ISSN 2079-3316 **ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ** **т. 13**, № 2(53), с. 3–33
научная статья — программное и аппаратное обеспечение распределенных и суперкомпьютерных систем

УДК 004.627+004.272.45
10.25209/2079-3316-2022-13-2-3-33;

Сжатие сенсорных данных с малым расходом оперативной памяти

Юрий Владимирович **Шевчук**✉
Институт программных систем им. А. К. Айламазяна РАН, Веськово, Россия,
sizif@botik.ru✉ (*подробнее об авторе на с. 32*)

Аннотация. Рассматривается задача сжатия скалярных данных в узлах сенсорной сети в потоковом режиме (без накопления блока некомпрессированных данных). Рассмотрено несколько экспериментальных алгоритмов сжатия, основанных на сочетании дельта-кодирования (LPC) с кодированием повторов (RLE). На стадии статистического кодирования применялись: a) код переменной длины с динамическими префиксами, полученными с помощью MTF-преобразования, b) адаптивный бинарный код, c) адаптивные коды Голомба-Райса. Проведено сравнение известных и экспериментальных алгоритмов на 75 источниках сенсорных данных. В тестах достигнуты коэффициенты сжатия порядка 1.5/4/1000000 (мин/медиана/макс) при размере контекста алгоритма сжатия порядка 10 байт.

Ключевые слова и фразы: LPC, линейное прогнозирующее кодирование, DTN, сеть устойчивая к разрывам, сеть с эпизодической связностью, распределение Лапласа, адаптивный алгоритм сжатия, стопка книг, MTF-преобразование, RLE, RLGR, префиксный код, Гамма-код Элиаса, код Голомба-Райса, код vbinary

Благодарности:
Работа выполнена в рамках госзадания по теме AAAA-A19-119020690043-9.

Для цитирования: Шевчук Ю. В. *Сжатие сенсорных данных с малым расходом оперативной памяти* // Программные системы: теория и приложения. 2022. **Т. 13**, № 2(53). С. 3–33. `http://psta.psiras.ru/read/psta2022_2_3-33.pdf`

Введение

Для проекта из области сенсорных *сетей с эпизодической связностью* (DTN [1]) мы рассматриваем задачу накопления данных в сенсорном узле со сравнительно большим числом датчиков $n \sim 10 \div 100$.

This Article in English: *http://psta.psiras.ru/read/psta2022_2_35-63.pdf*

Мы хотим использовать сжатие для увеличения объема данных, вмещающихся в память узла, чтобы увеличить максимально допустимое время между сеансами связи, при котором еще не происходит потери данных.

Сжатие данных популярно в сенсорных сетях, поскольку позволяет сократить объем передачи данных между узлами, получая экономию (a) ресурса автономного источника энергии и (b) пропускной способности сети. Разработан ряд алгоритмов сжатия, учитывающих специфику сенсорных узлов: ограниченный объем памяти (таблица 1) и вычислительных ресурсов [2–5]. Вопреки ожиданиям, не удалось найти готового алгоритма, подходящего для поставленной задачи.

Таблица 1. Объем ОЗУ сенсорных узлов

Год	Процессор	ОЗУ
2006	MSP430	10кБ
2020	NRF52	256кБ

Таблица 2. Размеры входного блока некоторых алгоритмов сжатия

Алгоритм	Размер (байт)
`S-LZW` [2]	528
`FELACS` [4]	$16 \div 300$
`FLAC` [6]	$2048 \div 6144$
`Shorten` [7]	256

Большинство алгоритмов работают с входными блоками данных определенного размера (таблица 2), а размер выходного блока переменный и определяется фактически достигнутым коэффициентом сжатия. Для нашей задачи желателен алгоритм, который принимает входные данные в виде потока время от времени прибывающих значений и наполняет *выходной буфер заданного размера*. Размер может быть выбран из соображений атомарной передачи за один сеанс связи, или из соображений управления памятью (пул буферов фиксированного размера). Также требуется, чтобы объем данных состояния алгоритма сжатия был небольшим, поскольку он умножается на число датчиков n, и чтобы хорошо сжимались «прогоны» — интервалы постоянных значений $v = \text{const}$, которые часто встречаются в реальных сенсорных данных.

Сенсорные данные представляют собой временны́е ряды

$$\{(t_0, v_0), (t_1, v_1), \dots\}.$$

Авторы, занимающиеся сжатием данных на уровне базы данных, дополнительно рассматривают вопрос сжатия отметок времени t_i [8, 9].

В нашей задаче мы исходим из того, что сенсорный узел считывает данные по заданному расписанию: $t_i = \mathbf{schedule}(t_0, i)$, данные не теряются, поэтому вопрос сжатия последовательности t_i не стоит; достаточно передать t_0 с каждым блоком компрессированных данных.

1. Методология

Мы будем рассматривать алгоритмы сжатия, выбранные с учетом требования малого расхода памяти и ресурсов процессора. Для каждого алгоритма мы реализуем прототип кодера, позволяющего измерить коэффициент сжатия, и тестируем кодер на реальных сенсорных данных: собственных данных (таблица 3) и данных контрольных ваттметров («ground truth») проекта EMBED [**10**].

Таблица 3. Описание своих тестовых наборов данных

Имя	Бит	Частота	Описание
Pa	18	1/10Гц	данные учета общего потребления электроэнергии коттеджа: средняя активная потребляемая мощность, полученная дифференцированием показаний счетчика электроэнергии на базе микросхемы *Analog Devices ADE7758*[URL], фаза A
Pb	18	1/10Гц	то же, фаза B
Pc	18	1/10Гц	то же, фаза C
T1	12	1/60Гц	температура воздуха в помещении 1 (датчик *Sensirion SHTW2*[URL])
T2	12	1/60Гц	температура воздуха в помещении 2
Tdp1	12	1/60Гц	точка росы воздуха в помещении 1
Tdp2	12	1/60Гц	точка росы воздуха в помещении 2

Для каждого собственного источника мы рассматриваем три серии продолжительностью три дня каждая, чтобы оценить изменение характера данных во времени. Все результаты тестирования сведены в общую таблицу 4 и обсуждаются по ходу дальнейшего изложения. Алгоритмы описываются неформально, но реализации прототипов на языке Perl5 свободно доступны: *https://github.com/yysizif/sencomp*[URL].

