KnowItNow: Fast, Scalable Information Extraction from the Web

Michael J. Cafarella, Doug Downey, Stephen Soderland, Oren Etzioni
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
{mjc,ddowney,soderlan,etzioni}@cs.washington.edu

Abstract

Numerous NLP applications rely on search-engine queries, both to extract information from and to compute statistics over the Web corpus. But search engines often limit the number of available queries. As a result, query-intensive NLP applications such as Information Extraction (IE) distribute their query load over several days, making IE a slow, offline process.

This paper introduces a novel architecture for IE that obviates queries to commercial search engines. The architecture is embodied in a system called KNOWITNOW that performs high-precision IE in minutes instead of days. We compare KNOWITNOW experimentally with the previously-published KNOWITALL system, and quantify the tradeoff between recall and speed. KNOWITNOW's extraction rate is two to three orders of magnitude higher than KNOWITALL's.

1 Background and Motivation

Numerous modern NLP applications use the Web as their corpus and rely on queries to commercial search engines to support their computation (Turney, 2001; Etzioni et al., 2005; Brill et al., 2001). Search engines are extremely helpful for several linguistic tasks, such as computing usage statistics or finding a subset of web documents to analyze in depth; however, these engines were not designed as building blocks for NLP applications. As a result, the applications are forced to issue literally millions of queries to search engines, which limits the speed, scope, and scalability of the applications. Further, the applications must often then fetch some web documents, which at scale can be very time-consuming.

In response to heavy programmatic search engine use, Google has created the "Google API" to shunt programmatic queries away from Google.com and has placed hard quotas on the number of daily queries a program can issue to the API. Other search engines have also introduced mechanisms to limit programmatic queries, forcing applications to introduce "courtesy waits" between queries and to limit the number of queries they issue.

To understand these efficiency problems in more detail, consider the KNOWITALL information extraction system (Etzioni et al., 2005). KNOWITALL has a generate-and-test architecture that extracts information in two stages. First, KNOWITALL utilizes a small set of domain-independent extraction patterns to *generate* candidate facts (*cf.* (Hearst, 1992)). For example, the generic pattern "NP1 such as NPList2" indicates that the head of each simple noun phrase (NP) in NPList2 is a member of the class named in NP1. By instantiating the pattern for class `City`, KNOWITALL extracts three candidate cities from the sentence: "We provide tours to cities such as Paris, London, and Berlin." Note that it must also fetch each document that contains a potential candidate.

Next, extending the PMI-IR algorithm (Turney, 2001), KNOWITALL automatically *tests* the plausibility of the candidate facts it extracts using *pointwise mutual information* (PMI) statistics computed from search-engine hit counts. For example, to assess the likelihood that "Yakima" is a city, KNOWITALL will compute the PMI between Yakima and a set of k *discriminator phrases* that tend to have high mutual information with city names (*e.g.*, the simple phrase "city"). Thus, KNOWITALL requires at least k search-engine queries for every candidate extraction it assesses.

Due to KNOWITALL's dependence on search-engine queries, large-scale experiments utilizing KNOWITALL *take days and even weeks to complete*, which makes research using KNOWITALL slow and cumbersome. Private access to Google-scale infrastructure would provide

Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), pages 563–570, Vancouver, October 2005.

sufficient access to search queries, but at prohibitive cost, and the problem of fetching documents (even if from a cached copy) would remain (as we discuss in Section 2.1). Is there a feasible alternative Web-based IE system? If so, what size Web index and how many machines are required to achieve reasonable levels of precision/recall? What would the architecture of this IE system look like, and how fast would it run?

To address these questions, this paper introduces a novel architecture for web information extraction. It consists of two components that supplant the generate-and-test mechanisms in KNOWITALL. To generate extractions rapidly we utilize our own specialized search engine, called the Bindings Engine (or BE), which efficiently returns bindings in response to variabilized queries. For example, in response to the query *"Cities such as ProperNoun(Head(⟨NounPhrase⟩))"*, BE will return a list of proper nouns likely to be city names. To assess these extractions, we use URNS, a combinatorial model, which estimates the probability that each extraction is correct without using any additional search engine queries.[1] For further efficiency, we introduce an approximation to URNS, based on frequency of extractions' occurrence in the output of BE, and show that it achieves comparable precision/recall to URNS.

