FIND: Fast **IN**dexing into **D**atasets

Abdi-Hakin Dirie & Jason Tong
6.905 | Spring 2015

Project overview and objective

The motivation behind FIND stems from a desire to parse and search the Child Language Data Exchange System (CHILDES), a corpus commonly used to research child language acquisition. The dataset of conversations consists of structured data that lends itself to systematically parsing tagged fields annotating the context, syntactic structure, accompanying action, and other information attached to a particular utterance in a transcript.

An extensive command line system exists for browsing the dataset online, but we wanted to create a generalized system for searching text documents that could be easily extended to the same degree of specialization as that of the existing tool, but not only for CHILDES but other similar corpora as well. There are many other bodies of well-structured textual data that do not necessarily follow precisely the same format from article to article (Wikipedia immediately comes to mind as a well-known example) and in today's data-centric world, we believe that the number of corpora that fit this bill will only grow.

```
Let me first understand you, I replied.
justice, as you say, is the
interest of the stronger. What, Thrasymachus,
is the meaning of this?
You cannot mean to say that because Polydamas,
the pancratiast, is
stronger than we are, and finds the eating of
beef conducive to his
bodily strength, that to eat beef is therefore
equally for our good
who are weaker than he is, and right and just
for us?

That's abominable of you, Socrates; you take
the words in the sense
which is most damaging to the argument.

Not at all, my good sir, I said; I am trying to
understand them; and
I wish that you would be a little clearer.
```

```
@Loc: Eng-NA-MOR/Bates/Free20/amy20.cha
@PID: 11312/c-00015218-1
@Begin
@Languages: eng
@Participants:    CHI Child , MOT Mother
@ID:  eng|Bates|CHI|1;8.
|female|typical|MC|Child|||
@ID:  eng|Bates|MOT|||||Mother|||
@Bg:  freeplay
*MOT: what's that ?
%mor: pro:wh|what~cop|be&3S pro:dem|that ?
%gra: 1|2|SUBJ 2|0|ROOT 3|2|PRED 4|2|PUNCT
%act: holds object out to Amy
*CHI: yyy .
%gpx: looks at chicken
%act: holds nesting cups
%xpho:      wi
```

Structured (left) and unstructured (right) data

One additional feature we found to be lacking in the existing tool for searching in CHILDES is a ranking metric. With such a large corpus of data, the inability to rank or in some way order positive results is a significant drawback that FIND seeks to address.

