PISA Documentation

PISA

Mar 31, 2020

Contents:

Performant Indexes and Search for Academia

CHAPTER 1

Description

PISA is a text search engine able to run on large-scale collections of documents. It allows researchers to experiment with state-of-the-art techniques, allowing an ideal environment for rapid development.

Some features of PISA are listed below:

- Written in C++ for performance;
- Indexing & Parsing & Sharding capabilities;
- Many index compression methods implemented;
- Many query processing algorithms implemented;
- Implementation of document reordering;
- Free and open-source with permissive license;

Note: PISA is still in its **initial release** and many new features are going to come in the next versions. Contributions are also welcome!

1.1 Getting Started

1.1.1 Building the code

The code is tested on Linux with `GCC 7.4.0`, `GCC 8.1.0`, `Clang 5.0`, `Clang 6.0` and on macOS with `AppleClang 9.1.0`.

The following dependencies are needed for the build.

- CMake >= 3.0, for the build system
- OpenMP (optional)

To build the code:

```
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ make
```

1.1.2 Run unit tests

To run the unit tests simply perform a `make test`.

The directory `test/test_data` contains a small document collection used in the unit tests. The binary format of the collection is described in a following section. An example set of queries can also be found in `test/test_data/queries`.

1.2 Parsing

A *forward index* is a data structure that stores the term identifiers associated to every document. Conversely, an *inverted index* stores for each unique term the document identifiers where it appears (usually, associated to a numeric value used for ranking purposes such as the raw frequency of the term within the document).

The objective of the parsing process is to represent a given collection as a forward index. To parse a collection, use the `parse_collection` command:

```
parse_collection - parse collection and store as forward index.
Usage: ./bin/parse_collection [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -o,--output TEXT REQUIRED   Forward index filename
  -j,--threads UINT           Thread count
  -b,--batch-size INT=100000  Number of documents to process in one thread
  -f,--format TEXT=plaintext  Input format
  --stemmer TEXT              Stemmer type
  --content-parser TEXT       Content parser type
  --debug                     Print debug messages

  Subcommands:
    merge                       Merge previously produced batch files.
                                When parsing process was killed during merging, use␣
↪this
                                command to finish merging without having to restart␣
↪building batches.
```

For example:

```
$ mkdir -p path/to/forward
$ zcat ClueWeb09B/*/*.warc.gz |    # pass unzipped stream in WARC format
    parse_collection \
    -j 8                           # use up to 8 threads at a time
    -b 10000                       # one thread builds up to 10k documents in memory
    -f warc                        # use WARC
    --stemmer porter2              # stem every term using the Porter2 algorithm
    --content-parser html          # parse HTML content before extracting tokens
    -o path/to/forward/cw09b
```

In case you get the error `-bash: /bin/zcat: Argument list too long`, you can pass the unzipped stream using:

```
$ find ClueWeb09B -name '*.warc.gz' -exec zcat -q {} \;
```

The parsing process will write the following files:

- `cw09b`: forward index in binary format.
- `cw09b.terms`: a new-line-delimited list of sorted terms, where term having ID N is on line N, with N starting from 0.
- `cw09b.termlex`: a binary representation of the `.terms` file that is used to look up term identifiers at query time.
- `cw09b.documents`: a new-line-delimited list of document titles (e.g., TREC-IDs), where document having ID N is on line N, with N starting from 0.
- `cw09b.doclex`: a binary representation of the `.documents` file that is used to look up document identifiers at query time.
- `cw09b.urls`: a new-line-delimited list of URLs, where URL having ID N is on line N, with N starting from 0. Also, keep in mind that each ID corresponds with an ID of the `cw09b.documents` file.

1.2.1 Generating mapping files

Once the forward index has been generated, a binary document map and lexicon file will be automatically built. However, they can also be built using the `lexicon` utility by providing the new-line delimited file as input. The `lexicon` utility also allows efficient look-ups and dumping of these binary mapping files.