Таблица 4. Коэффициенты сжатия данных 75 источников 16-ю алгоритмами

Цвет фона: вне конкурса | лучшее в строке сжатие | -5% | -10% | -20% | -50% | -80% | -90%

№	Источник	z22	z1	EB	F16	F256	RLGR	EG	EW	e1	e2	e3	e4	e5	DD	RLGR′	e10
1	период 1, Pa	5.21	3.12	2.43	1.16	1.18	1.49	1.14	1.23	1.47	1.62	1.61	1.59	1.51	1.48	1.56	1.61
2	период 1, Pa-шум	4.90	3.29	2.53	1.16	1.18	1.55	1.18	1.28	1.59	1.70	1.70	1.64	1.57	1.61	1.57	1.64
3	период 1, Pb	4.99	3.36	4.02	1.93	1.89	0.31	1.81	1.74	2.11	2.25	2.21	2.19	2.17	2.13	1.76	2.28
4	период 1, Pb-шум	5.96	4.24	4.95	1.96	1.90	0.67	2.20	2.15	2.72	2.78	2.77	2.70	2.74	2.68	2.19	2.78
5	период 1, Pc	5.43	3.67	4.85	2.39	2.52	1.92	2.13	1.94	2.58	2.58	2.57	2.60	2.54	2.48	2.39	2.71
6	период 1, Pc-шум	7.68	4.85	6.86	2.45	2.56	2.69	2.91	2.65	3.55	3.39	3.39	3.55	3.60	3.38	3.08	3.65
7	период 1, Tdp1	3.85	3.63	4.30	2.05	2.20	2.30	1.81	1.64	2.14	2.02	2.02	2.08	2.03	2.04	2.25	2.28
8	период 1, Tdp1-шум	6.65	5.03	9.52	2.57	2.57	5.00	3.99	3.57	4.78	4.46	4.46	4.78	4.86	4.54	4.90	5.20
9	период 1, T1	4.52	3.87	5.05	2.23	2.44	2.50	2.06	1.86	2.31	2.18	2.18	2.26	2.23	2.23	2.45	2.51
10	период 1, T1-шум	7.04	5.27	12.71	2.99	3.34	6.59	5.30	4.69	6.33	5.85	5.85	6.34	6.50	6.01	6.48	6.84
11	период 1, Tdp2	4.17	3.58	4.13	2.00	2.17	2.24	1.75	1.59	2.08	1.98	1.98	2.05	2.01	1.99	2.20	2.22
12	период 1, Tdp2-шум	6.34	4.89	7.71	2.38	2.51	4.06	3.26	2.94	4.02	3.73	3.73	3.97	4.04	3.82	4.00	4.23
13	период 1, T2	4.40	3.77	4.21	2.03	2.16	2.26	1.78	1.62	2.07	2.00	2.00	2.06	2.02	2.02	2.21	2.26
14	период 1, T2-шум	6.62	4.89	7.69	2.50	2.70	4.05	3.28	2.98	4.12	3.85	3.85	4.07	4.10	3.91	3.99	4.24
15	период 2, Pa	4.64	3.06	2.35	1.25	1.28	1.56	1.11	1.20	1.44	1.59	1.58	1.56	1.47	1.47	1.55	1.60
16	период 2, Pa-шум	4.25	3.17	2.42	1.25	1.28	1.55	1.13	1.23	1.52	1.64	1.64	1.58	1.51	1.55	1.55	1.61
17	период 2, Pb	4.95	3.33	3.82	1.89	1.90	0.39	1.73	1.68	2.07	2.19	2.16	2.15	2.12	2.08	1.91	2.23
18	период 2, Pb-шум	5.46	4.10	4.61	1.91	1.90	0.71	2.06	2.03	2.58	2.65	2.63	2.59	2.61	2.54	2.25	2.66
19	период 2, Pc	4.55	3.42	3.93	1.94	1.97	0.12	1.77	1.67	2.22	2.26	2.25	2.27	2.20	2.16	1.77	2.32
20	период 2, Pc-шум	5.31	3.98	4.71	1.96	1.98	0.24	2.07	1.98	2.68	2.61	2.60	2.64	2.63	2.55	1.87	2.61
21	период 2, Tdp1	4.58	3.72	4.24	2.04	2.23	2.29	1.79	1.60	2.17	2.02	2.02	2.08	2.05	2.02	2.25	2.26
22	период 2, Tdp1-шум	6.20	5.32	11.87	2.63	2.71	6.18	4.89	4.23	5.51	5.12	5.12	5.68	5.78	5.25	6.04	6.30
23	период 2, T1	4.40	3.64	3.74	1.90	2.02	2.12	1.61	1.43	2.01	1.89	1.89	1.94	1.91	1.88	2.08	2.11
24	период 2, T1-шум	6.40	4.24	7.23	2.11	2.23	3.77	2.98	2.62	3.52	3.26	3.26	3.58	3.67	3.34	3.71	3.87
25	период 2, Tdp2	4.61	4.04	6.53	2.53	2.84	2.91	2.52	2.32	2.65	2.42	2.42	2.61	2.58	2.57	2.83	2.91
26	период 2, Tdp2-шум	7.89	6.46	52.54	4.77	6.84	32.02	22.14	19.27	22.01	21.03	21.03	23.34	23.29	21.52	31.61	27.84
27	период 2, T2	5.24	4.20	6.21	2.49	2.81	2.86	2.43	2.23	2.62	2.38	2.38	2.56	2.52	2.52	2.78	2.84
28	период 2, T2-шум	8.87	5.82	92.51	5.42	8.02	48.58	39.23	34.38	40.60	38.58	38.58	43.21	43.10	39.47	47.45	48.81
29	период 3, Pa	4.46	3.07	2.30	1.07	1.08	1.55	1.09	1.19	1.43	1.57	1.56	1.53	1.45	1.46	1.55	1.58
30	период 3, Pa-шум	4.56	3.17	2.36	1.07	1.08	1.54	1.11	1.22	1.49	1.62	1.61	1.56	1.48	1.53	1.55	1.59
31	период 3, Pb	4.92	3.39	4.23	1.92	1.88	0.34	1.89	1.81	2.15	2.30	2.27	2.24	2.24	2.17	1.89	2.31
32	период 3, Pb-шум	6.08	4.43	5.51	1.96	1.89	0.62	2.43	2.36	2.91	3.02	3.00	2.92	3.00	2.86	2.46	2.98
33	период 3, Pc	4.44	3.53	4.32	2.14	2.22	0.23	1.93	1.79	2.37	2.40	2.39	2.42	2.35	2.29	1.95	2.49
34	период 3, Pc-шум	5.77	4.31	5.53	2.16	2.23	0.34	2.39	2.23	3.01	2.90	2.90	2.99	3.00	2.86	2.22	2.98
35	период 3, Tdp1	4.52	3.93	6.32	2.51	2.81	2.88	2.46	2.26	2.61	2.39	2.39	2.57	2.54	2.54	2.81	2.87
36	период 3, Tdp1-шум	7.72	6.52	55.66	4.77	6.85	33.80	23.51	20.50	23.65	22.58	22.58	25.31	25.28	23.08	33.26	29.43
37	период 3, T1	5.24	4.31	6.56	2.56	2.91	2.96	2.53	2.32	2.65	2.42	2.42	2.62	2.60	2.61	2.88	2.95
38	период 3, T1-шум	8.88	6.20	95.44	5.31	8.27	52.45	40.72	36.17	42.35	40.18	40.18	44.80	44.69	41.09	51.23	50.34
39	период 3, Tdp2	4.24	3.89	6.62	2.56	2.85	2.95	2.55	2.34	2.65	2.43	2.43	2.62	2.59	2.61	2.87	2.96
40	период 3, Tdp2-шум	7.67	6.64	52.87	4.74	6.67	32.65	22.32	19.46	22.22	21.19	21.19	23.51	23.51	21.73	32.17	27.73
41	период 3, T2	5.08	4.38	7.08	2.65	2.95	3.06	2.68	2.46	2.72	2.49	2.49	2.71	2.69	2.70	2.96	3.08
42	период 3, T2-шум	8.83	6.15	117.0	5.60	8.97	67.36	50.62	45.29	52.31	49.86	49.86	55.17	55.00	51.01	65.30	61.75