Our contributions are as follows:

1. We present a novel architecture for Information Extraction (IE), embodied in the KNOWITNOW system, which does not depend on Web search-engine queries.

2. We demonstrate experimentally that KNOWITNOW is the first system able to extract tens of thousands of facts from the Web in minutes instead of days.

3. We show that KNOWITNOW's extraction rate is two to three orders of magnitude greater than KNOWITALL's, but this increased efficiency comes at the cost of reduced recall. We quantify this tradeoff for KNOWITNOW's 60,000,000 page index and extrapolate how the tradeoff would change with larger indices.

Our recent work has described the BE search engine in detail (Cafarella and Etzioni, 2005), and also analyzed the URNS model's ability to compute accurate probability estimates for extractions (Downey et al., 2005). However, this is the first paper to investigate the composition of these components to create a fast IE system, and to compare it experimentally to KNOWITALL in terms of time, recall, precision, and extraction rate. The frequency-based approximation to URNS and the demonstration of its success are also new.

The remainder of the paper is organized as follows. Section 2 provides an overview of BE's design. Section 3 describes the URNS model and introduces an efficient approximation to URNS that achieves similar precision/recall. Section 4 presents experimental results. We conclude with related and future work in Sections 5 and 6.

2 The Bindings Engine

This section explains how relying on standard search engines leads to a bottleneck for NLP applications, and provides a brief overview of the Bindings Engine (BE)—our solution to this problem. A comprehensive description of BE appears in (Cafarella and Etzioni, 2005).

Standard search engines are computationally expensive for IE and other NLP tasks. IE systems issue multiple queries, downloading all pages that potentially match an extraction rule, and performing expensive processing on each page. For example, such systems operate roughly as follows on the query ("cities such as ⟨*NounPhrase*⟩"):

1. Perform a traditional search engine query to find all URLs containing the non-variable terms (*e.g.*, "cities such as")

2. For each such URL:
   (a) obtain the document contents,
   (b) find the searched-for terms ("cities such as") in the document text,
   (c) run the noun phrase recognizer to determine whether text following "cities such as" satisfies the linguistic type requirement,
   (d) and if so, return the string

We can divide the algorithm into two stages: obtaining the list of URLs from a search engine, and then processing them to find the ⟨*NounPhrase*⟩ bindings. Each stage poses its own scalability and speed challenges. The first stage makes a query to a commercial search engine; while the number of available queries may be limited, a single one executes relatively quickly. The second stage fetches a large number of documents, each fetch likely resulting in a random disk seek; this stage executes slowly. Naturally, this disk access is slow regardless of whether it happens on a locally-cached copy or on a remote document server. The observation that the second stage is slow, even if it is executed locally, is important because it shows that merely operating a "private" search engine does not solve the problem (see Section 2.1).

The Bindings Engine supports queries containing *typed variables* (such as *NounPhrase*) and

[1] In contrast, PMI-IR, which is built into KNOWITALL, requires multiple search engine queries to assess each potential extraction.

string-processing functions (such as "head(X)" or "ProperNoun(X)") as well as standard query terms. BE processes a variable by returning every possible string in the corpus that has a matching type, and that can be substituted for the variable and still satisfy the user's query. If there are multiple variables in a query, then all of them must simultaneously have valid substitutions. (So, for example, the query "*<NounPhrase>* is located in *<NounPhrase>*" only returns strings when noun phrases are found on both sides of "is located in".) We call a string that meets these requirements a *binding* for the variable in question. These queries, and the bindings they elicit, can usefully serve as part of an information extraction system or other common NLP tasks (such as gathering usage statistics). Figure 1 illustrates some of the queries that BE can handle.

president Bush *<Verb>*
cities such as *ProperNoun(Head(<NounPhrase>))*
<NounPhrase> is the CEO of *<NounPhrase>*

Figure 1: **Examples of queries that can be handled by** BE**. Queries that include typed variables and string-processing functions allow NLP tasks to be done efficiently without downloading the original document during query processing.**

BE's novel *neighborhood index* enables it to process these queries with $O(k)$ random disk seeks and $O(k)$ serial disk reads, where k is the number of non-variable terms in its query. As a result, BE can yield orders of magnitude speedup as shown in the asymptotic analysis later in this section. The neighborhood index is an augmented inverted index structure. For each term in the corpus, the index keeps a list of documents in which the term appears and a list of positions where the term occurs, just as in a standard inverted index (Baeza-Yates and Ribeiro-Neto, 1999). In addition, the neighborhood index keeps a list of left-hand and right-hand *neighbors* at each position. These are adjacent text strings that satisfy a recognizer for one of the target types, such as *NounPhrase*.