Examples of the `lexicon` command are shown below:

```
Build, print, or query lexicon
Usage: ./bin/lexicon [OPTIONS] SUBCOMMAND

Options:
    -h,--help                   Print this help message and exit

Subcommands:
    build                       Build a lexicon
    lookup                      Retrieve the payload at index
    rlookup                     Retrieve the index of payload
    print                       Print elements line by line
```

For example, assume we have the following plaintext, new-line delimited file, `example.terms`:

```
aaa
bbb
def
zzz
```

We can generate a lexicon as follows: `./bin/lexicon build example.terms example.lex`

You can dump the binary lexicon back to a plaintext representation: `./bin/lexicon print example.lex` which should output:

```
aaa
bbb
```

(continues on next page)

(continued from previous page)

```
def
zzz
```

You can retrieve the term with a given identifier: `./bin/lexicon lookup example.lex 2` which outputs `def`

Finally, you can retrieve the id of a given term: `./bin/lexicon rlookup example.lex def` which outputs `2`. NOTE: This requires the initial file to be lexicographically sorted, as `rlookup` depends on binary search.

1.2.2 Supported stemmers

- Porter2
- Krovetz

1.2.3 Supported formats

- `plaintext`: every line contains the document's title first, then any number of .. code-block:: guess

  whitespaces, followed by the content delimited by a new line character.
- `trectext`: TREC newswire collections.
- `trecweb`: TREC web collections.
- *warc*: Web ARChive format as defined in [the format specification](https://iipc.github.io/warc-specifications/specifications/warc-format/warc-1.0/).
- `wapo`: TREC Washington Post Corpus.

In case you want to parse a set of files where each one is a document (for example, the collection wiki-large), use the `files2trec.py` script to format it to TREC (take into account that each relative file path is used as the document ID). Once the file is generated, parse it with the `parse_collection` command specifying the `trectext` value for the `--format` option.

1.3 Inverting

Once the parsing phase is complete, use the `invert` command to turn a *forward index* into an *inverted index*:

```
invert - turn forward index into inverted index
Usage: ./invert [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -i,--input TEXT REQUIRED    Forward index filename
  -o,--output TEXT REQUIRED   Output inverted index basename
  -j,--threads UINT           Thread count
  --term-count UINT REQUIRED  Term count
  -b,--batch-size INT=100000  Number of documents to process at a time
```

For example, assuming the existence of a forward index in the path `path/to/forward/cw09b`:

```
$ mkdir -p path/to/inverted
$ ./invert -i path/to/forward/cw09b -o path/to/inverted/cw09b --term-count `wc -w <␣
↪path/to/forward/cw09b.terms`
```

Note that the script requires as parameter the number of terms to be indexed, which is obtained by embedding the `wc -w < path/to/forward/cw09b.terms` instruction.

1.3.1 Inverted index format

A *binary sequence* is a sequence of integers prefixed by its length, where both the sequence integers and the length are written as 32-bit little-endian unsigned integers. An *inverted index* consists of 3 files, `<basename>.docs`, `<basename>.freqs`, `<basename>.sizes`:

- `<basename>.docs` starts with a singleton binary sequence where its only integer is the number of documents in the collection. It is then followed by one binary sequence for each posting list, in order of term-ids. Each posting list contains the sequence of document-ids containing the term.
- `<basename>.freqs` is composed of a one binary sequence per posting list, where each sequence contains the occurrence counts of the postings, aligned with the previous file (note however that this file does not have an additional singleton list at its beginning).
- `<basename>.sizes` is composed of a single binary sequence whose length is the same as the number of documents in the collection, and the i-th element of the sequence is the size (number of terms) of the i-th document.

