Efficient implementation of ACB compression algorithm for ExCom library

Author: Ing. Michal Valach **Supervisor**: doc. Ing. Jan Holub, Ph.D. Czech Technical University in Prague

Motivation

ACB is an unexplored context-based compression method. It is called *Associative Coder of Buyanovsky* or in short *ACB*, published by *George Buyanovsky* in 1994 [1]. The article was written in Russian, and the compression method was called *Associative coding*. *George Buyanovsky* also created an implementation of this algorithm, but distributed it only as a *MS-DOS* executable.

ExCom library

The *ExCom* is a library containing many different compression method, for example PPM, DCA, LZ-family, Huffman coder, and many others. It was created by Filip Šimek in his master's thesis [4] to become a library you would use whenever you need to use any compression method. It is written in C++ with strong emphasis on simple usage, future extensibility, and thread safety. It was published under *GNU LGPL* license

Related work

There are few implementations, except the one presented by George Buyanovsky. Their creators are Lukáš Cerman in 2003 [2], Martin Děcký in 2006 [3], and Filip Šimek in 2009 [4]. They are mostly only simple implementations including only part of the algorithm and they are also very slow, except the implementation by Cerman.

ACB algorithm

ACB uses LZ77-like sliding window. Left part of the sliding window is called the context, and the right part is called the content part of the context. While compression or decompression proceeds, new pairs of contexts and contents are created. Each such pair is added to a dictionary. The dictionary consists of context-content pairs, which are sorted lexicographically by their context from the last symbol to the first one. One compression cycle is shown below. The text before the pipe represents already encoded text.

`swiss m|iss is missing`

Sliding window

1. Best context $c_{tx} = 3$
2. Best content $c_{nt} = 6$
3. Common length $l = 4$ symbols
4. Output consists of (d, l, c) where
   - $d = c_{nt} - c_{tx} = 6 - 3 = -3$
   - $l = 4$
   - $c =$'i'
5. Update dictionary

	Dictionary Context|Content
0	|swiss m
1	swiss |m
2	swi|ss m
3	s|wiss m
4	swis|s m
5	swiss|m
6	sw|iss m

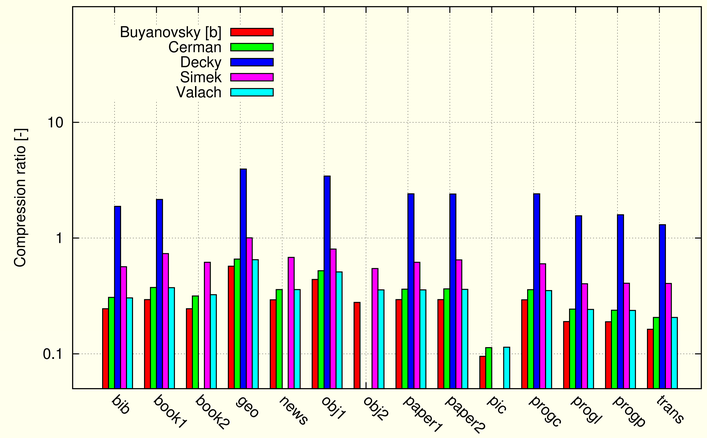
Implementation

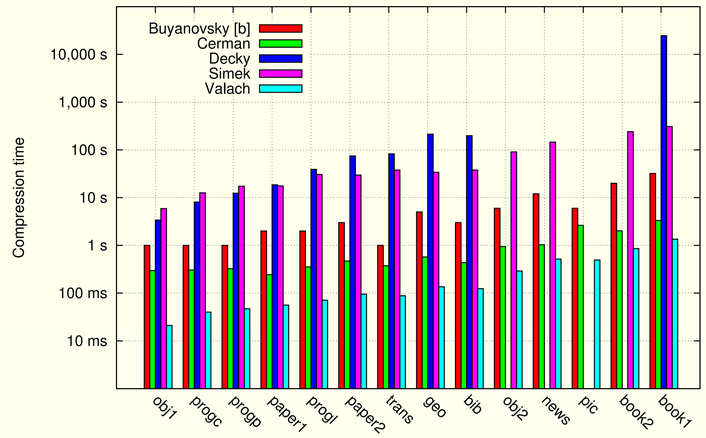
The dictionary is implemented using the suffix tree and the suffix array. The suffix array is implemented using two different representations: a simple two dimensional array and the B+ tree. The B+ tree is a modification of a B tree, where all the data are stored in the leafs to make simple access to neighbourhood data, and also the tree leafs are linked together.

Encoding of the data produced by the main ACB algorithm, the triplets, must be done, otherwise the compression gained by the algorithm would be lost. There are two parts of the encoding process. The first part is to choose what to store into the output stream to not loose any information. The second part is how to compress the output stream (how to compress items of the triplets). Adaptive Huffman coder and adaptive arithmetic coder are presented as viable coders for the triplets.

Experiments

The figures below show the comparison of all presented ACB implementations. The figure on the left compares the compression ratio. The figure on the right compares the compression time. The decompression times are almost identical to the compression times. The tests were run on Calgary corpus files.





The implementation by Buyanovsky is superior to all by the means of the compression ratio. The implementation by Cerman gives similar results as the implementation in this thesis. It is better for two files, which can be explained by differently setted default parameters. The other two implementations are not challengable.

As you can see, the implementation by Děcký provides the worst time performance among the implementations. For small files it performs better than the implementation by Šimek, but with the increasing size of the files it is far inferior and unusable. The best time performance is provided by the implementation presented in this thesis. It is approximately 400 times faster than the previous implementation by Šimek, 4–10 times faster then the implementation by Cerman, and 10–20 times faster than the implementation by Buyanovsky.

Conclusion

Experiments and performance tests showed, that the implementation based on this thesis is not better, than the implementation by George Buyanovsky in terms of the compression ratio. It only acknowledges that the known description of the algorithm is not accurate. On the other hand, it gives better results in comparison to the other known implementations.

The part of the algorithm called *sorted contents* was also implemented, but the test results showed, that it gives neither better compression ratio, nor compression time. So, it is by default disabled, but can be turned on by using compiler directives.

References

[1] G. M. Buyanovsky, "Associative Coding," *Monitor*, pp. 10–22, 1994.

[2] L. Cerman, "Acb compression algorithm," Czech Technical University in Prague, Jan. 2003.

[3] M. Děcký, "Associative coder of buyanovsky," 2006. [Online]. Available: {http://projects.decky.cz/ACB/}

[4] F. Simek, "Data compression library," Master's thesis, Czech Technical University in Prague, May 2009.