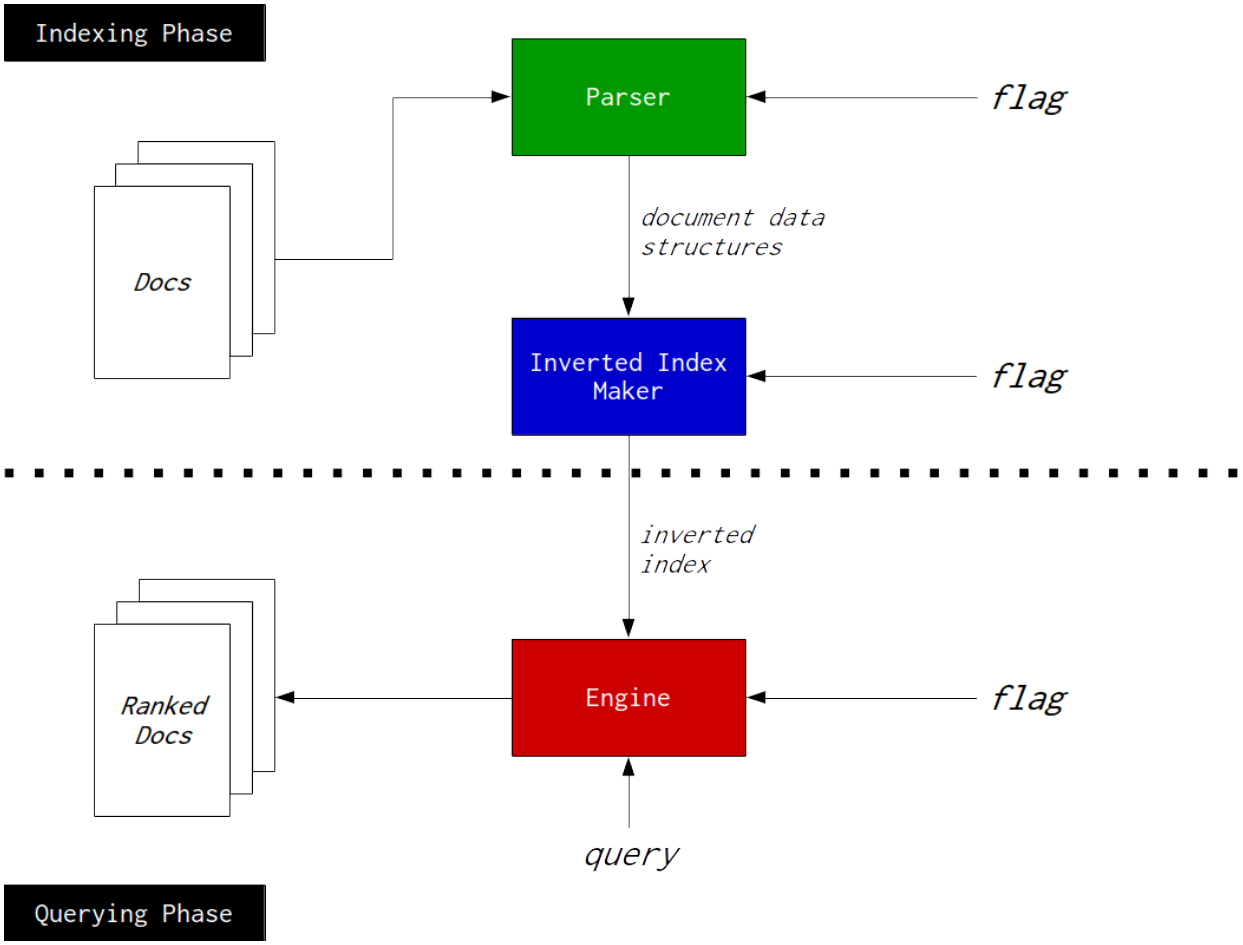
System Overview

The system has two main phases: the indexing phase and querying phase. The indexing phase starts up the system by creating all the necessary data structures. During the querying phase, the user can interact with the system to get results.

First, the indexing phase. The system ingests a corpora of textual documents and feeds it into the parser. The parser forwards its results to the inverted index maker, which then outputs the primary data structure used by the engine.

In the querying phase, the user provides a well-formed query to the system. The engine uses the provided inverted index to best find the documents that the user query asks for. The engine returns a subset of the documents, ranked from most relevant to least relevant.

The flag parameter specifies which corpus we are dealing with, and, consequently, how the three main modules - parser, inverted index maker, and engine - execute their computations.



System diagram

Modules

The system has three main modules: the parser, the inverted index maker, and the engine. To illustrate the behaviour of each component, we will use plain text and CHILDES (linguistic) documents as the corpora.

Parser

The job of the parser is to transform a textual document into a temporary in-memory data structure that is used to retrieve pieces of the document efficiently. The parser does this computation with not one document, but a batch of documents. The parser receives a list of document names and their paths. For each document, the parser return a list data structure that represents the document and its contents. For example, the parser would output the following for the document "republic.txt":

```
(plain
 "republic.txt"
 (... "the" "Piraeus" "with" "Glaucon" "the" "son" "of" "Ariston"
...))
```

The returned object is a tagged data structure where the first element is the type of document (as a symbol), the second element is the pathname of the document within the corpus directory (as a string), and the third element is a list of the words (each a string) in the order they appear in the document.

The parser can be augmented to parse a different document if it knows its structure as indicated by the flag. As another example, a sample parse for a linguistic document might look as follows:

```
(ling
 "english-na-mor/Bates/Free20/amy20.cha"
 (...
  ("@PID:" "11312/c-00015218-1")
  ("@Begin")
  ("@Languages:" "eng")
  ("@Participants:" "CHI" "Child" "," "MOT" "Mother")
  ...))
 (...
  (("*CHI:" "yyy" ".")
   ("%gpx:" "looks" "at" "chicken")
   ("%act:" "holds" "nesting" "cups")
```

```
 ("%xpho:" "wi"))
(("*MOT:" "it's" "a" "woof+(woof)" ".")
 ("%mor:" "pro|it~cop|be&3S" "det|a" "on|+on|woof+on|woof" ".")
 ("%gra:" "1|2|SUBJ" "2|0|ROOT" "3|4|DET" "4|2|PRED" "5|2|PUNCT"))
...))
```

