The English-Corpora.org corpus architecture

The fastest, most versatile corpus architecture for large corpora

Mark Davies

The architecture for the corpora from English-Corpora.org is extremely robust. It can handle corpora with (tens of) billions of words of data, and is much faster than corpora from other sites like Sketch Engine and corpora that use CQPWeb, as well as providing a very wide range of query types. The following overview discusses four aspects of the corpus architecture that make it so powerful.

1. Numbers, not words

(Note that this is something that is shared by most other corpus architectures, or at least it should be).

Although texts composed of words and sentences are imported into the corpora, at a certain stage in the process these words are converted to numbers. For example, all instances of *beautiful* might be replaced by the number 1069. The reason for doing this is that it is much faster to search for numbers than to search for strings of characters. To match the word *beautiful*, we would have to find all cases of a [b], and then when we find that, we would check to see if the next letter is an [e] and then [a], [u], and so on (that's nine different comparisons for a word like *beautiful*). But a number like 1069 either matches or doesn't match – all in "one shot". (A number like 1069 also only takes up one byte in terms of storage, whereas with a word it is one byte for each letter in the word.)

So in this case we have an entry in a [lexicon] table (as in Table 1) that has [wordID] of 1069 for *beautiful*.

Table 1. Lexicon (COCA)

wordID	freq	word	lemma	PoS
1069	110525	beautiful	beautiful	jj
2406	38508	beauty	beauty	nn1
10240	7991	beautifully	beautifully	rr
32250	1332	beauties	beauty	nn2
81934	283	beautification	beautification	nn1
113149	192	beautify	beautify	vvi
94405	189	beautician	beautician	nn1

When we do a search like *beautiful NOUN*, it looks up *beautiful* in the lexicon, finds the matching value 1069, and then searches for 1069 in the corpus.

If we have a small corpus like the British National Corpus (BNC) (just 100-200 million words, depending on the version), this doesn't really matter – the corpus is so small that any architecture should work fine. But for *modern* corpora with billions of words of data, we need to use a more powerful approach like this.

2. (Clustered) indexes

This is perhaps the most important aspect of our approach. Suppose we want to search for the most frequent strings of *beautiful NOUN* (*beautiful idea, beautiful scenery*, etc). In COCA (one billion words), *beautiful* occurs 120,698 times (upper and lower case), and in iWeb (14 billion words) it occurs 2,061,276 times.

It would be far too expensive to start at word #1 in the corpus and see if it is the word *beautiful* (or actually the number 1069, as is explained above) and then see if the following word is a noun, and then follow this process for the next billion or so words.

At the most basic level, we would want to have an index of all of the places that *beautiful* (=1069) occurs in the corpus, and then just go to those locations and check to see if the following word is a noun.

But it's actually more difficult than this. In a corpus like iWeb, we would have to read from the drive hundreds

of thousands or millions of times, to check all of the 2,061,276 locations where *beautiful* occurs. This will be slow, even with fast SSD's (solid state drives).

In our approach, we use *clustered* indexes. In our architecture, there is one row in the relational database for each word, as is shown in Table 2 (there are actually 16,402,027,655 rows for the 14 billion word iWeb corpus, since it includes rows for punctuation and text delimiters as well).

And this is where *clustered* indexes become very important. All of the 2,061,276 rows of *beautiful* (in iWeb) are physically stored on the drive in one single "cluster" – *physically adjacent to each other on the drive* (as in Table 2). This means, of course, that the corpus is no longer stored as "sequential words" – *the beautiful scenery inspired her to write poetry* – although that "text" can be reconstructed from the [ID] field in each row (which refers to "word offset" – 1 to *n* in a corpus with *n* words).

So now when we look for *beautiful NOUN* in iWeb, we read from the drive *once* (and grab all 2,061,276 rows for *beautiful* at one time) – rather than reading from the drive hundreds of thousands or millions of times to find the 2,061,276 tokens where *beautiful* occurs.

As far as we know, ours is the only approach that uses clustered indexes, and it shows. This is why our approach is so much faster than any others.

3. Context in columns

If each word in the corpus is represented by one row in the database, and if there is no longer any sequential ordering of the rows (following the sequential words in the text), then how do we find sequences of words, such as *beautiful NOUN*?

One way would be to use a massive "self join" in the database. In this case, we would find all of the [ID] values where *beautiful* occurs (for example, (sequential) word #536497452, 535970915, etc, as in Table 2). We would then check to see if the following word (# 536497453, 535970916, etc) is a noun.