As with a standard inverted index, a term's list is processed from start to finish, and can be kept on disk as a contiguous piece. The relevant string for a variable binding is included directly in the index, so there is no need to fetch the source document (thus causing a disk seek). Expensive processing such as part-of-speech tagging or shallow syntactic parsing is performed only once, while building the index, and is not needed at query time. It is important to note that simply preprocessing the corpus and placing the results in a database would not avoid disk seeks, as we would still have to explicitly fetch these results. The run-time efficiency of the neighborhood index comes from integrating the results of corpus processing with the inverted index (which determines which of those results are relevant).

	Query Time	Index Space
BE	$O(k)$	$O(N)$
Standard engine	$O(k+B)$	$O(N)$

Table 1: BE **yields considerable savings in query time over a standard search engine.** k **is the number of concrete terms in the query,** B **is the number of variable bindings found in the corpus, and** N **is the number of documents in the corpus.** N **and** B **are typically extremely large, while** k **is small.**

The neighborhood index avoids the need to return to the original corpus, but it can consume a large amount of disk space, as parts of the corpus text are folded into the index several times. To conserve space, we perform simple dictionary-lookup compression of strings in the index. The storage penalty will, of course, depend on the exact number of different types added to the index. In our experiments, we created a useful IE system with a small number of types (including *NounPhrase*) and found that the neighborhood index increased disk space only four times that of a standard inverted index.

Asymptotic Analysis:

In our asymptotic analysis of BE's behavior, we count query time as a function of the number of random disk seeks, since these seeks dominate all other processing tasks. Index space is simply the number of bytes needed to store the index (not including the corpus itself).

Table 1 shows that BE requires only $O(k)$ random disk seeks to process queries with an arbitrary number of variables whereas a standard engine takes $O(k+B)$, where k is the number of concrete query terms, and B is the number of bindings found in a corpus of N documents. Thus, BE's performance is the same as that of a standard search engine for queries containing only concrete terms. For variabilized queries, N may be in the billions and B will tend to grow with N. In our experiments, eliminating the B term from our query processing time has resulted in speedups of two to three orders of magnitude over a standard search engine. The speedup is at the price of a small constant multiplier to index size.

2.1 Discussion

While BE has some attractive properties for NLP computations, is it necessary? Could fast, large-scale information extraction be achieved merely by operating a "private" search engine?

The release of open-source search engines such as Nutch[2], coupled with the dropping price of CPUs and

[2]http://lucene.apache.org/nutch/

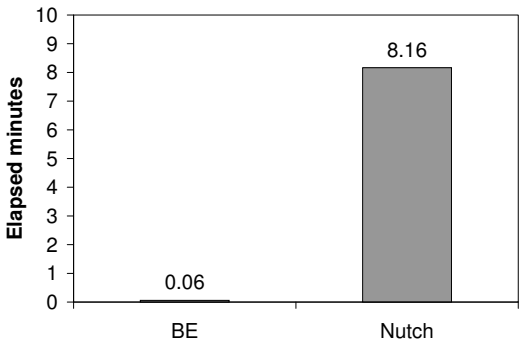

Figure 2: **Average time to return the relevant bindings in response to a set of queries was 0.06 CPU minutes for** BE**, compared to 8.16 CPU minutes for the comparable processing on Nutch. This is a 134-fold speed up. The CPU resources, network, and index size were the same for both systems.**

disks, makes it feasible for NLP researchers to operate their own large-scale search engines. For example, Turney operates a search engine with a terabyte-sized index of Web pages, running on a local eight-machine Beowulf cluster (Turney, 2004). Private search engines have two advantages. First, there is no query quota or need for "courtesy waits" between queries. Second, since the engine is local, network latency is minimal.

However, to support IE, we must also execute the second stage of the algorithm (see the beginning of this section). In this stage, each document that matches a query has to be retrieved from an arbitrary location on a disk.[3] Thus, the number of random disk seeks scales linearly with the number of documents retrieved. Moreover, many NLP applications require the extraction of strings matching particular syntactic or semantic types from each page. The lack of linguistic data in the search engine's index means that many pages are fetched only to be discarded as irrelevant.