1.3.1.1 Reading the inverted index using Python

```
import os
import numpy as np

class InvertedIndex:
    def __init__(self, index_name):
        index_dir = os.path.join(index_name)
        self.docs = np.memmap(index_name + ".docs", dtype=np.uint32,
              mode='r')
        self.freqs = np.memmap(index_name + ".freqs", dtype=np.uint32,
              mode='r')

    def __iter__(self):
        i = 2
        while i < len(self.docs):
            size = self.docs[i]
            yield (self.docs[i+1:size+i+1], self.freqs[i-1:size+i-1])
            i += size+1

    def __next__(self):
        return self

for i, (docs, freqs) in enumerate(InvertedIndex("cw09b")):
    print(i, docs, freqs)
```

1.4 Sharding

We support partitioning a collection into a number of smaller subsets called *shards*. Right now, only a forward index can be partitioned by running `partition_fwd_index` command. Then, the resulting shards must be inverted individually with `invert` command. For convenience, we provide `script/invert-shards` that takes a file prefix to shard forward indexes and inverts them all.

1.4.1 partition_fwd_index

```
Partition a forward index
Usage: ./bin/partition_fwd_index [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -i,--input TEXT REQUIRED    Forward index filename
  -o,--output TEXT REQUIRED   Basename of partitioned shards
  -j,--threads INT            Thread count
  -r,--random-shards INT Excludes: --shard-files
                              Number of random shards
  -s,--shard-files TEXT ... Excludes: --random-shards
                              List of files with shard titles
  --debug                     Print debug messages
```

For example, one can partition collection randomly:

```
$ partition_fwd_index \
    -j 8                          # use up to 8 threads at a time
    -i full_index_prefix \
    -o shard_prefix \
    -r 123                          # partition randomly into 123 shards
```

Alternatively, a set of files can be provided. Let's assume we have a folder `shard-titles` with a set of text files. Each file contains new-line-delimited document titles (e.g., TREC-IDs) for one partition. Then, one would call:

```
$ partition_fwd_index \
    -j 8                          # use up to 8 threads at a time
    -i full_index_prefix \
    -o shard_prefix \
    -s shard-titles/*
```

Note that the names of the files passed with `-s` will be ignored. Instead, each shard will be assigned a numerical ID from `0` to `N - 1` in order in which they are passed in the command line. Then, each resulting forward index will have appended `.ID` to its name prefix: `shard_prefix.000`, `shard_prefix.001`, and so on.

1.4.2 invert-shards.sh

This script inverts all shards with a common prefix.

```
USAGE:
    invert-shards <PROGRAM> <INPUT_BASENAME> <OUTPUT_BASENAME> [program flags]
```

For example, if the following command was used to partition a collection:

```
$ partition_fwd_index \
    -j 8                          # use up to 8 threads at a time
    -i full_index_prefix \
    -o shard_prefix \
    -r 123                          # partition randomly into 123 shards
```

Then, one can invert the shards by executing the following script:

```
$ invert-shards.sh \
    /path/to/invert               # provide path to program
```

(continues on next page)

(continued from previous page)

```
    shard_prefix                  # basename to shard collections
    shard_prefix_inverted         # basename to shard inverted indexes
    -j 8 -b 1000                    # any arguments to be appended to each program␣
↪execution
```

1.4.3 compress-shards.sh

Next, you can compress the inverted shards with `compress-shards.sh`:

```
USAGE:
    compress-shards <PROGRAM> <INPUT_BASENAME> <OUTPUT_BASENAME> [program flags]
```

For example, following the above example:

```
$ compress-shards.sh \
    /path/to/create_freq_index    # provide path to program
    shard_prefix_inverted         # basename to shard inverted indexes
    shard_prefix_inverted_simdbp  # basename to shard compressed indexes
    -t block_simdbp --check         # any arguments to be appended to each program␣
↪execution
```

Note that this script can be also used for creating WAND data by replacing the program:

```
$ compress-shards.sh \
    /path/to/create_wand_data     # provide path to program
    shard_prefix_inverted         # basename to shard inverted indexes
    shard_prefix_inverted_wand      # basename to shard compressed indexes
```

1.5 Compress Index

1.5.1 Usage

To create an index use the command `create_freq_index`. The available index types are listed in `index_types.hpp`.

```
create_freq_index - a tool for creating an index.
Usage:
  create_freq_index [OPTION...]

  -h, --help                 Print help
  -t, --type type_name       Index type
  -c, --collection basename  Collection basename
  -o, --out filename         Output filename
      --check                Check the correctness of the index (default:
                             false)
```