Продолжение на следующей странице

Таблица 4. Коэффициенты сжатия данных 75 источников 16-ю алгоритмами (продолжение)

Цвет фона: вне конкурса, лучшее в строке сжатие, -5%, -10%, -20%, -50%, -80%, -90%

№	Источник	z22	z1	EB	F16	F256	RLGR	EG	EW	e1	e2	e3	e4	e5	DD	RLGR′	e10
43	1/Hair Dryer	11.47	7.03	18.60	5.02	7.01	0.01	6.22	5.85	5.11	4.92	4.92	5.81	5.80	6.05	6.73	7.27
44	1/Iron	2000	1428	3407	7.81	15.68	1.00	1468	1548	1492	1642	1618	1645	1663	1243	590.0	1165
45	1/Kettle	625.0	526.0	838.0	7.72	15.06	0.20	391.1	417.1	513.0	533.0	528.0	536.0	519.0	493.1	263.1	495.1
46	1/Laptop	3.61	3.04	3.25	1.92	2.03	1.65	1.49	1.49	1.92	2.02	2.01	2.02	1.95	1.97	2.02	2.13
47	1/Modem	8.64	5.15	12.25	3.92	4.97	4.88	4.46	4.13	4.25	3.95	3.95	4.32	4.34	4.37	4.71	5.00
48	1/Monitor	8.85	5.72	12.60	4.24	5.35	3.47	4.68	4.32	4.33	4.24	4.24	4.68	4.66	4.78	5.19	5.66
49	1/NILM LCD	6.05	4.26	6.12	2.72	3.10	3.14	2.60	2.33	3.05	2.96	2.96	2.90	2.87	2.86	3.09	3.18
50	1/Refrigerator	4.63	3.67	4.22	2.26	2.41	0.001	1.90	1.78	2.49	2.46	2.46	2.50	2.44	2.38	2.55	2.65
51	1/Router	5.73	3.92	4.84	2.57	2.91	3.01	2.13	1.83	2.85	2.63	2.63	2.85	2.81	2.69	2.96	2.96
52	1/TV	38.17	28.82	52.36	6.69	11.02	2.58	21.04	18.83	22.50	21.13	21.10	23.15	22.90	22.35	22.10	24.96
53	1/Toaster	909.0	769.0	1305	7.80	15.22	0.90	612.0	648.0	807.0	858.0	850.0	861.0	832.0	806.0	351.1	800.0
54	1/Washing Machine	17.09	10.42	25.77	5.49	8.35	0.42	7.97	7.77	5.83	5.78	5.78	7.16	7.14	7.69	8.64	9.56
55	2/AC	9.07	7.40	10.46	3.54	4.15	0.17	4.79	4.76	6.25	6.39	6.32	6.42	6.15	6.14	5.59	6.58
56	2/Cablebox	7.14	4.78	7.64	3.06	3.56	3.61	3.11	2.79	3.52	3.28	3.28	3.30	3.27	3.27	3.54	3.64
57	2/Coffeemaker	9.78	6.02	15.11	4.38	5.73	0.005	5.14	4.73	4.47	4.21	4.21	4.88	4.91	5.01	5.45	6.01
58	2/Floor Lamp	10000	10000	1.5e6	8.00	16.70	7.2e5	9.4e5	1.1e6	9.9e5	9.9e5	9.9e5	1.0e6	1.0e6	1.1e6	6.0e5	8.8e5
59	2/Kitchen Light	8.08	5.10	2.04	1.06	1.08	1.53	0.97	1.10	1.49	1.50	1.50	1.48	1.40	1.48	1.52	1.53
60	2/Laptop	8.10	6.00	9.80	3.94	4.98	3.14	3.91	3.86	4.04	3.98	3.98	4.45	4.41	4.54	4.95	5.28
61	2/Microwave	8.49	5.63	12.55	4.23	5.37	2.95	4.73	4.39	4.64	4.35	4.35	4.85	4.79	4.88	5.43	5.59
62	2/Refrigerator	8.95	7.02	10.01	3.42	3.38	0.009	4.54	4.30	5.93	5.92	5.88	6.04	5.87	5.70	5.05	6.24
63	2/Speaker	7.56	5.02	9.77	3.56	4.33	4.36	3.77	3.47	3.96	3.62	3.62	3.92	3.88	3.85	4.24	4.39
64	2/TV	14.10	8.70	14.44	4.81	6.33	3.10	5.21	4.61	5.02	4.60	4.60	5.40	5.40	5.23	5.67	5.99
65	2/Toaster	1428	1250	2202	7.85	15.69	0.79	1035	1096	1307	1426	1417	1425	1372	1338	549.0	1329
66	2/xbox	11.07	6.53	15.70	4.47	5.87	2.21	5.65	5.21	5.22	4.93	4.93	5.54	5.55	5.64	6.05	6.51
67	3/DishWasher	196.1	161.1	273.1	7.50	14.44	1.48	127.0	134.1	172.1	179.1	176.1	179.1	172.1	171.1	142.1	179.1
68	3/HairDryer	18.98	11.74	28.97	6.07	9.65	0.004	9.02	8.69	6.59	6.51	6.51	8.03	8.03	8.65	9.48	10.23
69	3/Kettle	256.1	227.1	357.1	7.41	13.68	0.07	163.1	168.1	196.1	199.1	198.1	207.1	202.1	189.1	104.0	196.1
70	3/Laptop	17.54	14.58	22.51	5.37	7.70	8.75	9.93	9.36	12.02	12.08	12.05	12.18	11.95	11.86	12.23	12.96
71	3/Microwave	8.73	5.33	14.95	4.39	5.69	0.24	5.15	4.78	4.51	4.26	4.26	4.91	4.92	5.04	5.51	5.99
72	3/Refrigerator	7.21	5.10	6.65	2.77	2.45	0.007	3.01	2.87	3.93	3.94	3.91	4.01	3.90	3.81	3.33	4.15
73	3/Surface	10.48	6.22	16.74	4.76	6.27	5.79	5.64	5.32	4.68	4.54	4.54	5.36	5.35	5.56	6.02	6.70
74	3/TV	10.38	6.60	17.21	4.83	6.40	2.53	5.81	5.44	4.87	4.69	4.69	5.51	5.51	5.72	6.32	6.93
75	3/Toaster	434.1	357.1	615.0	7.60	14.39	0.20	288.1	303.1	385.1	406.1	403.1	408.1	391.1	386.1	197.1	388.1

Значения данных в тестовых наборах — действительные числа, а тестируемые алгоритмы рассчитаны на сжатие целых чисел (значений АЦП). Поэтому перед сжатием проводилось преобразование данных в целочисленную форму путем умножения на 10^k, где k—число десятичных знаков дробной части в наборе.

1.1. Вычисление коэффициента сжатия

Под коэффициентом сжатия будем понимать $K = S_0/S_c$, где S_0 — исходный объем данных в битах, S_c — объем данных после сжатия в битах, т.е. чем больше коэффициент сжатия, тем лучше результат.