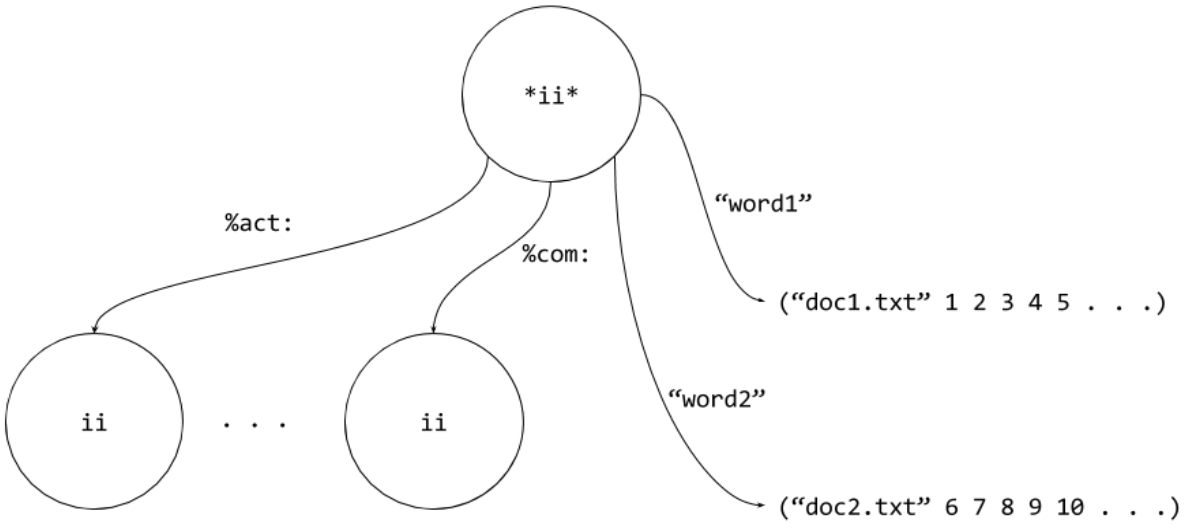
Here, the document is tagged with the symbol *ling*, indicating that it is a linguistic document. The second element is the pathname to the document, the third element holds a variety of metadata, and the fourth and final element holds the exchange by the participants in the interaction. With the appropriate accessor functions, the system can pick out specific pieces of information to index.

Inverted Index Maker

The inverted index maker is tasked with creating an inverted index-like data structure. The inverted index is used by the engine to provide fast lookup for certain pieces of information within a specified corpora. In its simplest form, the inverted index is a hash map that maps individual words to a list where each element is a pair of the document's name and a list of the word's index in the document. For example, querying for the word "fantastic" will return:

```
(("republic.txt" 4954 104681)
 ("sherlock.txt" 17138 23346 38866 40055 42641)
 ("time.txt" 31504))
```

Extensible versions of the inverted index may use more than just words as keys and the values may be more than just association lists. As documents become more structured, nested inverted indices can be constructed within the inverted index maker so that structured information can still be looked up quickly. With regards to CHILDES, the top level inverted index pointed to other inverted indices that are used to perform queries on only a subset of the CHILDES corpus.



Nested inverted index

This enables user to ask queries semantically along the lines of "Find me all the documents where the phrase 'picks up' is present in the comments field." The top-level inverted index still points to each individual whitespace-free string, allowing for plain text queries to still function uninhibited. The system uses random generated hashes as keys to the nested inverted indices to prevent shadowing of words.

Engine

The engine takes on the task of compiling the desired results of a user query using the inverted index. It is the component that looks up the relevant documents in the corpus, and then applies the appropriate operations to retrieve the documents of interest in a ranked order.

Basic searches include the single keyword search, multiple keyword search, and phrasal search. In addition, as part of the phrasal search, there is a functionality under the hood that allows for searching two words and specifying the offset between them. At the moment, it does not seem likely for a user to want such a specific functionality, so it has not been included into the search generic operator.

```
(define search (make-generic-operator 1 'search))
(defhandler search s:keyword word?)
(defhandler search s:keywords words?)
(defhandler search s:phrase phrase?)
```

Below are several examples of `search`:

single-keyword searches: returns documents exact matches to the input keyword

```
(s:rank (search "courage"))
;(("republic.txt" . 14) ("paradise.txt" . 4) ("time.txt" . 3))

(s:rank (search "righteousness"))
;(("paradise.txt" . 2) ("republic.txt" . 1))

(s:rank (search "virtue"))
;(("republic.txt" . 48) ("time.txt" . 1))
```

multiple-keyword search: returns documents exactly matching any of the input keywords

```
(s:rank (search (list "courage" "righteousness" "virtue")))
;(("republic.txt" . 63) ("paradise.txt" . 6) ("time.txt" . 4))
```

phrasal search: returns documents exactly matching the input phrase

```
(s:rank (search "a box"))
;(("alice.txt" . 2) ("sherlock.txt" . 2) ("time.txt" . 1))
```

No basic search engine would be complete without the capacity to combine searches. For example, we might be interested in matching multiple keywords and be only interested in the presence of at least one of the words. We might make a stricter demand that the keywords all appear in each document. We might string together a series of predicates with boolean operators to find documents matching a very specific set of criteria. These queries of unbounded complexity are achieved with set operations on the association lists returned by the inverted index for individual keyword matches in the corpus. After ranking, the results are returned in such a way that the indices to matches are hidden, but before s:rank, we actually leave the indices to allow for further manipulation of the individual results, which is what allows us to score and combine search results.