To quantify the speedup due to BE, we compared it to a standard search index built on the open-source Nutch engine. All of our Nutch and BE experiments were carried out on the same corpus of 60 million Web pages and were run on a cluster of 23 dual-Xeon machines, each with two local 140 Gb disks and 4 Gb of RAM. We set all configuration values to be exactly the same for both Nutch and BE. BE gave a 134-fold speed up on average query processing time when compared to the same queries with the Nutch index, as shown in Figure 2.

[3]Moving the disk head to an arbitrary location on the disk is a mechanical operation that takes about 5 milliseconds on average.

3 The URNS Model

To realize the speedup from BE, KNOWITNOW must also avoid issuing search engine queries to validate the correctness of each extraction, as required by PMI computation. We have developed a probabilistic model obviating search-engine queries for assessment. The intuition behind this model is that correct instances of a class or relation are likely to be extracted repeatedly, while random errors by an IE system tend to have low frequency for each distinct incorrect extraction.

Our probabilistic model, which we call URNS, takes the form of a classic "balls-and-urns" model from combinatorics. We think of IE abstractly as a generative process that maps text to extractions. Each extraction is modeled as a labeled ball in an urn. A *label* represents either an instance of the target class or relation, or represents an error. The information extraction process is modeled as repeated draws from the urn, with replacement.

Formally, the parameters that characterize an urn are:

- C – the set of unique target labels; $|C|$ is the number of unique target labels in the urn.
- E – the set of unique error labels; $|E|$ is the number of unique error labels in the urn.
- $num(b)$ – the function giving the number of balls labeled by b where $b \in C \cup E$. $num(B)$ is the multi-set giving the number of balls for each label $b \in B$.

The goal of an IE system is to discern which of the labels it extracts are in fact elements of C, based on repeated draws from the urn. Thus, the central question we are investigating is: *given that a particular label x was extracted k times in a set of n draws from the urn, what is the probability that $x \in C$?* We can express the probability that an element extracted k of n times is of the target relation as follows.

$$P(x \in C | x \text{ appears } k \text{ times in } n \text{ draws}) = \frac{\sum_{r \in num(C)} (\frac{r}{s})^k (1 - \frac{r}{s})^{n-k}}{\sum_{r' \in num(C \cup E)} (\frac{r'}{s})^k (1 - \frac{r'}{s})^{n-k}} \quad (1)$$

where s is the total number of balls in the urn, and the sum is taken over possible repetition rates r.

A few numerical examples illustrate the behavior of this equation. Let $|C| = |E| = 2,000$ and assume for simplicity that all labels are repeated on the same number of balls ($num(c_i) = R_C$ for all $c_i \in C$, and $num(e_i) = R_E$ for all $e_i \in E$). Assume that the extraction rules have precision $p = 0.9$, which means that $R_C = 9 \times R_E$ — target balls are nine times as common in the urn as error balls. Now, for $k = 3$ and $n = 10,000$ we have $P(x \in C) = 93.0\%$. Thus, we see that a small number of repetitions can yield high confidence in an extraction. However, when the sample size increases so that

$n = 20,000$, and the other parameters are unchanged, then $P(x \in C)$ drops to 19.6%. On the other hand, if C balls repeat much more frequently than E balls, say $R_C = 90 \times R_E$ (with $|E|$ set to 20,000, so that p remains unchanged), then $P(x \in C)$ rises to 99.9%.

The above examples enable us to illustrate the advantages of URNS over the noisy-or model used in previous work. The noisy-or model assumes that each extraction is an independent assertion that the extracted label is "true," an assertion that is correct a fraction p of the time. The noisy-or model assigns the following probability to extractions:

$$P_{noisy-or}(x \in C | x \text{ appears } k \text{ times}) = 1 - (1-p)^k$$

Therefore, the noisy-or model will assign the same probability — 99.9% — in *all three* of the above examples, although this is only correct in the case for which $n = 10,000$ and $R_C = 90 \times R_E$. As the other two examples show, for different sample sizes or repetition rates, the noisy-or model can be highly inaccurate. This is not surprising given that the noisy-or model ignores the sample size and the repetition rates.