But this operation is extremely "expensive" for a corpus with billions of rows of data, since we are in essence loading the entire corpus "twice" (hence the term "self join"), and then seeing how the [ID] match up. And for a five word string (e.g. *beautiful NOUN in the NOUN*), we would be loading the database five times. That might work fine for a small 100-200 million word corpus like the BNC, but it would be prohibitive for a 10-15 billion word corpus.

Our approach is to store "context" right in the database row. For example, the following are a few sample rows in the database for the word *beautiful* (again, *beautiful* being replaced by 1069). Each row also shows the [ID] (sequential word in the text) and [textID] (a unique number for each text in the corpus).

Table 2. Basic corpus architecture

ID	textID	w6	w7	w8	w9	w10	w11	w12	w12	w13	w14	w15	w16
536497452	298	1668	4	361	12	6	1069	229	229	2	11	86	192
535970915	2298	33	150	8	25	6	1069	311	311	688	9	135	6505
536178344	2597	4486	4096161	41	59	67	1069	7209	7209	1	41	44	97
535099670	3299	2	58	15	33	6	1069	259	259	2	4558474	40	54
535099678	**3299**	**4558474**	**40**	**54**	**12**	**6**	**1069**	**259**	**259**	**2**	**11**	**44**	**9664**
535395358	3399	4054	75	8	6457	6	1069	1	1	311	1	10646	306
536181787	3897	106	4462	8	25	122	1069	1	1	4	37	84	23687
536351108	5095	40	340	1	15	33	1069	2	2	4244256	40	107	1009
536533504	6092	30	187	17	44	1037	1069	1	1	12312	1977	2	92

In each row, columns before and after the *beautiful* column ([w11] or "word11") provide several words of context to the left and to the right. To save space in this document, we only show columns [w6]-[w10] (to the left) and [w12]-[w16] (to the right). But for most corpora, this would actually go from [w1] (ten words to the left of *beautiful*) to [w21] (ten words to the right).

So now when we do a search like *beautiful NOUN* in COCA, it finds the 120,698 rows where *beautiful* occurs (with one single read of the drive, as explained above), and then (in RAM, which is very fast) we look to see which [w12] match up with something labeled as a NOUN in the lexicon.[1] That is why a search like this in COCA takes just about 1 second (with another second or so of "overhead" to log the query, check information regarding the IP address, check the number of queries done by the user, etc). And in iWeb (which is 14 times as big as COCA), it only takes about 3 seconds to search through the 14+ billion rows of data.

To search for collocates within a given span, we would simply do a UNION ALL statement in the SQL command, and count the frequency of the words in the relevant columns (e.g. [w8]-[w10] and [w12]-[w14] (for collocates in the window 3 words left to 3 words right).[2]

The downside of this approach is that we end up using about 21 times as much disk space, because there is so much "redundant" context for each word. For example, there are 21 different rows for the word *beautiful* (wordID = 1069) in this one case in text [3299], as the word is in the central [w11] position, as well as the other 20 positions (10 words left to 10 words right; only 5 left, 5 right shown here).

But drive space is relatively inexpensive nowadays, and so the extra storage space should not be much of an issue. And the upside is that searches are tens or hundreds of times as fast as with other approaches. This really does matter with a site like English-Corpora.org, which is the most heavily-used corpus website in the world.

Table 3. [corpus] sorted by ID (sequential words)

ID	textID	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16
53509966	3299	2	54	33	1697	2	58	15	33	6	**1069**	259
53509967	3299	54	33	1697	2	58	15	33	6	**1069**	259	2
53509968	3299	33	1697	2	58	15	33	6	**1069**	259	2	4558474
535099669	3299	1697	2	58	15	33	6	**1069**	259	2	4558474	40
535099670	3299	2	58	15	33	6	**1069**	259	2	4558474	40	54
535099671	3299	58	15	33	6	**1069**	259	2	4558474	40	54	12
535099672	3299	15	33	6	**1069**	259	2	4558474	40	54	12	6
535099673	3299	33	6	**1069**	259	2	4558474	40	54	12	6	1069
535099674	3299	6	**1069**	259	2	4558474	40	54	12	6	1069	259

4. Scalability, flexibility, and extensibility

As mentioned, English-Corpora.org relies on relational databases. One of the main advantages of this approach (other than speed), is how "extensible" the searches can be. As is shown in Table 1, the basic [lexicon] table contains information for word form, lemma, part of speech, and more. By matching up the [wordID] column with the columns [w11], [w10], etc in

