GDCC 2021 Student Test (Test 5)

The task for the student category is to optimize a given binary statistical compressor using its parameter file. The goal is to minimize the compressed-data size *c_size* for the Test 1 (private) data set using this optimized compressor. We therefore rank tuned compressors only on the basis of the compression ratio and ignore processing speed. Also, unlike the main GDCC categories, we **ignore the compressed size of the decompressor**.

The software we provide implements an order-2 binary statistical compressor. To achieve the goal, participants must optimize the graph and counters of the statistical model, presented as a finite-state machine (FSM), so the resulting compressor minimizes *c_size* for the Test 1 data. They must submit a parameter file (or a generator program for a parameter file) with an FSM description of the compressor that enables the coder to perform lossless encoding and decoding of the Test 1 data.

If a participant provides a generator program that creates a parameter file, we run the program on the (private) Test 1 data, fetch the resulting parameter file, and run the compressor using this file. Generation of the parameter file must take less than 10 hours on the test hardware.

Task Materials and Notes

1. Test 5 Necessary Files

Download FSM_v1.7z compressor: https://www.gdcc.tech/wp-content/uploads/FSM_v1.7z

Download the Test 1 sample data, which is similar to the Test 1 private (hidden) data set: https://mega.nz/file/Q2wVVSaZ#iXX0gtPrOkuKPatM-QXFKE7perZKrxuM7PNS1XvVQuI

Compile coder.cpp (simple g++ coder.cpp should be sufficient), then run "coder c test1_demo T5data.cmp FSM1.txt" to encode the file and "coder d T5data.cmp T5data.unp FSM1.txt" to decode.

Generate (or submit a generator program for) an FSM.txt file, by which the coder losslessly encodes and decodes data while achieving a minimal compressed size.

Sample variants of the parameter file appear in the provided archive, corresponding to different finite-state machines. Use this data to find the maximum-compression parameter file, or create your own generator to produce FSM configuration files.

2. Compressor Description

2.1. The file coder.cpp implements an order-2 binary statistical compressor. It works by collecting contextual statistics, predicting the probability of next bit being 0, and encoding the bits using a version of arithmetic coding known as *rangecoder.*

http://mattmahoney.net/dc/dce.html#Section_412

2.2. The parameter file FSM.txt contains *N* lines, each with a comma-delimited triplet of decimal numbers, where *N*<=32,768. The numbers, from first to third, correspond to the following:

* Next state in case of bit=0

* Next state in case of bit=1

* Probability of bit=0 multiplied by 32,768

2.3. The model's basic prediction component is called a *counter,* and it can have many implementations ranging from simply counting the occurrences of 0 and 1 to estimating the next probability as SCALE*n0/(n0+1).

Because of the ratio/speed/memory tradeoff, however, numerous counter implementations have been developed for various compression algorithms and data types.

2.4. Many counter implementations can be represented in a universal way: since memory usage is an important point and the interface is fixed, we can limit a counter's memory usage (15 bits in coder.cpp) and implement the probability estimation and state update simply by filling a parameter table with an arbitrary algorithm. Such a representation is called an FSM (see https://en.wikipedia.org/wiki/Finite-state_machine).

3. Approaches

3.1. An FSM can be viewed as a directed graph (https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)) with states appearing as nodes and state transitions appearing as links. Therefore, some known algorithm from graph theory may be applicable to some approximation of given price function (based on entropy estimation).

3.2. Reduce any counter implementation's precision such that 15 bits is enough to store the state variables, and then calculate the probability estimation, update each state, and fill FSM.txt.

Parameters can be optimized simply by generating a corresponding FSM.txt file and testing the compression until an optimal combination emerges, but many smarter optimization methods can, of course, find the best result much faster.

3.3. The whole FSM configuration can be treated as a parameter set and optimized as a black box without first designing a base algorithm.

3.4. These two strategies can be combined by generating an initial configuration with some algorithm then further refining the results using different methods—for example, generate a redundant FSM with a higher initial number of states, then merge state pairs until the maximum allowed number of states is reached.

3.5. FSM states can be interpreted as lossy-compressed context histories. Probability estimation from a state is easy in this case using any slow NN/ML or CM model that can estimate the probability of the next bit from a given bit string.

Submission Requirements

1. The submission must be either a text file representing the FSM's configuration for the compressor model or a generator program for such a text file. In the latter case, we decline to reveal any information about the resulting parameter (configuration) file.
2. Submissions shall be emailed to submission@gdcc.tech
3. Each submission email should contain the following information:

   - Author name
   - Preferred contact email address and, optionally, additional contact information
   - "Test 5" string
   - URL for the parameter file or generator, or an attachment containing it

For example:

John Snow
johnsnow97@mail.com
Test 5
https://www.cloud.com/file_path

4. If you submit a parameter-file generator, it must meet the following requirements:

   - The file shall be a Linux or Windows executable or a Python script (check our test-hardware specification)
   - The first command-line parameter shall be the name of input-data file for analysis (specifically, the Test 1 private data file); the second parameter shall be the name of the output parameter file.

5. You may update your submission to Test 5 as many as 14 times (for a maximum total of 15 test runs, including the initial submission). Although we attempt to test submissions beyond the fourteenth update, especially for the top competitors on the interim leaderboard, we make no guarantees.