UDC 004.627

Serhiy USTENKO, Serhiy LUKYANCHIKOV
Mykolaiv
ustenko.s.a@gmail.com, lsd57@ukr.net

APPLICATION DICTIONARY COMPRESSION METHOD FOR COMPRESSION OF AUDIO DATA WITH ADJUSTABLE LOSS

The work is dedicated to the development of audio data compression method with adjustable losses using pseudo-difference analogs using dictionaries

Keywords: dictionary, audio samples, the mechanism of compression, compression algorithm, lossy compression.

By the moment, lossy compression methods are mostly used for effective audio data compression, which ones can be classified as psycho-acoustic methods (MPEG Audio, OGG Vorbis, AAC), and other ones based on fractals, wavelets and stochasic differential equations [1].

On the other hand, dictionary-based methods of lossless compression, like Lempel-Ziv method [2] and its modifications [3-6] give only minor compression, and sometimes compression cannot be reached. Some attempts to develop dictionary-based method which could solve this problem were presented in [7], where projectional compression method had been proposed. Although, after practical experiments of using this method for compressing audio data, it appeared that it gives almost the same compression rates as Lempel-Ziv method, also being much more resource-consumable.

For reaching higher compresion rates in such applications it is possible to use lossy dictionary-based compression methods. The goal of current presentation is to demonstrate method which could allow to do this.

Let's call compression method being presented here as method of differential pseudo-analogs. During compression process, audio information is accepted as a stream of amplitude values of the sound signal (audio samples). Coder contains a dictionary buffer having capacity of L audio samples. At the start of compression process the dictionary is being filled by audio samples from the input stream. Coder also contains special input buffer (audio samples cache) having capacity of l audio samples ($l << L$) which is being constantly filled up from input stream. After filling up audio samples cache algorithm tries to find differential pseudo-analogs of audio samples cache contents inside the dictionary.

Let's assume differential pseudo-analog (DPA) having length p a block of $p \le l$ audio samples from the dictionary which matches the following condition:

$$\left| q_{i+j} - s_i - r \right| \le \delta,\ i = \overline{1, p}, \tag{1}$$

where q_{i+j} – audio samples from the dictionary; j – DPA position inside the dictionary; s_i – appropriate audio samples from the cache; r – average DPA difference; δ – given acceptable error of audio sample value.

Average DPA difference is an arithmetic mean between values of appropriate audio samples from DPA and the cache:

$$r = \frac{1}{p} \sum_{i=1}^{p} q_{i+j} - s_i .$$

Thus, DPA audio sample values differs from corresponding values of cache audio samples by the difference r in range of given acceptable error δ. If $r = 0$, then such DPA is called simple pseudo-analog (SPA).

Compression mechanism sense is that once DPA has found, only its position j inside the dictionary, length p and difference r is saved instead of corresponding part of the cache. Thus, part of the samples cache is replaced with corresponding representation from inside the dictionalry. In this case if $\delta > 0$, then it's obvious that lossy compression is being performed.

There are 3 differrent situations possible durig compression progress:

– DPA has been found ($r \neq 0$) – need to save j, p and r;

– SPA has been found ($r = 0$) – need to save only j and p;

– neither of DPA/SPA have been found – need to exclude first audio sample from the cache and save it as is, without any compression.

Thus, results of the compression will have one of the following types: DPA, SPA or the separate audio sample from the cache. In order to save compression result the following bit structures are proposed:

– DPA:

1	1	j	p	r

– SPA:

1	0	j	p

– A separate audio sample from the cache:

0	s_1

The left-most bit of those structures allows to distinguish pseudo-analog from the audio sample from cache. The next bit allows to distinguish DPA from SPA and determine if there is a need to read the difference r at the end.

Structure field sizes for j and p are calculated using the following:

$$size(j) = ceil(\log_2 L),$$
$$size(p) = ceil(\log_2 l),$$

where $ceil$ – is an operation of getting the closest larger integer; L, l – number of audio samples in the dictionary and cache respectively.

Sizes of fileds r and s_1 are the same and equal to the bit size b of audiostream samples. If value of r is impossible to place in b bits:

$$ceil(\log_2 r) > b, \quad (2)$$

then such DPA should be discarded.

Let's state an algorithm of compressing the audio samples stream as a set of steps. At the start of the compression process the size L of the dictionary, size l of audio samples cache and acceptable audio samples value error δ are being preset. Algorithm accepts the stream of audio samples and generates the stream of compressd results as an output (archive).

Compression algorithm.

Step 1. Fill the dictionary with L audiosamples from the input stream and save initial dictionary state in the archive.

Step 2. Set current fill w of the audiosamples cache: $w = 0$ (cache is empty).

Step 3. Refill audiosamples cache from the input stream updating current fill w of the cache.

Step 4. According to the bit structure for saving DPA, calculate minimum number z of audiosamples in DPA which will give at least minimal comporession:

$$z = ceil\left[\frac{2 + size(j) + size(p)}{b} + 1\right].$$

Step 5. According to the condition (1) for p from w to z search DPA having length p in entire contents of the dictionary, discarding DPA matching condition (2). Amon all found DPAs select the one matching

$$\sum_{i=1}^{p} \left| q_{i+j} - s_i \right| \to \min,$$

where q_{i+j} – audiosamples from the dictionary; j – DPA position inside the dictionary; s_i – corresponding audiosamples from the cache.

Step 6. If DPA has been found during Step 5 then:

– store the values of j, p and the difference r;

– if $r \neq 0$, then save DPA to the archive using appropriate bit structure; else – save DPA using SPA structure;

– push DPA found to the dictionary (which should be organized as FIFO buffer).