URNS uses an EM algorithm to estimate its parameters, and currently the algorithm takes roughly three minutes to terminate.[4] Fortunately, we determined experimentally that we can approximate URNS's precision and recall using a far simpler frequency-based assessment method. This is true because good precision and recall merely require an appropriate *ordering* of the extractions for each relation, and not accurate probabilities for each extraction. For unary relations, we use the simple approximation that items extracted more often are more likely to be true, and order the extractions from most to least extracted. For binary relations like `CapitalOf(X,y)`, in which we extract several different candidate capitals `y` for each known country `X`, we use a smoothed frequency estimate to order the extractions. Let $freq(\texttt{R}(\texttt{X},\texttt{y}))$ denote the number of times that the binary relation $\texttt{R}(\texttt{X},\texttt{y})$ is extracted; we define:

$$smoothed\_freq(\texttt{R}(\texttt{X},\texttt{y})) = \frac{freq(\texttt{R}(\texttt{X},\texttt{y}))}{\max_{y'} freq(\texttt{R}(\texttt{X},\texttt{y}')) + 1}$$

We found that sorting by smoothed frequency (in descending order) performed better than simply sorting by $freq$ for relations $\texttt{R}(\texttt{X},\texttt{y})$ in which different known `X` values may have widely varying Web presence.

Unlike URNS, our frequency-based assessment does not yield accurate probabilities to associate with each extraction, but for the purpose of returning a ranked list of high-quality extractions it is comparable to URNS (see Figures 3 through 6), and it has the advantage of being much faster. Thus, in the experiments reported on below, we use frequency-based assessment as part of KNOWITNOW.

[4]This code has not been optimized at all. We believe that we can easily reduce its running time to less than a minute on average, and perhaps substantially more.

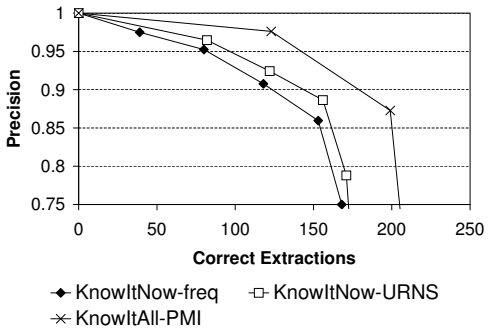

Figure 3: `Country`**: KNOWITALL maintains somewhat higher precision than KNOWITNOW throughout the recall-precision curve.**

4 Experimental Results

This section contrasts the performance of KNOWITNOW and KNOWITALL experimentally. Before considering the experiments in detail, we note that a key advantage of KNOWITNOW is that it does not make *any* queries to Web search engines. As a result, KNOWITNOW's scale is not limited by a query quota, though it is limited by the size of its index.

We report on the following metrics:

- **Recall:** how many distinct extractions does each system return at high precision?[5]
- **Time:** how long did each system take to produce and rank its extractions?
- **Extraction Rate:** how many distinct high-quality extractions does the system return per minute? The extraction rate is simply recall divided by time.

We contrast KNOWITALL and KNOWITNOW's precision/recall curves in Figures 3 through 6. We compared KNOWITNOW with KNOWITALL on four relations: `Corp`, `Country`, `CeoOf(Corp,Ceo)`, and `CapitalOf(Country,City)`. The unary relations were chosen to examine the difference between a relation with a small number of correct instances (`Country`) and one with a large number of extractions (`Corp`). The binary relations were chosen to cover both functional relations (`CapitalOf`) and set-valued relations (`CeoOf`—we treat former CEOs as correct instances of the relation).

[5]Since we cannot compute "true recall" for most relations on the Web, the paper uses the term "recall" to refer to the size of the set of facts extracted.

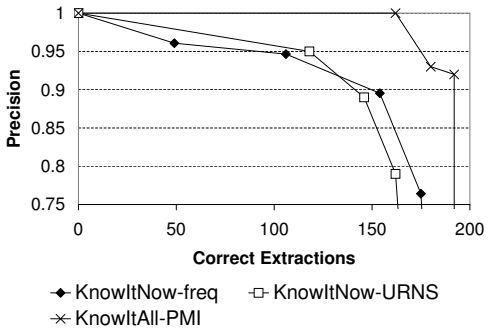

Figure 4: `CapitalOf`**:** KNOWITNOW **does nearly as well as** KNOWITALL**, but has more difficulty than** KNOWITALL **with sparse data for capitals of more obscure countries.**