As an example, there are two combinations of the above single-keyword searches:

```
(s:rank
    (s:or (search "courage") (search "righteousness") (search
"virtue")))
;(("republic.txt" . 63) ("paradise.txt" . 6) ("time.txt" . 4))

(s:rank
    (s:and (search "courage") (search "righteousness") (search
"virtue")))
;(("republic.txt" . 63))
```

The result of `s:or` matches the multiple-keyword search result, as expected. This becomes more powerful when we introduce the idea of tiers and the scope of the disjuncts are different, which leads us to our work in the linguistic search engine through CHILDES.

The linguistic search engine works very much the same way plaintext search does, with the exception of requiring an additional *tier* argument, that specifies the scope of the search. For example, the *tier* containing notes about action is "%act:" and "*" holds all the utterances made by participants. Note that plaintext search queries still work on CHILDES datasets and that the linguistic search engine simply augments the base engine's functionality.

Below are several examples:

```
(s:rank (search:ling "smiles" "%gpx:"))
;(("english-na-mor/Bates/Free20/amy20.cha" . 3)
 ("english-na-mor/Bates/Free20/betty20.cha" . 1)
 ("english-na-mor/Bates/Free20/chuck20.cha" . 1))

(s:rank (search:ling (list "fingers" "wiggles") "%act:"))
;(("english-na-mor/Bates/Free20/amy20.cha" . 4)
 ("english-na-mor/Bates/Free20/betty20.cha" . 2))
```

An example where we combine the linguistic search engine with the basic search engine

```
(s:rank (s:and (search:ling "ball" "%act:") (search "MOT Mother")))
;(("english-na-mor/Bates/Free20/amy20.cha" . 11)
 ("english-na-mor/Bates/Free20/chuck20.cha" . 5))
```

The simplest way to rank documents is by the frequency with which the searched word appears. Alternatively, the term frequency-inverse document frequency (tf-idf), which is a slightly more nuanced metric that considers the "significance" of a word and takes into consideration the prevalence of a particular word in the entire corpus could be plugged in as the metric instead. The search infrastructure is designed in such a way that different metrics could be added and swapped with ease, which is important as it is probably a decision informed by the type of data in the corpus. In the above examples, word frequency was the metric for relevance.

Future work

CHILDES

As mentioned in the introduction, the original inspiration for FIND is derived from the recognition that there is a lot of power in a general search engine that could be extended to take advantage of structure in the data.

A particular use case for CHILDES might be a search for children who overregularize for verb tense morphological inflection. Children initially learn irregular forms well because there is no generalization mechanism and the production is very much tied to input from the environment. However, children learn morphological rules, such as adding "-ed" to produce a past tense form. At some point, children *overregularize* and produce erroneous constructions that they were unlikely to have picked up from the environment, such as "goed" or "runned." A researcher interested in this phenomenon would be greatly facilitated by FIND if it could provide a way to search for utterances produced by children within a particular age range (header data that we can access thanks to special handling in the parser and inverted index maker) and within those, words that have been tagged as verbs tagged with past tense inflection, and even with the additional condition that it end with [d] in the phonological analysis of the utterance.

The base level search functions can be augmented to include searches that are specifically dedicated to particular annotation fields attached to utterances, also known as *tiers*. For example, we might tackle the above use case an intersection of documents with children under the age of three and documents where there is a child-produced utterance tagged as a verb in the past tense in the %mor or morphological tier (e.g. |go&PAST V|). The possibilities are unbounded.

Wikipedia

Wikipedia is a treasure trove of structured data attached to documents. Though not in the same manner as CHILDES transcripts, which contain blocks of data (utterances) each annotated with the same fields, Wikipedia has a lot of structured data attached to the entity that is the subject of a given page, such as birthplace, chemical symbol, geographic coordinates, or alma mater. Recognizing this and extending the parser and inverted index maker to handle additional specificity over where a word appears significantly reduces the search space for matching a particular query.

CHILDES and Wikipedia are only two examples sources of partially structured textual data that can benefit from the extensibility of the base level search functions offered by FIND. The entire purpose of FIND was to provide a general search engine that could then be extended to take advantage of structures in the corpus that provides more information than what is readily available from treating the entire document as a list of strings. As such, there many directions in which FIND could be extended, depending on the nature of the corpus of interest.

Code

The source code for FIND can be found at https://github.com/aadah/childes