При вычислении коэффициента сжатия исходный объем данных вычислялся исходя из известной разрядности АЦП $m = 12$ для источников T, T_{dp}. Для данных электрической мощности $P_{A/B/C}$ и данных проекта EMBED разрядность АЦП не определена и была вычислена исходя из диапазона измерения:

$$m = \lceil \log_2(I_{\text{макс}} V_{\text{ном}} 10^k) \rceil,$$

где $V_{\text{ном}}$ и $I_{\text{макс}}$ — номинальное напряжение электросети и максимальный измеряемый ток ($I_{\text{макс}} = 100$A для данных $P_{A/B/C}$, $I_{\text{макс}} = 15$A для данных EMBED), k — число десятичных знаков дробной части ($k = 1$ для данных $P_{A/B/C}$, $k = 2$ для данных EMBED). В обоих случаях получилась разрядность $m = 18$.

1.2. Режим сжатия

Большинство[1] алгоритмов тестируются в потоковом режиме с размером выходного буфера $B_{out} = 256$ байт. Когда выходной буфер заполняется, состояние алгоритма реинициализируется с потерей накопленных данных адаптации, чтобы была возможность независимой декомпрессии блоков сжатых данных. Предполагается, что с каждым блоком передается значение v_0 в бинарном коде с разрядностью АЦП (m); это учитывается при вычислении коэффициента сжатия.

1.3. Избыточная точность и шумопонижение

Данные температуры и влажности (T, T_{dp}) изменяются медленно и должны хорошо сжиматься методом RLE[2], но (как мы увидим ниже) этого не происходит из-за присутствия в данных небольшого уровня шумов. В данных T, T_{dp} присутствует два знака после запятой, что соответствует паспортному разрешению датчика 0.01°C [11]. Но

[1] исключения отмечены в описаниях алгоритмов

[2] RLE — Run Length Encoding, кодирование повторов («прогонов»)

повторяемость[3] результатов измерения на уровне 3σ, ограниченная уровнем шумов чувствительного элемента, составляет 0.1°C. Поэтому не имеет смысла хранить данные с максимальным разрешением, можно применить фильтрацию шумов перед сжатием. Для проверки эффекта от фильтрации в программу тестирования добавлены источники с суффиксом -шум: результаты обработки гистерезисным алгоритмом шумопонижения, приведённым на рисунке 1).

```
double hysteretic_filter(double sample)
{
    static double prev = 0;
    if (abs(sample - prev) > THRESHOLD) {
        prev = sample;
    }
    return prev;
}
```

Рисунок 1. Гистерезисный алгоритм шумопонижения. Пороговое значение `THRESHOLD=0.1` для источников T, T_{dp}, `THRESHOLD=1.0` для источников $P_{A/B/C}$

Применение шумопонижения похоже на переход к сжатию с потерями[4], но это не совсем одно и то же. При сжатии с потерями теряются детали, которые не вписываются в схему сжатия. При шумопонижении удаляются шумы, которые не имеют отношения к измеряемому параметру (температуре), и (или) избыточная точность измерения, не нужная в конкретном приложении, и за счет этого повышается коэффициент сжатия алгоритмом без потерь.

2. Выбор метода сжатия

Сенсорные данные относятся к источникам типа «аналоговый сигнал» [12], для которых естественным выбором является метод сжатия LPC — линейное прогнозирующее кодирование. Этот метод встречается с вариациями во многих алгоритмах, предназначенных для сжатия данных, характеризующихся наличием корреляции между последовательными значениями [3, 6, 7, 13–16]. Сенсорные данные тоже относятся к этой категории, если частота дискретизации значительно выше верхней частоты в спектре сигнала. Эта схема сжатия потоковая — может работать без накопления блока данных перед сжатием и может быть реализована с минимальными затратами оперативной памяти и ресурсов процессора.

[3] repeatability
[4] lossy compression

Блок прогнозирования строит прогноз очередного значения E_{v_i} исходя из k предыдущих значений: [5]

$$E_{v_i} = E(v_{i-k}, \dots v_{i-1}).$$

Вычисляется ошибка прогнозирования[6]:

$$r_i = v_i - E_{v_i},$$

и исходная последовательность $V = [v_0 \dots v_n]$ заменяется последовательностью $V' = [v_0, r_1, \dots r_n]$. При удачном прогнозировании число значащих разрядов в r_i в среднем значительно меньше, чем в v_i. Закодировав последовательность V' кодом с переменной длиной кодовых слов, можно получить эффект сжатия. Для максимального эффекта код должен быть выбран так, чтобы чаще встречающиеся значения r_i представлялись более короткими кодовыми словами.

Распределение r чаще всего [7, 15, 16] моделируют распределением Лапласа:

$$p(r) = \mathcal{L}(0, b) = \frac{1}{2b} \mathrm{e}^{-\frac{|r|}{b}},$$

где $2b^2$ — дисперсия. Для распределения Лапласа, оно же двойное экспоненциальное распределение, близкое к оптимальному кодирование достигается [18] использованием префиксных кодов Голомба-Райса [19, 20], и эти коды часто используются в алгоритмах сжатия аудиоданных. Префиксные коды дают компактное представление для малых значений r, и в то же время, не требуя памяти для хранения таблицы кодирования, позволяют закодировать любые возможные значения r:

$$r \in [-2^m \dots 2^m], \tag{1}$$

где $m = \max\limits_i \lceil log_2(v_i) \rceil$ — разрядность чисел в исходной последовательности V.

Встречается также применение более мощных методов статистического сжатия (арифметического кодирования) для улучшения эффективности кодирования r [5, 16]. Алфавит всех возможных значений r большой (1), поэтому для уменьшения объема памяти, необходимой для хранения таблицы кодирования, арифметическое кодирование применяют только для ограниченного интервала наиболее вероятных значений r. Значения, не входящие в интервал, в [16] кодируются кодами Голомба-Райса, а в [5] бинарным кодом фиксированной разрядности. Но и для выбора оптимального параметра

[5] Параметр $k \in 0, 1, ...$ называется порядком прогнозирования [6]. В частном случае $k = 1, E(v_i) = v_{i-1}$ схема сжатия называется дельта-кодированием, или дифференциальным кодированием [17]

[6] residual error

кода Голомба-Райса $\mathbf{GR}(k)$, и для арифметического кодирования требуется знание распределения входных данных, что требует памяти для буферизации некомпрессированных данных, а это несовместимо с поставленной задачей.

Решением могут быть адаптивные потоковые алгоритмы, в которых сжатие каждого значения r происходит немедленно после его появления, а параметры алгоритма кодирования подстраиваются после обработки каждого r. В кодере и декодере используются идентичные алгоритмы адаптации, что устраняет необходимость передачи словаря вместе с компрессированными данными. Примеры адаптивных алгоритмов: адаптивный код Хаффмана [21], адаптивное арифметическое кодирование [22], сжатие методом «стопки книг» [23], адаптивный код Голомба-Райса — `RLGR` [24]. Последний отличается минимальными требованиями к объему памяти, попробуем применить его к нашей задаче.

2.1. Выбор метода прогнозирования

Мы будем использовать линейное прогнозирование первого порядка $E(v_i) = v_{i-1}$, которое требует минимальных затрат памяти и ресурсов процессора. Есть ряд подходов для повышения точности прогнозирования: методами линейного прогнозирования [25], контекстного предсказания [9, 14], нейронных сетей [26, 27] — и в конкретных случаях, когда известен характер данных и ресурсы памяти и процессора не так сильно ограничены, можно к ним обратиться. В данной статье мы сосредоточимся на поиске способа кодирования последовательности V' с минимальными требованиями к ресурсам памяти и процессора, который обеспечил бы приемлемые результаты даже при не очень удачном прогнозировании.