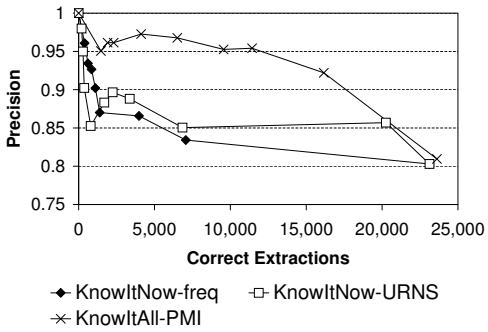

Figure 5: `Corp`**:** KNOWITALL**'s PMI assessment maintains high precision.** KNOWITNOW **has low recall up to precision 0.85, then catches up with** KNOWITALL**.**

For the two unary relations, both systems created extraction rules from eight generic patterns. These are hyponym patterns like "NP1 {,} such as NPList2" or "NP2 {,} and other NP1", which extract members of NPList2 or NP2 as instances of NP1. For the binary relations, the systems instantiated rules from four generic patterns. These are patterns for a generic "of" relation. They are "NP1 , rel of NP2", "NP1 the rel of NP2", "rel of NP2 , NP1", and "NP2 rel NP1". When rel is instantiated for `CeoOf`, these patterns become "NP1 , CEO of NP2" and so forth.

Both KNOWITNOW and KNOWITALL merge extractions with slight variants in the name, such as those differing only in punctuation or whitespace, or in the presence or absence of a corporate designator. For binary extractions, CEOs with the same last name and same company were also merged. Both systems rely on the OpenNlp maximum-entropy part-of-speech tagger and chunker (Ratnaparkhi, 1996), but KNOWITALL applies them to pages downloaded from the Web based on the results of Google queries, whereas KNOWITNOW applies them once to crawled and indexed pages.[6] Overall, each of the above elements of KNOWITALL and KNOWITNOW are the same to allow for controlled experiments.

Whereas KNOWITNOW runs a small number of variabilized queries (one for each extraction pattern, for each relation), KNOWITALL requires a stopping criterion. Otherwise, KNOWITALL will continue to query Google and download URLs found in its result pages over many days and even weeks. We allowed a total of 6 days of search time for KNOWITALL, allocating more search for the relations that continued to be most productive. For `CeoOf` KNOWITNOW returned all pairs of `Corp,Ceo` in its corpus; KNOWITALL searched for CEOs of a random selection of 10% of the corporations it found, and we projected the total extractions and search effort for all corporations. For `CapitalOf`, both KNOWITNOW and KNOWITALL looked for capitals of a set of 195 countries.

Table 2 shows the number of queries, search time, distinct correct extractions at precision 0.8, and extraction rate for each relation. Search time for KNOWITNOW is measured in seconds and search time for KNOWITALL is measured in hours. The number of extractions per minute counts the distinct correct extractions. Since we limit KNOWITALL to one Google query per second, the time for KNOWITALL is proportional to the number of queries. KNOWITNOW's extraction rate is from 275 to 4,707 times that of KNOWITALL at this level of precision.

While the number of distinct correct extractions from KNOWITNOW at precision 0.8 is roughly comparable to that of 6 days search effort from KNOWITALL, the situation is different at precision 0.9. KNOWITALL's PMI assessor is able to maintain higher precision than KNOWITNOW's frequency-based assessor. The number of correct corporations for KNOWITNOW drops from 23,128 at precision 0.8 to 1,116 at precision 0.9. KNOWITALL is able to identify 17,620 correct corporations at precision 0.9. Even with the drop in recall, KNOWITNOW's extraction rate is still 305 times higher than KNOWITALL's. The reason for KNOWITNOW's difficulty at precision 0.9 is due to extraction errors that occur with high frequency, particularly generic references to companies ("the Seller is a corporation ...", "corporations such as Banks", etc.) and truncation of certain company names by the extraction rules. The more expensive PMI-based assessment was not fooled by these systematic extraction errors.

Figures 3 through 6 show the recall-precision curves for KNOWITNOW with URNS assessment, KNOWITNOW with the simpler frequency-based assessment, and

[6] Our time measurements for KNOWITALL are not affected by the tagging and chunking time because it is dominated by time required to query Google, waiting a second between queries.