[1] The SQL command (which is generated by the web interface) would be:

```
select top 100 count(*) as freq,d1.word, d2.word from lexicon as d1, lexicon as d2, corpus as x where d1.word
= 'beautiful' and d1.wordID = x.w11 and d2.wordID = x.w12 and d2.pos like 'nn%'
group by d1.word,d2.word order by count(*) desc
```

[2] The SQL command (which is generated by the web interface) to find the top 100 NOUN collocates of *plants* between 3 words left and 3 words right would be:

```
select top 100 count(*),b.w1 from (
SELECT x.w8 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID
UNION ALL SELECT x.w9 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID
UNION ALL SELECT x.w10 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID
UNION ALL SELECT x.w12 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID
UNION ALL SELECT x.w13 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID
UNION ALL SELECT x.w14 FROM corpus as x, lexicon as x1 where x1.word like 'plants' and x.w11 = x1..wordID )
a, lexicon as b where b.pos like 'nn%' and a.w8 = b.wordID group by b.w1 order by count(*) desc
```

the [corpus] table (see Table 2 above), we can search the corpus by word, lemma, or part of speech.

But because this is a relational database, we can add additional information, and there is little if any decrease in performance. For example, we could have a [synonyms] or [userDefinedWords] table that is linked to the [lexicon] table, to allow searches like:

=CLEAN v * NOUN: *cleaned the house, wiping her hands, mops the floor, scrubbing the floor*

@COLORS @CLOTHES: *black tie, white blouse, red shoes, yellow hat*

Likewise, we can have a [sources] table with essentially any metadata, and this is then linked to the [textID] in the [corpus] table (Table 2 above). The following are a few columns from the COCA [sources] table.[3]

Table 4. [Sources] table (COCA)

textID	year	genre	Source (abbreviated)	title
221235	1993	SPOK	ABC_20/20	Believe It Or Else; The Heart Attack Test
1001235	1995	FIC	SatEvenPost	Old-fashioned Thanksgiving
2001234	1992	MAG	FieldStream	Spirits in the Sky
3001235	1992	NEWS	NYTimes	Judge and Heiress: The Rise and Fall of a Private Affair
4016912	1997	ACAD	BioCycle	Investing in organics diversion at state prisons.
5041236	2012	WEB	wikihow.com	How to Know if a Girl Likes You: 12 steps (with pictures)
5131236	2012	BLOG	blogs.adobe.com	John Nack on Adobe : Animation: A building's windows as pixels
5211234	2004	MOV	Highwaymen	Highwaymen
5231234	2012	TV	How I Met Your Mother	Trilogy Time

In addition, users can quickly (2-3 seconds) and easily (just a few clicks) create a "Virtual Corpus" composed of texts selected by source, author, date, genre, sub-genres, or many other types of metadata (even information from IMDB for the TV and Movies corpora). Users can also create a Virtual Corpus in 2-3 seconds, based on words or phrases in the texts (such as *DNA, basketball, nuclear energy*, or *Harry Potter*).

The corpus stores all of the [textID] for their Virtual Corpus, and then the SQL JOIN statement between their personalized list and the main [sources] table is what allows them to limit their search to a particular set of texts, or to extract keywords from their Virtual Corpora (in just 1-2 seconds).

Conclusion

The architecture for the corpora at English-Corpora.org is based on relational databases, and this approach uses 1) integer values for words 2) clustered indexes and 3) in-row context.

With this architecture, users can perform powerful searches involving word, lemma, part of speech, synonyms, user-defined wordlists, as well as a wide range of metadata for the texts (including the ability to quickly and easily create Virtual Corpora).

These searches can be done on even very large corpora (with billions of words of data) much more quickly than with any other corpus architecture, including Sketch Engine. And all of this allows researchers, teachers, and students to gain insight into language in ways that are not possible with any other corpora.

[3] The SQL statement (which is generated by the web interface) to find the top 100 adjectives before the word *refugees* in the 15 billion word NOW corpus, in texts from GB (the UK) where *Guardian* is in the source and the texts are from June-Dec 2015, would be the following (and it takes about 2 seconds to run):

```
select top 100 count(*),d1.word,d2.word from sources as s, lexicon as d1, lexicon as d2, corpus as x where
d2.word = 'refugees' and d2.wordID = x.w11 and d1.wordID = x.w10 and d1.pos like 'j%' and x.textID = s.textID
and s.country = 'GB' and s.source like '%Guardian%' and s.date between '15-06-01' and '15-12-31' group by
d1.word,d2.word order by count(*) desc
```