3. Тестирование известных алгоритмов

3.1. Алгоритмы `z22`, `z1`

Результаты сжатия популярным архиватором `zstd` [28] с максимальным (`zstd -ultra -22`) и минимальным (`zstd -1`) уровнем сжатия включены в общую таблицу экспериментов для оценки энтропии тестовых источников данных. Данные на вход `zstd` подавались в текстовом виде, по одному отсчету на строку, после удаления даты и десятичной точки. Коэффициент сжатия вычислялся по выводу `zstd` при запуске с опцией `-v`. Результаты (столбцы `z22`, `z1` таблицы 4) в большинстве случаев лучше, чем у потоковых алгоритмов, поскольку `zstd` располагает значительным объемом памяти и может использовать

мощные алгоритмы сжатия, в частности LZ77[29] и ANS[30]. Алгоритм `zstd` сжимает набор данных целиком, без ограничения объема выходного буфера B_{out}.

3.2. Алгоритмы `F16`, `F256`

В этих двух тестах используется блочный алгоритм `FELACS` [4] с размером входного блока 16 и 256 байт соответственно. `FELACS` представляет собой алгоритм из аппаратных модулей сжатия NASA [31], адаптированный для программной реализации путем добавления эвристического алгоритма выбора параметра k кода Голомба-Райса $\mathbf{GR}(k)$.

Поскольку `FELACS` не потоковый алгоритм, он не может рассматриваться в качестве решения поставленной задачи; эти тесты интересны как оценка эффективности сжатия алгоритмом, ориентированным на сенсорные данные. Чтобы получить оценку коэффициента сжатия сверху, `FELACS` тестируется в наилучших для него условиях, без ограничения объема выходного буфера B_{out}.

Отметим, что в `FELACS` нет поддержки режима сжатия RLE, поэтому в тестах с большим числом повторов (строки №№ 44, 45, 53, 58, 65 таблицы 4) его результаты существенно ниже, чем у других алгоритмов.

3.3. Алгоритм `RLGR`

Алгоритм `RLGR` [24] сочетает кодирование повторов (RLE) и статистическое кодирование кодом Голомба-Райса $\mathbf{GR}(k)$. В отличие от `FELACS`, `RLGR` это потоковый алгоритм, полностью соответствующий нашей постановке задачи. Алгоритм адаптивный: параметр k кода Голомба-Райса изменяется после передачи каждого кодового слова синхронно в кодере и декодере, с целью уменьшить среднюю длину кодовых слов при кодировании конкретного потока V'. После кодирования нескольких $r = 0$ подряд алгоритм переключается в «режим с повторами», в котором перед каждым закодированным значением $r \neq 0$ передается счетчик предшествующих ему значений $r = 0$. Для кодирования счетчика повторов также используется код Голомба-Райса, параметр $k2$ которого адаптируется независимо от параметра k.

Результаты теста противоречивые: для некоторых источников коэффициент сжатия несколько выше других алгоритмов, но для ряда источников он ниже 1, т.е. вместо сжатия происходит увеличение объема данных. Прототип кодера ***rlgr.pl*** позволяет[7] рассмотреть

[7] команда: `./rlgr.pl -18 data/1/aem1-BWATT`

последовательность генерируемых кодером кодовых слов и объяснить эффект. На длинной последовательности малых r алгоритм успевает сильно уменьшить k. Если далее следует большое r, его кодирование кодом $\mathbf{GR}(k)$ с малым k оказывается очень неэффективным. Например, на данных источника № 3 на шаге $i = 44$ происходит кодирование числа 1530 кодом $\mathbf{GR}(2)$, в результате формируется «сверхдлинное» кодовое слово длиной 385 бит:

$$\mathbf{bitlength}(\mathbf{GR}(x,k)) = \lceil \frac{x}{2^k} \rceil + k$$
$$\mathbf{bitlength}(\mathbf{GR}(1530,2)) = 385.$$

Причин такого поведения три.

(*1*) Алгоритм работает на принципе постадаптации[8]. Не предусмотрено сигнализации о скачкообразном изменении k. Единственным сигналом от кодера к декодеру об изменении k является передача субоптимальных кодовых слов.

(*2*) Код Голомба-Райса *линейный* (при $k = \text{const}$ длина кодового слова пропорциональна кодируемому числу)

$$\mathbf{bitlength}(\mathbf{GR}(x,k)) \sim x, \quad x > k,$$

т.е. стоимость кодирования больших чисел при недостаточно большом k очень высока (см. рисунок 2).

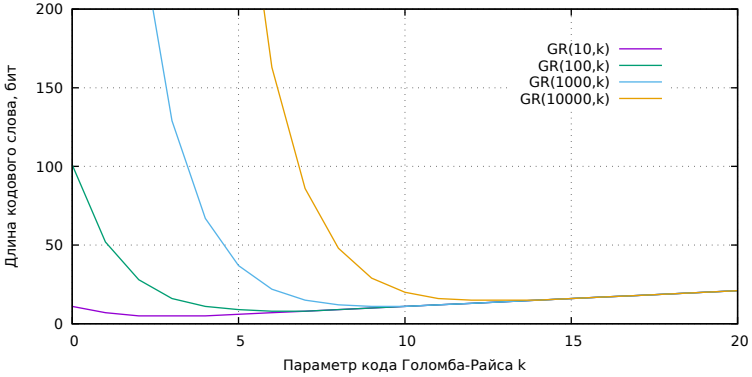


Рисунок 2. Эффективность кодирования кодом Голомба-Райса $\mathbf{GR}(k)$

Для *экспоненциальных* кодов, с увеличением длины кодового слова число представимых значений растет в геометрической прогрессии, так что кодирование больших чисел не приводит

[8] «backward adaptation rule»

к появлению сверхдлинных кодовых слов. Например, для кода Elias γ [**32**]

$$\mathbf{bitlength}(\gamma(x)) \sim \log_2(x),$$
$$\mathbf{bitlength}(\gamma(1530)) = 21.$$

(*3*) Правило адаптации «вверх» имеет вид $k_{i+1} = k_i + (p+1)/4$, где $p > 1$ — длина унарного префикса в только что сгенерированном субоптимальном кодовом слове $\mathbf{GR}(k)$. В рассматриваемом примере это правило приводит к $k = 97$ (!) на шаге $i = 45$. Далее алгоритм адаптации начинает снижать k с довольно низкой скоростью 0.5/шаг, так что оптимальные значения $k = 8$ восстанавливаются только к шагу $i = 222$. Средний размер кодового слова на шагах $i \in [44, 221]$ равен 56, что больше чем разрядность кодируемых чисел (1). Следовательно, на данном участке коэффициент сжатия будет низкий: $K = S_0/S_c = (18+1)/56 = 0.34$.

Таким образом, на реальных сенсорных данных алгоритм `RLGR` иногда попадает в «трудные ситуации», и для использования в нашей задаче нуждается в доработке. Отложим его пока и попробуем использовать экспоненциальные коды. Но мы еще вернемся к `RLGR` в разделе 4.11. Данный предварительный тест проводился без ограничения размера выходного буфера B_{out}.