For example, to create an index using the optimal partitioning algorithm using the test collection, execute the command:

```
$ ./bin/create_freq_index -t opt -c ../test/test_data/test_collection -o test_
↪collection.index.opt --check
```

where `test/test_data/test_collection` is the *basename* of the collection, that is the name without the `.{docs,freqs,sizes}` extensions, and `test_collection.index.opt` is the filename of the output index. `--check` perform a verification step to check the correctness of the index.

1.5.2 Compression Algorithms

1.5.2.1 Binary Interpolative Coding

Binary Interpolative Coding (BIC) directly encodes a monotonically increasing sequence. At each step of this recursive algorithm, the middle element m is encoded by a number $m\ \ l\ \ p$, where l is the lowest value and p is the position of m in the currently encoded sequence. Then we recursively encode the values to the left and right of m. BIC encodings are very space-efficient, particularly on clustered data; however, decoding is relatively slow.

To compress an index using BIC use the index type `block_interpolative`.

> Alistair Moffat, Lang Stuiver: Binary Interpolative Coding for Effective Index Compression. Inf. Retr. 3(1): 25-47 (2000)

1.5.2.2 Elias-Fano

Given a monotonically increasing integer sequence S of size n, such that (S_{n-1} < u), we can encode it in binary using (lceillog urceil) bits. Elias-Fano coding splits each number into two parts, a low part consisting of (l = lceillog frac{u}{n}rceil) right-most bits, and a high part consisting of the remaining (lceillog urceil - l) left-most bits. The low parts are explicitly written in binary for all numbers, in a single stream of bits. The high parts are compressed by writing, in negative-unary form, the gaps between the high parts of consecutive numbers.

To compress an index using Elias-Fano use the index type `ef`.

> Sebastiano Vigna. 2013. Quasi-succinct indices. In Proceedings of the sixth ACM international conference on Web search and data mining (WSDM '13). ACM, New York, NY, USA, 83-92.

1.5.2.3 MaskedVByte

> Jeff Plaisance, Nathan Kurz, Daniel Lemire, Vectorized VByte Decoding, International Symposium on Web Algorithms 2015, 2015.

1.5.2.4 OptPFD

> Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted index compression and query processing with optimized document ordering. In Proceedings of the 18th international conference on World wide web (WWW '09). ACM, New York, NY, USA, 401-410. DOI: https://doi.org/10.1145/1526709.1526764

1.5.2.5 Partitioned Elias Fano

> Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned Elias-Fano indexes. In Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval (SIGIR '14). ACM, New York, NY, USA, 273-282. DOI: https://doi.org/10.1145/2600428.2609615

1.5.2.6 QMX

Quantities, Multipliers, and eXtractor (QMX) packs as many integers as possible into 128-bit words (Quantities) and stores the selectors (eXtractors) separately in a different stream. The selectors are compressed (Multipliers) with RLE (Run-Length Encoding).

To compress an index using QMX use the index type `block_qmx`.

> Andrew Trotman. 2014. Compression, SIMD, and Postings Lists. In Proceedings of the 2014 Australasian Document Computing Symposium (ADCS '14), J. Shane Culpepper, Laurence Park, and Guido Zuccon (Eds.). ACM, New York, NY, USA, Pages 50, 8 pages. DOI: https://doi.org/10.1145/2682862.2682870

1.5.2.7 SIMD-BP128

> Daniel Lemire, Leonid Boytsov: Decoding billions of integers per second through vectorization. Softw., Pract. Exper. 45(1): 1-29 (2015)

1.5.2.8 Simple8b

> Vo Ngoc Anh, Alistair Moffat: Index compression using 64-bit words. Softw., Pract. Exper. 40(2): 131-147 (2010)