	Google Queries		Time		Extractions		Extractions per minute		
	NOW	ALL	NOW (sec)	ALL (hrs)	NOW	ALL	NOW	ALL	ratio
Corp	0 (16)	201,878	42	56.1	23,128	23,617	33,040	7.02	4,707
Country	0 (16)	35,480	42	9.9	161	203	230	0.34	672
CeoOf	0 (6)	263,646	51	73.2	2,402	5,823	2,836	1.33	2,132
CapitalOf	0 (6)	17,216	55	4.8	169	192	184	0.67	275

Table 2: **Comparison of** KNOWITNOW **with** KNOWITALL **for four relations, showing number of Google queries (local** BE **queries in parentheses), search time, correct extractions at precision 0.8, and extraction rate (the number of correct extractions at precision 0.8 per minute of search). Overall,** KNOWITNOW **took a total of slightly over 3 minutes as compared to a total of 6 days of search for** KNOWITALL**.**

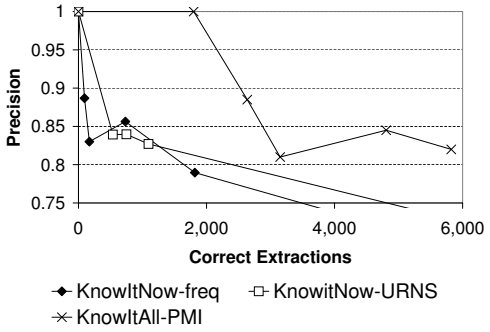

Figure 6: `CeoOf`**:** KNOWITNOW **has difficulty distinguishing low frequency correct extractions from noise.** KNOWITALL **is able to cope with the sparse data more effectively.**

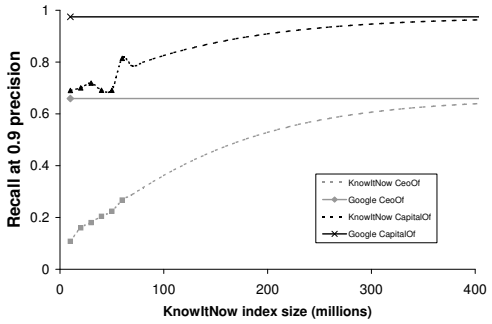

Figure 7: **Projections of recall (at precision 0.9) as a function of** KNOWITNOW **index size. At 400 million pages,** KNOWITNOW**'s recall rapidly approaches the recall achieved by** KNOWITALL **using roughly 300,000 Google queries.**

KNOWITALL with PMI-based assessment. For each of the four relations, PMI is able to maintain a higher precision than either frequency-based or URNS assessment. URNS and frequency-based assessment give roughly the same levels of precision.

For the relations with a small number of correct instances, `Country` and `CapitalOf`, KNOWITNOW is able to identify 70-80% as many instances as KNOWITALL at precision 0.9. In contrast, `Corp` and `CeoOf` have a huge number of correct instances and a long tail of low frequency extractions that KNOWITNOW has difficulty distinguishing from noise. Over one fourth of the corporations found by KNOWITALL had Google hit counts less than 10,500, a sparseness problem that was exacerbated by KNOWITNOW's limited index size.

Figure 7 shows projected recall from larger KNOWITNOW indices, fitting a sigmoid curve to the recall from index size of 10M, 20M, up to 60M pages. The curve was fitted using logistic regression, and is restricted to asymptote at the level reported for Google-based KNOWITALL for each relation. We report recall at precision 0.9 for capitals of 195 countries and CEOs of a random selection of the top 5,000 corporations as ranked by PMI. Recall is defined as the percent of countries with a correct capital or the number of correct CEOs divided by the number of corporations.

The curve for `CeoOf` is rising steeply enough that a 400 million page KNOWITNOW index may approach the same level of recall yielded by KNOWITALL when it uses 300,000 Google queries. As shown in Table 2, KNOWITALL takes slightly more than three days to generate these results. KNOWITNOW would operate over a corpus 6.7 times its current one, but the number of required random disk seeks (and the asymptotic run time analysis) would remain the same. We thus expect that with a larger corpus we can construct a KNOWITNOW system that reproduces KNOWITALL levels of precision and recall while still executing in the order of a few minutes.

5 Related Work

There has been very little work published on how to make NLP computations such as PMI-IR and IE fast for large corpora. Indeed, extraction rate is not a metric typically used to evaluate IE systems, but we believe it is an important metric if IE is to scale.