4. Реализация и тестирование экспериментальных алгоритмов

4.1. Алгоритмы `EG` и `EW`

В алгоритме `EG` значения r кодируются кодом Elias γ. После значения $r = 0$, соответствующего точному прогнозу, передается счетчик повторений — также в коде Elias γ. Таким образом в схему кодирования добавлен режим RLE. Алгоритм `EW` полностью аналогичен алгоритму `EG`, но используется код Elias ω.

Результаты `EG` и `EW` примерно одинаковые, ни один не имеет уверенного преимущества. Результаты значительно ровнее, чем в тесте `RLGR`, но для некоторых источников коэффициент сжатия низкий. Как мы увидим ниже, у этих источников распределение r имеет «тяжелые хвосты» — значительную долю $r : |r| >= 128$.

4.2. Экспоненциальный бинарный код (expbinary)

При описании следующих алгоритмов нам понадобится код, который мы будем называть expbinary за неимением лучшего имени. Это код переменной длины, не обладающий свойством префиксного

кода[9], поэтому он не имеет самостоятельного применения и редко упоминается в литературе. В [18] он упоминается с именем $\overline{B}(n)$. Его можно увидеть, например, в суффиксной части кодовых слов префиксных экспоненциальных кодов переменной длины Elias γ и Elias ω. Коды expbinary, Elias γ и Elias ω показаны на примерах в приложении 1.

В коде binary n-битное слово представляет числа в диапазоне

$$[0 \dots 2^n - 1], n \in 1, 2, \dots. \tag{2}$$

В коде expbinary n-битное слово представляет числа в диапазоне

$$[2^n \dots 2^{n+1} - 1], n \in 0, 1, 2, \dots. \tag{3}$$

Таким образом,

- длина кодовых слов в expbinary на один бит меньше, чем в binary: $\mathbf{bitlength}(C_{expbinary}(N)) = \mathbf{bitlength}(C_{binary}(N)) - 1$;
- код expbinary не позволяет закодировать $N = 0$;
- используется пустое (0-битное) кодовое слово: $C_{expbinary}(1)$; это возможно, поскольку кодовые слова expbinary всегда предваряются каким-либо префиксом.

Для декодирования кодового слова $C = C_{expbinary}(N)$ необходимо по префиксу определить длину кодового слова $n = \mathbf{bitlength}(C)$, вычислить начало диапазона значений кодовых слов expbinary длиной n: $S = 2^n$, и вычислить N

$$N = \begin{cases} S + C, & n > 0, \\ 0, & n = 0, \end{cases}$$

где C интерпретируется как n-битное число в бинарном коде.

4.3. Алгоритм EB

Тест предназначен для получения оценки сверху коэффициента сжатия при кодировании V' кодом переменной длины на основе кода expbinary (в частности, кодами Elias γ, Elias ω). Алгоритм состоит в записи в выходной поток значений r_i в коде expbinary без префиксов, по которым декодер мог бы разделить битовый поток на кодовые слова, т.е. реализовать декодер невозможно. Результаты в таблице 4 выделены серым цветом, чтобы показать, что они «вне конкурса». Они примерно в два раза выше, чем результаты тестов `EG`, `EW`. Поэтому можно улучшить результаты `EG`, `EW`, если удастся найти более компактный способ кодирования префиксов, чем способы, используемые в кодах Elias γ и Elias ω.

[9] $\nexists A, B : Ax = B$ (не существует кодового слова B, начало которого совпадает с каким-либо другим кодовым словом A)

4.4. Распределение дельта-кодированных данных

Распределение ошибок $p(r)$, которое обычно моделируют распределением Лапласа, не всегда хорошо соответствует этой модели. В лучшем случае ($\forall i : r_i = 0$) распределение имеет вид $p(0) = 1$ (рисунок 3, **a**). Этот лучший случай возникает при идеальной работе предиктора; для простейшего дельта-предиктора это случается только при $\forall i : v_i = \text{const}$. В худшем случае ($v_i$ — случайные данные с равномерным распределением) распределение $p(r)$ имеет вид равнобедренного треугольника (рисунок 3, **b**).

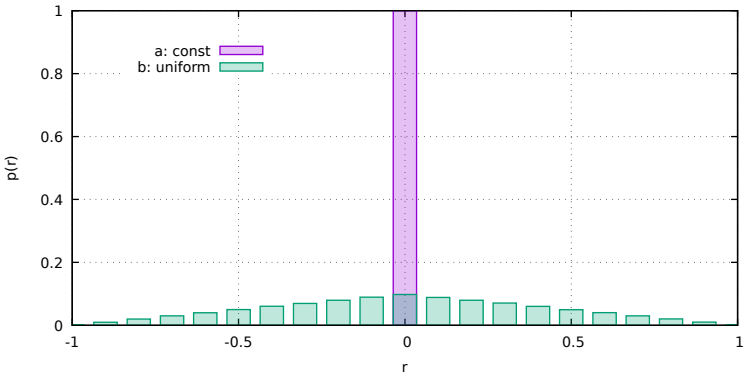


Рисунок 3. Распределение ошибок прогнозирования **a)** для постоянных данных $v_i = \text{const}$, **b)** для равномерно распределенных случайных данных $v_i = \mathbf{uniform}(0, 1)$

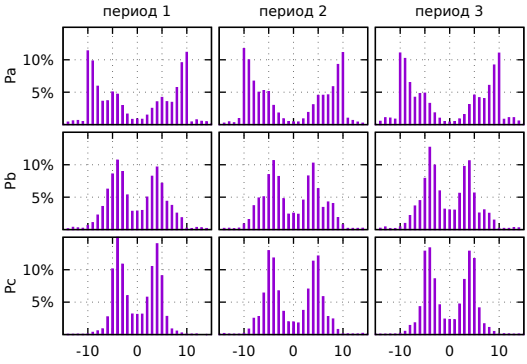
Рассмотрим распределения $p(r)$ для наборов данных, используемых в тестах. В контексте выбора метода кодирования удобнее рассматривать не функцию плотности вероятности $p(r)$, а дискретное распределение вспомогательной функции $p(W(r))$, где

$$W(r) = \begin{cases} \mathbf{bitlength}(C_{expbinary}(r)) + 1, & r > 0 \\ 0, & r = 0 \\ -\mathbf{bitlength}(C_{expbinary}(-r)) - 1, & r < 0 \end{cases}$$

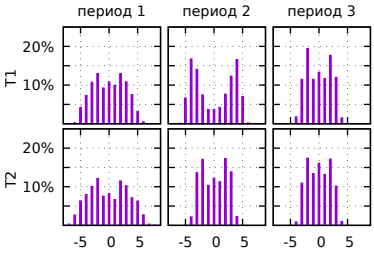
а $w = |W(r)| - 1$ это число двоичных разрядов, необходимых для представления r в коде expbinary. Для отрицательных значений r используются отрицательные значения W, чтобы было видно асимметричность распределения. Значение $W(0) = 0$ появляется при $v_i = v_{i-1}$ и указывает на возможность кодирования повторов (RLE).

