COBS Documentation

Release 0.1.2

Timo Bingmann

Nov 24, 2021

Contents

COBS (COmpact Bit-sliced Signature index) is a cross-over between an inverted index and Bloom filters. Our target application is to index k-mers of DNA samples or q-grams from text documents and process **approximate pattern matching** queries on the corpus with a user-chosen coverage threshold. Query results may contain a number of false positives which decreases exponentially with the query length and the false positive rate of the index determined at construction time. COBS' compact but simple data structure outperforms other indexes in construction time and query performance with Mantis by Pandey et al. in second place. However, unlike Mantis and other previous work, COBS does not need the complete index in RAM and is thus designed to scale to larger document sets.

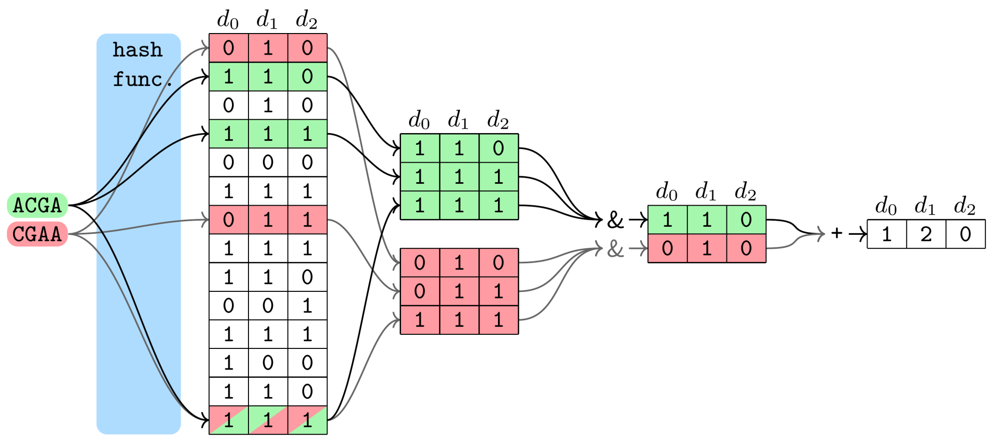

Fig. 3. Architecture of the bit-sliced signature index and query processing steps.

More information about COBS is presented in our current research paper Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. "COBS: a Compact Bit-Sliced Signature Index". In: *26th International Symposium on String Processing and Information Retrieval (SPIRE)*. pages 285-303. Spinger. October 2019. preprint arXiv:1905.09624.

See the tutorial page on how to use COBS in python scripts.

CHAPTER 1

Table of Contents

1.1 Tutorial for COBS Python Interface

1.1.1 Installation

Installation of COBS with Python interface is easy using pip. The package name on PyPI is `cobs_index` and you need cmake and a recent C++11 compiler to build the C++ library source.

```
$ pip install --user cobs_index
```

1.1.2 Document Lists

COBS can read and create an index from the following document types:

- FastA (`.fasta`, `.fa`, `.fasta.gz`, `.fa.gz`)
- FastQ (`.fastq`, `.fq`, `.fastq.gz`, `.fq.gz`)
- McCortex (`.ctx`, `.cortex`)
- text files (`.txt`)
- MultiFastA (`.mfasta`)

The document types are identified by extension and compressed `.gz` files are handled transparently. The set of k-mers extracted from each file type is handled slightly differently: for FastA files each continuous subsequence is broken into k-mers individually, while McCortex files explicitly list all k-mers, and for text files the entire continuous file is broken into k-mers. Each document creates one entry in the index, except for MultiFastA were each subsequence is considered an individual document.

COBS usually scans a directory and creates an index containing all documents it finds. For more fine-grain control, document lists are represented using `DocumentList` objects. DocumentLists can be created empty or by scanning a directory, files can be added, and they contain `DocumentEntry` objects which can be iterated over.

```
import cobs_index as cobs

doclist1 = cobs.DocumentList("/path/to/documents")
print("doclist1: ({} entries)".format(len(doclist1)))
for i, d in enumerate(doclist1):
    print("doc[{}] name {} size {}".format(i, d.name, d.size))

doclist2 = cobs.DocumentList()
doclist2.add("/path/to/single/document.fa")
doclist2.add_recursive("/path/to/documents", cobs.FileType.Fasta)
print("doclist2: ({} entries)".format(len(doclist2)))
for i, d in enumerate(doclist2):
    print("doc[{}] name {} size {}".format(i, d.name, d.size))
```

1.1.3 Index Construction

Compact indices are constructed using the functions `compact_construct()` or `compact_construct_list()`. The first scans a directory for documents and constructs an index from them, while the latter takes a explicit `DocumentList`. Note that the output index file *must* end with `.cobs_compact`.

```
cobs.compact_construct("/path/to/documents", "my_index.cobs_compact")
```

Parameters for index construction may be passed using a `CompactIndexParameters` object. See the class documentation for a complete list of parameters. The default parameters are a reasonable choice for most DNA k-mer applications.

```
import cobs_index as cobs

p = cobs.CompactIndexParameters()
p.term_size = 31              # k-mer size
p.clobber = True              # overwrite output and temporary files
p.false_positive_rate = 0.4   # higher false positive rate -> smaller index

cobs.compact_construct("/path/to/documents", "my_index.cobs_compact", index_params=p)
```

Besides compact indices, COBS also constructs and supports "classic" indices. These are however usually not be used in practice and thus not discussed here further.

1.1.4 Querying an Index

To query an index, first load it using a `Search` object. This method detects the type of index, reads the metadata, and opens the entire file using `mmap`.

Querying is performed with the `Search.search()` method. This method returns **a list containing pairs**: `(#occurrences, document name)`.

```
import cobs_index as cobs

s = cobs.Search("out.cobs_compact")
r = s.search("AGTCAACGCTAAGGCATTTCCCCCCTGCCTCCTGCCTGCTGCCAAGCCCT")
print(r)
# output: [(20, 'sample1'), (16, 'sample2'), ...]
```

With the default search parameters **all document scores** are returned. For large corpora creating this Python list is a substantial overhead, such that the result set should be limited using a) the `threshold` parameter or b) the `num_results` parameter. Threshold determines the fraction of k-mers in the query a document be reach to be included in the result, while `num_results` simply limits the list size to a given number.