1.5.2.9 Simple16

> Jiangong Zhang, Xiaohui Long, and Torsten Suel. 2008. Performance of compressed inverted list caching in search engines. In Proceedings of the 17th international conference on World Wide Web (WWW '08). ACM, New York, NY, USA, 387-396. DOI: https://doi.org/10.1145/1367497.1367550

1.5.2.10 StreamVByte

> Daniel Lemire, Nathan Kurz, Christoph Rupp: Stream VByte: Faster byte-oriented integer compression. Inf. Process. Lett. 130: 1-6 (2018). DOI: https://doi.org/10.1016/j.ipl.2017.09.011

1.5.2.11 Varint-G8IU

> Alexander A. Stepanov, Anil R. Gangolli, Daniel E. Rose, Ryan J. Ernst, and Paramjit S. Oberoi. 2011. SIMD-based decoding of posting lists. In Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM '11), Bettina Berendt, Arjen de Vries, Wenfei Fan, Craig Macdonald, Iadh Ounis, and Ian Ruthven (Eds.). ACM, New York, NY, USA, 317-326. DOI: https://doi.org/10.1145/2063576.2063627

1.5.2.12 VarintGB

> Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems: invited talk. In Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM '09), Ricardo Baeza-Yates, Paolo Boldi, Berthier Ribeiro-Neto, and B. Barla Cambazoglu (Eds.). ACM, New York, NY, USA, 1-1. DOI: http://dx.doi.org/10.1145/1498759.1498761

1.6 Query an index

1.6.1 Usage

```
queries - a tool for performing queries on an index.
Usage: ./bin/queries [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  --config TEXT               Configuration .ini file
  -t,--type TEXT REQUIRED     Index type
  -a,--algorithm TEXT REQUIRED
                              Query algorithm
  -i,--index TEXT REQUIRED    Collection basename
  -w,--wand TEXT              Wand data filename
  -q,--query TEXT             Queries filename
  --compressed-wand           Compressed wand input file
  -k UINT                     k value
  -T,--thresholds TEXT        k value
  --terms TEXT                Text file with terms in separate lines
  --nostem Needs: --terms     Do not stem terms
  --extract                   Extract individual query times
  --silent                    Suppress logging
```

Now it is possible to query the index. The command `queries` parses each line of the standard input (or a file if `-q` present) as a tab-separated collection of term IDs (or words if `--terms` present), where the i-th term is the i-th list in the input collection.

```
$ ./bin/queries -t opt -a and -i test_collection.index.opt -w test_collection.wand -q␣
↪../test/test_data/queries
```

This performs conjunctive queries (`and`). In place of `and` other operators can be used (`or`, `wand`, ..., see `queries.cpp`), and also multiple operators separated by colon (`and:or:wand`).

If the WAND file is compressed, please append `--compressed-wand` flag.

1.6.2 Build additional data

To perform BM25 queries it is necessary to build an additional file containing the parameters needed to compute the score, such as the document lengths. The file can be built with the following command:

```
$ ./bin/create_wand_data -c ../test/test_data/test_collection -o test_collection.wand
```

If you want to compress the file append `--compress` at the end of the command. When using variable-sized blocks (for VBMW) via the `--variable-block` parameter, you can also specify lambda with the `-l <float>` or `--lambda <float>` flags. The value of lambda impacts the mean size of the variable blocks that are output. See the VBMW paper (listed below) for more details. If using fixed-sized blocks, which is the default, you can supply the desired block size using the `-b <UINT>` or `--block-size <UINT>` arguments. Note that if using fixed/variable sized blocks, and the `-l` or `-b` parameters are not set, the default parameters will be used from the configuration file `configuration.hpp`.