Hobbs *et al.* point out the advantage of fast text processing for rapid system development (Hobbs et al., 1992). They could test each change to system parameters

and domain-specific patterns on a large sample of documents, having moved from a system that took 36 hours to process 100 documents to FASTUS, which took only 11 minutes. This allowed them to develop one of the highest performing MUC-4 systems in only one month.

While there has been extensive work in the IR and Web communities on improvements to the standard inverted index scheme, there has been little work on efficient large-scale search to support natural language applications. One exception is Resnik's Linguist's Search Engine (Elkiss and Resnik, 2004), a tool for searching large corpora of parse trees. There is little published information about its indexing system, but the user manual suggests its corpus is a combination of indexed sentences and user-specific document collections driven by the user's AltaVista queries. In contrast, the BE system has a single index, constructed just once, that serves all queries. There is no published performance data available for Resnik's system.

6 Conclusions and Future Directions

In previous work, statistical NLP computation over large corpora has been a slow, offline process, as in KNOWITALL (Etzioni et al., 2005) and also in PMI-IR applications such as sentiment classification (Turney, 2002). Technology trends, and open source search engines such as Nutch, have made it feasible to create "private" search engines that index large collections of documents; but as shown in Figure 2, firing large numbers of queries at private search engines is still slow.

This paper described a novel and practical approach towards substantially speeding up IE. We described KNOWITNOW, which extracts thousands of facts in minutes instead of days. Furthermore, we sketched URNS, a probabilistic model that both obviates the need for search-engine queries and outputs more accurate probabilities than PMI-IR. Finally, we introduced a simple, efficient approximation to URNS, whose probability estimates are not as good, but which has comparable precision/recall to URNS, making it an appropriate assessor for KNOWITNOW.

The speed and massively improved extraction rate of KNOWITNOW come at the cost of reduced recall. We quantified this tradeoff in Table 2, and also argued that as KNOWITNOW's index size increases from 60 million to 400 million pages, KNOWITNOW would achieve in minutes the same precision/recall that takes KNOWITALL days to obtain. Of course, a hybrid approach is possible where KNOWITNOW has, say, a 100 million page index and, when necessary, augments its results with a limited number of queries to Google. Investigating the extraction-rate/recall tradeoff in such a hybrid system is a natural next step.

While our experiments have used the Web corpus, our approach transfers readily to other large corpora; experimentation with other corpora is another topic for future work. In conclusion, we believe that our techniques transform IE from a slow, offline process to an online one. They could open the door to a new class of interactive IE applications, of which KNOWITNOW is merely the first.

7 Acknowledgments

This research was supported in part by NSF grant IIS-0312988, DARPA contract NBCHD030010, ONR grant N00014-02-1-0324, and gifts from Google and the Turing Center.

References

R. Baeza-Yates and B. Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison Wesley.

E. Brill, J. Lin, M. Banko, S. T. Dumais, and A. Y. Ng. 2001. Data-intensive question answering. In *Procs. of Text REtrieval Conference (TREC-10)*, pages 393–400.

M. Cafarella and O. Etzioni. 2005. A Search Engine for Natural Language Applications. In *Procs. of the 14th International World Wide Web Conference (WWW 2005)*.

D. Downey, O. Etzioni, and S. Soderland. 2005. A Probabilistic Model of Redundancy in Information Extraction. In *Procs. of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*.

E. Elkiss and P. Resnik, 2004. *The Linguist's Search Engine User's Guide*. University of Maryland.

O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134.

M. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Procs. of the 14th International Conference on Computational Linguistics*, pages 539–545, Nantes, France.

J.R. Hobbs, D. Appelt, M. Tyson, J. Bear, and D. Israel. 1992. Description of the FASTUS system used for MUC-4. In *Procs. of the Fourth Message Understanding Conference*, pages 268–275.

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Procs. of the Empirical Methods in Natural Language Processing Conference*, Univ. of Pennsylvania.

P. D. Turney. 2001. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Procs. of the Twelfth European Conference on Machine Learning (ECML-2001)*, pages 491–502, Freiburg, Germany.

P. D. Turney. 2002. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Procs. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pages 417–424.

P. D. Turney, 2004. *Waterloo MultiText System*. Institute for Information Technology, Nat'l Research Council of Canada.