If DPA has not been found then:

– assume DPA length as $p = 1$;

– save only first sample from the cache to the archive using appropriate bit structure;

– remove this audiosample from the cache and push it to the dictionary.

Step 7. Remove processed part of the cache having length p and refresh the cache fill: $w = w - p$.

Step 8. If the input stream end isn't reached, go to Step 3.

Step 9. End.

Restoring audio samples from the archive is much more trivial task. All you need is just read the initial state of the dictionary from the archive, and then read coded compression results and decode them performing following actions:

– if DPA or SPA data has obtained (, and), then restore corresponding block of audio samples based on dictionary contents, wrire it to the resulting audio stream and push to the dictionary;

– if the separate audiosample has obtained then write it as is to the resulting audio stream and push to the dictionary.

The actions above should be performed until end of the archive has been reached.

Proposed method of differential pseudo-analogs (MDPA) had been implemented programmatically using C programming language. Also, LZSS [4] method had been implemented (one of modifications of Lempel-Ziv method). There were experiments performed with compressing monofonic audio files having RAW PCM format, 8 bit/sample, 22kHz using both methods implementations. Tables 1 and 2 contain results of compressing audio files containing real musical recordings:

1) voodooch.raw – "Voodoo child" Jimi Hendrix song (blues/rock-n-roll music).
2) hero.raw – "Hero" song of Nickelback rock band (rock music);
3) oskolok.raw – "Oskolok L'da" song of ARIA rock band (hard-rock music);
4) disciple.raw – "Disciple" song of Slayer rock band (thrash metal).

Compression results are obtained using method of differential pseudo analogs using following parameter values:

1) dictionary size $L = 125$ audio samples;
2) audio samples cache size $l = 7$ samples;
3) acceptable audio sample values error $\delta = 1$ (minimal losses).

Compression results using LZSS methods are obtained using dictionary size 4096 chars and input/output buffer size 16 chars.

Table 1

Audio files sizes before and after compression

Audio file	Size before compression, bytes	Size after compression, bytes	
		MDPA	LZSS
voodooch.raw	6 964 417	2 971 696	4 452 455
hero.raw	3 254 977	1 867 521	2 681 712
oskolok.raw	7 181 569	4 174 509	6 017 960
disciple.raw	4 757 761	3 955 270	4 874 740

Table 2

Reached compression rates

Audio file	Comression rate	
	MDPA	LZSS
voodooch.raw	2,34	1,56
hero.raw	1,74	1,21
oskolok.raw	1,72	1,19
disciple.raw	1,20	0,98

As you can see, both methods (MDPA, LZSS) have the common tendency: compression rates are lower for the harder music styles. At the same time, even in case of minimal losses, MDPA allows to reach higher compression rated than LZSS using much more compact dictionary size.

The only backdraw of MDPA is the distortion of initial audio stream. The noise with amplitude δ appears (δ – acceptable error of audio sample values). For the recordings with 8 bits per sample the noise is audibly noticable even in case of $\delta > 1$. This issue limits the usage of the proposed method for the effective audio data compression by the moment.

Further investigations are planned to increase efficiency of the proposed compression method to eliminate the noise and increase data compression rates.

References

1. Приходько С.Б. Сжатие звука на основе стохастических дифференциальных уравнений второго порядка // Вестник Херсонского государственного технического университета. – Херсон: ХГТУ. – 2002. – Вып. 2(15). – С.386-388.
2. Ziv J., Lempel A. A universal algorithm for sequental data compression // IEEE Transaction on Information Theory – 1977. – Vol.23(3). – P.337-343.
3. Ziv J., Lempel A. Compression of individual sequences via variable-rate coding // IEEE Transaction on Information Theory – 1978. – Vol.24(5). – P.530-536.
4. Storer J.A., Szymanski T.G. Data compression via textual substitution // Journal of ACM – 1982. – Vol.29(4). – P.928-951.
5. Welch T.A. A technique for high-performance data compression // IEEE Computer – 1984. – Vol.17(6). – P.8-19.
6. Bender P., Wolf J. New asymptotic bounds and improvements on the Lempel-Ziv data compression algorithm // IEEE Transactions on Information Theory – 1991. - Vol.37(3). - P.721-727.
7. Кириченко Н.Ф., Лепеха Н.П., Попив И.А. Допустимая аппроксимация функций дискретного аргумента и ее применение к сжатию информации // Проблемы управления и информатики. – 1998. – №5. – С.113-127.

Сергій УСТЕНКО, Сергій ЛУКЬЯНЧІКОВ
м. Миколаїв

ЗАСТОСУВАННЯ СЛОВНИКОВИХ МЕТОДІВ СТИСНЕННЯ ДЛЯ КОМПРЕСІЇ АУДІО ДАНИХ З РЕГУЛЬОВАНИМИ ВТРАТАМИ

Робота присвячена розробці метода стиснення аудіо даних з регульованими втратами за допомогою різницевих псевдо-аналогів з використанням словників.

Ключові слова: словники, аудіосемпли, механізм стиснення, алгоритм стиснення, стиснення з втратами.

Сергей УСТЕНКО, Сергей ЛУКЬЯНЧИКОВ
г. Николаев

ПРИМЕНЕНИЕ СЛОВАРНОГО МЕТОДА СЖАТИЯ ДЛЯ КОМПРЕССИИ АУДИО ДАННЫХ С РЕГУЛИРУЕМЫМИ ПОТЕРЯМИ

Работа посвящена разработке метода сжатия аудио данных с регулируемыми потерями с помощью разностных псевдоаналогов с использованием словарей.

Ключевые слова: словари, аудиосемплы, механизм сжатия, алгоритм сжатия, сжатие с потерями.

Article received editorial board 05.10.2016