1.6.3 Query algorithms

1.6.3.1 AND

1.6.3.2 OR

1.6.3.3 MaxScore

Howard Turtle and James Flood. 1995. Query evaluation: strategies and optimizations. Inf. Process. Manage. 31, 6 (November 1995), 831-850. DOI=http://dx.doi.org/10.1016/0306-4573(95)00020-H

1.6.3.4 WAND

Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In Proceedings of the twelfth international conference on Information and knowledge management (CIKM '03). ACM, New York, NY, USA, 426-434. DOI: https://doi.org/10.1145/956863.956944

1.6.3.5 BlockMax WAND

Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (SIGIR '11). ACM, New York, NY, USA, 993-1002. DOI=http://dx.doi.org/10.1145/2009916.2010048

1.6.3.6 BlockMax MaxScore

1.6.3.7 Variable BlockMax WAND

Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17). ACM, New York, NY, USA, 625-634. DOI: https://doi.org/10.1145/3077136.3080780

1.7 Document Reordering

1.7.1 Document name & URL based ordering

To reorder an inverted index based on document names or their URLs, use the `generate_sorted_docids_mapping.py` script and the `shuffle_docids` command.

The first one allows to generate a file in which each line maps a `<current ID>` with a `<new ID>`. These identifiers are generated based on the lexicographical order of document names (or their URLs). Furthermore, this file serves as input to the `shuffle_docids` command, which materializes the reordering operation.

1.7.1.1 1. Mapping file creation

To generate the mapping file, it is necessary to take into account that each line number of the `.documents` and `.urls` files (which are part of the forward index) correspond to its docid. So, the first document of `.documents` (line 0, and therefore docid = 0) is equivalent to the first URL in the `.urls` file, and so on. In this way, if you want to generate the mapping file to perform a reordering using either files, you should use the following script:

```
usage: script/generate_sorted_docids_mapping.py [-h] documents output

Take a text file lexicon (e.g. '.documents' or '.urls' file) and sort it,
generating a file mapping '<current ID> <new ID>' to use with the
'suffle_docids' script.

positional arguments:
  documents   File containing one document (or URL) per line and where each
              line number (starting from zero) represents its docid
  output      Output file mapping '<current ID> <new ID>'

optional arguments:
  -h, --help  show this help message and exit
```

For example:

```
$ python script/generate_docids_sorted_map.py path/to/inverted.urls mapping.txt
```

Note that in the example the `.urls` file is used for reordering. If you want to generate the mapping based on document names, the `.documents` file should be used instead.

1.7.1.2 2. Index remapping

Once the mapping file has been created, you must use the `shuffle_docids` command to generate the new inverted index:

```
Usage: ./bin/shuffle_docids <collection basename> <output basename> [ordering file]
Ordering file is of the form <current ID> <new ID>
```

For example:

```
$ mkdir -p path/to/ordered
$ ./bin/shuffle_docids path/to/inverted path/to/ordered/inverted mapping.txt
```

1.7.2 Random Ordering

If you want to perfom a random ordering, use the `shuffle_docids` command (described in the previous section), but without specifying the `[ordering file]` option.

1.7.3 Recursive Graph Bisection

Recursive graph bisection algorithm used for inverted indexed reordering.

1.7.3.1 Description

Implementation of the *Recursive Graph Bisection* (aka *BP*) algorithm which is currently the state-of-the-art for minimizing the compressed space used by an inverted index (or graph) through document reordering. The algorithm tries to minimize an objective function directly related to the number of bits needed to storea graph or an index using a delta-encoding scheme.

L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, andA. Shalita. Compressing graphs and indexes with recursive graph bisec-tion. InProc. SIGKDD, pages 1535–1544, 2016.

1.7.3.2 Usage

```
Recursive graph bisection algorithm used for inverted indexed reordering.
Usage: ./bin/recursive_graph_bisection [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -c,--collection TEXT REQUIRED
                              Collection basename
  -o,--output TEXT            Output basename
  --store-fwdidx TEXT         Output basename (forward index)
  --fwdidx TEXT               Use this forward index
  -m,--min-len UINT           Minimum list threshold
  -d,--depth INT in [1 - 64] Excludes: --config
                              Recursion depth
  -t,--threads UINT           Thread count
  --prelim UINT               Precomputing limit
  --config TEXT Excludes: --depth
                              Node configuration file
  --nogb                      No VarIntGB compression in forward index
  -p,--print                  Print ordering to standard output
```