Сравнивая распределения данных датчиков электроэнергии `Pa`, `Pb`, `Pc`, температуры `T1`, `T2` и точки росы воздуха `Tdp1`, `Tdp2` (рисунок 4),

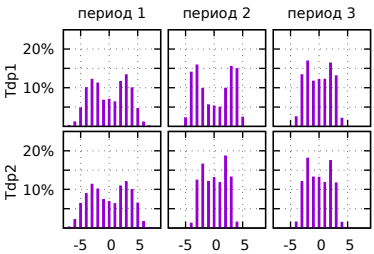
можно заметить следующее:



(*а*) датчики электроэнергии



(*б*) датчики температуры воздуха



(*в*) датчики точки росы воздуха

Рисунок 4. Распределение функции $W(r)$ для различных источников

- дельта-кодирование справляется с задачей понижения разрядности для всех рассматриваемых наборов данных, хотя точность предсказаний далека от идеальной, при которой распределение имело бы единственный пик $p(W(0)) = 1$;
- у источника `Pa`, на котором в тестах `EG`, `EW` самие плохие результаты, значительная (до 40%) доля больших значений, требующих 8–9 бит в коде expbinary—следствие не очень удачного прогнозирования. В коде Elias γ такие значения получают унарный префикс длиной 9–10 бит, общая длина кодового слова оказывается 18–20 бит, отсюда низкий коэффициент сжатия;
- диапазон значений $W(r)$ и моды распределения меняются как от источника к источнику, так и от периода к периоду. Поэтому никакой статический код не обеспечит оптимального кодирования; требуется адаптивный алгоритм;

- есть небольшая асимметричность распределения для положительных и отрицательных значений r, которая отражает неодинаковость скоростей нарастания и убывания в исходной последовательности V.

4.5. Алгоритм e1: «стопка книг»

Алфавит префиксов, необходимых для поддержки кодирования V' к коде expbinary, небольшой: в худшем случае $[-m, m]$ (1), а фактически (судя по рисунку 4*б*) еще меньше. Это позволяет с малыми затратами реализовать адаптивное назначение префиксов по методу «стопки книг»[10] [**23**].

В алгоритме `e1` в качестве префиксов используются значения функции $W(r)$, т.е. префикс определяет знак и разрядность следующего за ним числа в коде expbinary. Алгоритм оперирует структурой данных, аналогичной стопке книг; названия «книг» соответствуют значениям функции $W(r)$. В начальный момент стопка пуста: высота стопки $h = 0$. Для кодирования очередного значения r_i:

- вычисляется $w = W(r_i)$ и ищется в «стопке» линейным поиском сверху вниз: $j \in [0 \dots h-1]$;
  - если книга с названием w найдена в стопке с порядковым номером сверху j, передается кодовое слово префикса $C_p(j)$, а книга перекладывается на верх стопки;
  - если книга с названием w не найдена в стопке, передается два кодовых слова: $C_p(h), C_n(w)$, и новая книга добавляется на верх стопки.
- если $r_i \neq 0$, передается кодовое слово $C_{expbinary}(r_i)$ (его длина $|w|$ бит).
- если $\{r_i, \dots r_{i+R-1}\} = \{0 \dots 0\}$, передается кодовое слово $C_r(R)$, где $R \in 0, 1, \dots$ — счетчик повторений нулевого значения r.

Здесь $C_p(j)$ — код переменной длины, используемый для кодирования префиксов, например унарный код или код Голомба. В тестах лучшие результаты получаются с использованием кода vbinary2x1x [**34**], в котором первые три кодовых слова имеют одинаковую длину, 2 бита (см. приложение 1).

$C_r(x)$ — код переменной длины, используемый для кодирования счетчиков повторений RLE. В тестах лучшие результаты получаются с использованием кода Elias ω.

[10] также известен как Move-to-front transform [**33**]

По результатам тестирования алгоритм `e1` действительно дает улучшение для данных, на которых алгоритмы `EW` и `EG` показали слабые результаты, особенно № 59. Но на ряде тестов он все-таки проигрывает алгоритмам `F256` и `RLGR`.

Возможная причина проигрыша — используемый в алгоритме `e1` вынос информации о знаке r_i в префикс в надежде на использование асимметричности распределения. Благодаря выносу знака в коде expbinary кодируется $|r_i|$ вместо r_i, что уменьшает длину всех кодовых слов expbinary на один бит. Это могло бы дать выигрыш на монотонных участках последовательности V'. Но число различных префиксов (высота стопки h) при этом увеличивается вдвое, и для получения выигрыша требуется высокая локальность (частое использование префиксов, находящихся наверху стопки), иначе потери от увеличения длин кодовых слов префиксов будут больше, чем ожидаемый выигрыш.

4.6. Алгоритм e2: знак в бинарной части

Алгоритм `e2` отличается от `e1` только переносом знака r_i из префикса в код expbinary; теперь префикс вычисляется как $w = |W(r_i)|$, число префиксов (высота стопки h) уменьшилось вдвое. В тестах без ограничения размера выходного буфера B_{out} (не приведены в таблице 4) алгоритм `e2` превосходит `e1` на всех источниках, но при $B_{out} = 256$ `e2` часто уступает `e1`.

4.7. Алгоритм e3: ограничение высоты стопки

Недостатком наивной реализации алгоритма «стопки книг» является относительно высокая стоимость операции переноса книги на верх стопки, которая выполняется при обработке каждого r_i. При числе различных префиксов порядка 10–20 стопка реализуется в виде массива, и перенос книги наверх из позиции j требует сдвига (последовательного копирования) элементов массива от 0 до j, чтобы освободить место на верхушке стопки.

Если ограничить высоту стопки небольшим значением, например $h_{\max} = 5$, то стопку можно реализовать в виде 32-битного целого числа, и на 32-битных процессорах выполнять операции со стопкой очень эффективно в духе SWAR-алгоритмов [**35**]. Сдвиг стопки вместо цикла с обращениями к памяти реализуется командой побитового сдвига. Естественно, ограничение высоты приведет к увеличению числа случаев, когда нужная книга отсутствует в стопке, которые приходится

обрабатывать как «новый префикс». Чтобы уменьшить длину кодового слова для случая «новый префикс», будем использовать для него индекс $j = 0$, а не $j = h$, как в первоначальном алгоритме «стопки книг».

Заметим также, что найденная в глубине высокой стопки книга может быть не лучше, а даже хуже чем книга с полки, поскольку для больших индексов j длина кодового слова $C_p(j)$ может оказаться больше чем длина последовательности «новый префикс»: $C_p(0), C_n(w)$.

Результаты тестирования `e3` по сравнению с `e2` приведены в таблице 4: на 31 источнике из 75 незначительное снижение коэффициента сжатия, на 44 источниках без изменений.

4.8. Алгоритм `e4`: замедление адаптации

Еще одна модификация алгоритма — снизить скорость адаптации: вместо того, чтобы перемещать использованную книгу сразу на верх стопки (предоставляя ей кратчайший префикс, отбирая его у других), ограничиться перемещением использованной книги вверх на одну позицию, подобно алгоритму пузырьковой сортировки.

Результаты тестирования `e4` по сравнению с `e3` приведены в таблице 4: улучшение на 59 источниках из 75, ухудшение на 16 источниках. Результат тестирования по сравнению с `e1`: улучшение на 50 источниках, ухудшение на 21 источнике, на 4 источниках без изменений.

