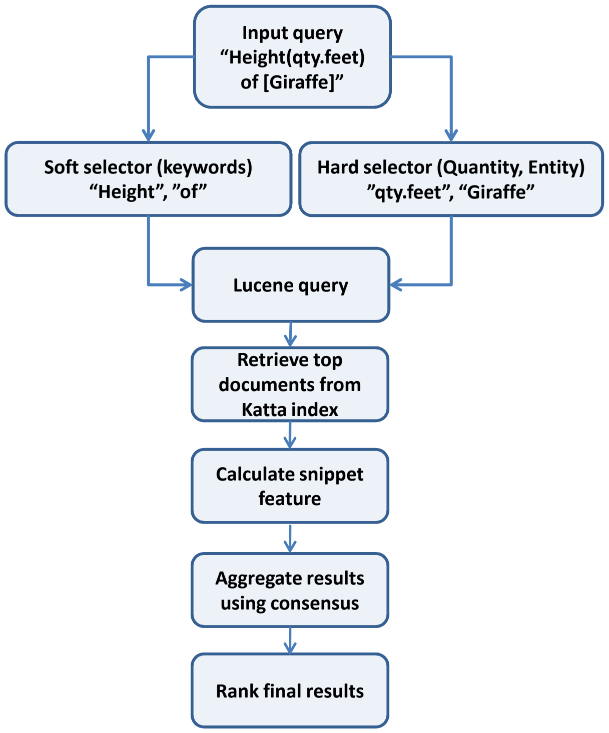

Input query
"Height(qty.feet) of [Giraffe]"
Soft selector (keywords)
"Height", "of"
Hard selector (Quantity, Entity)
"qty.feet", "Giraffe"
Lucene query
Retrieve top documents from Katta index
Calculate snippet feature
Aggregate results using consensus
Rank final results

Figure 7: Query execution.

4. REFERENCES

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

[2] A. Broder, S. Chakrabarti, J. Ribas, V. Singh, and N. Weininger. Web search APIs: The next generation. In *WWW Conference*, Apr. 2009. Panel discussion.

[3] S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *ECML/PKDD*, pages 9–16, 2004. Invited talk.

[4] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW Conference*, Edinburgh, May 2006.

[5] T. Cheng, X. Yan, and K. C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB Conference*, pages 387–398, Sept. 2007.

[6] G. Kasneci, F. M. Suchanek, G. Ifrim, S. Elbassuoni, M. Ramanath, and G. Weikum. NAGA: harvesting, searching and ranking knowledge. In *SIGMOD Conference*, pages 1285–1288. ACM, 2008.

[7] J. Ko, E. Nyberg, and L. Si. A probabilistic graphical model for joint answer ranking in question answering. In *SIGIR Conference*, pages 343–350, 2007.

[8] V. Krishnan, S. Das, and S. Chakrabarti. Enhanced answer type inference from questions using sequential models. In *EMNLP/HLT*, pages 315–322, 2005.

[9] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of Wikipedia entities in Web text. In *SIGKDD Conference*, 2009.

[10] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Intelligent User Interfaces*, pages 149–157, Miami, 2003. ACM.

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

[1] `http://webgraph.dsi.unimi.it/`