4.9. Алгоритм `e5`: несколько стопок

Ограничение высоты стопки привело к увеличению числа случаев, когда нужная книга отсутствует в стопке, которые приходится обрабатывать как «новый префикс», см. раздел 4.7. Можно попробовать снова уменьшить его, используя не одну, а несколько низких стопок. Организуем кодер как автомат Мура, число состояний в котором равно числу используемых префиксов N_w. После обработки r_i кодер переходит в состояние $w = w_i$ и находится в нем во время обработки r_{i+1}. С каждым состоянием ассоциирована своя стопка; кодер использует стопку текущего для преобразования $w_{i+1} \to j_{i+1}$ и переходит в состояние w_{i+1}. Если входные данные таковы, что не все переходы между состояниями равновероятны, есть надежда что стопка каждого состояния будет содержать нужные именно в этом состоянии книги.

Алгоритм `e5` отличается от предшественников увеличенным расходом памяти для хранения нескольких стопок:

$$M \sim \lceil \log_2(N_w) \rceil h_{\max}$$

Например, если число состояний автомата $N_w = 16$, а каждая стопка представлена 32-битным числом ($h_{\max} \le 8$), потребуется $M = 64$ байта памяти. Возможны компромиссные реализации с числом стопок $N_s < N_w$, в которых стопка выбирается хэшированием номера состояния w.

В тестах без ограничения длины выходного буфера, не включенных в таблицу 4, алгоритм `e5` по сравнению с `e4` дает улучшение на 26 источниках, незначительное ухудшение на 9 источниках (№ 26, 28, 36, 38, 42, 45, 50, 69, 75), и на одном источнике без изменений. В тестах с ограничением размера выходного буфера $B_{out} = 256$ `e5` по сравнению с `e4` дает улучшение на 16 источниках, ухудшение на 58, и на одном источнике без изменений.

В целом эффект от увеличения числа стопок оказался существенно ниже ожидаемого, и даже негативным в нужном нам режиме $B_{out} = 256$.

4.10. Алгоритм DD: «dynamic delta»

Попробуем другой подход, с использованием идеи из [3]: вместо префиксного кода переменной длины кодировать значения r_i бинарным кодом фиксированной длины w, которую можно время от времени менять, сигнализируя об изменении передачей специального кода. В [3] алгоритм не описан подробно; мы реализуем идею в адаптивном стиле, без накопления блока некомпрессированных данных для выбора w.

Кодер на рисунке 5 преобразует последовательность r_i в последовательность кодовых слов.

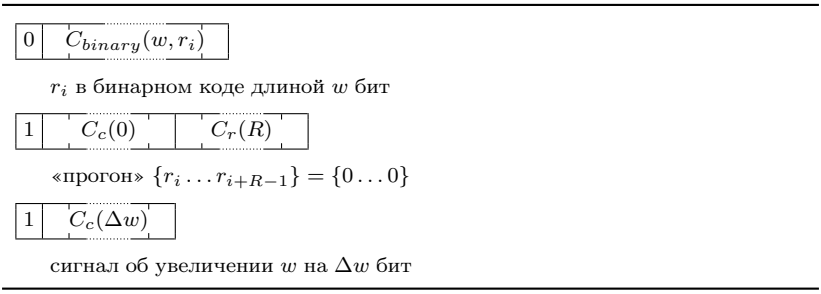


Рисунок 5. Кодовые слова алгоритма `DD`. C_c—код управления (используется vbinary2x), C_r—код длины прогона R (используется Elias ω)

Сигнала об уменьшении w не предусмотрено — уменьшение происходит автоматически:

$$w_{i+1} = \begin{cases} w_i - d & w_i > 1 \\ w_i & w_i = 1, \end{cases}$$

где d и w_0 — параметры алгоритма. В тестах лучшие результаты получены при $d = 0.5$, $w_0 = 7$.

Алгоритм `DD` уступает алгоритмам `e1 ...e5` в эффективности кодирования бинарной части кодовых слов, поскольку при кодировании малых r в бинарном коде фиксированной длины бесполезно передаются незначащие нули. Зато в эффективности кодирования префиксов возможен выигрыш, когда значительная часть данных передается с минимальным однобитовым префиксом. В среднем результаты `DD` немного хуже, чем у предшественников, но они ровные и неожиданно неплохие для такого простого алгоритма.

4.11. Алгоритм RLGR′: RLGR с $B_{out} = 256$

Вернемся еще раз к алгоритму `RLGR` из раздела 3.3 и протестируем его с ограничением выходного буфера до $B_{out} = 256$ байтов. Потребуется небольшая корректировка параметров: по умолчанию `RLGR` стартует с $k = 2$, и на 10 источниках (№№ 2, 3, 4, 18, 19, 20, 30, 32, 34, 43 таблицы 4) завершается аварийно — не может обработать ситуацию «в начале блока большое число r» из-за того что $\lceil \mathbf{bitlength}(\mathbf{GR}(r,k))/8 \rceil > B_{out}$. Если стартовать с $k = 10$, тест `RLGR′` проходит, и (сюрприз!) в результатах нет тех «провалов», которые наблюдались в тесте `RLGR`. Почему?

Алгоритм по-прежнему попадает в «трудные ситуации», но их последствия ограничены свободным объемом в текущем выходном блоке. В «трудной ситуации» текущий блок быстро наполняется, происходит переключение на новый блок с реинициализацией алгоритма и, тем самым, автоматический выход из «трудной ситуации», пусть и со снижением коэффициента сжатия для данного выходного блока. Это стихийное решение трудно назвать красивым, и оно становится менее эффективным с увеличением B_{out}. Тем не менее, тест прошел и показал улучшение на 42 источниках по сравнению с `e4` (таблица 4). Это значит, что у `RLGR` можно поучиться.

(*1*) Кодирование адаптивным кодом Голомба-Райса оказывается эффективнее, чем кодирование экспоненциальными кодами

(Elias γ, Elias ω), у которых в префиксе каждого кодового слова содержится полная информация о длине суффикса. Динамическое кодирование префикса с помощью MTF-преобразования в алгоритмах `e1`...`e5` сродни адаптивному изменению параметра k в адаптивном коде Голомба-Райса, и дает примерно такой же эффект.

(*2*) Автоматическое переключение режимов «с прогонами» — «без прогонов» несколько улучшает сжатие данных с малым числом прогонов, которые как раз сжимаются хуже всех (строки №№ 1, 2, 15, 16, 29, 30 таблицы 4). В то же время, на данных с одиночными значениями посреди прогонов (№№ 36, 38, 40, 42) результаты `RLGR'` тоже на 14–36% лучше, чем у ближайших конкурентов.

(*3*) Кодирование длин прогонов в режиме «с прогонами» в `RLGR` описано алгоритмически [**24**]; по сути длина тоже кодируется в коде Голомба-Райса GR($R, k2$), но с интересной особенностью: процедура адаптации $k2$ вызывается не после генерации всего кодового слова GR($R, k2$), а после генерации каждого бита унарного префикса. Таким образом, при кодировании большой длины прогона R с малым $k2$ изменение $k2$ происходит прямо в ходе генерации кодового слова. Это улучшает эффективность кодирования больших R при малых $k2$ ценой лишних затрат процессорного времени, но не помогает при кодировании малых R при больших $k2$. Т.е. при переходе датчика от долгого «молчания» к активности произойдет сравнительно эффективное кодирование большого R с увеличением $k2$, но затем алгоритм будет долго бесполезно оставаться в режиме «с прогонами», пока $k2$ не уменьшится до 0 медленным адаптивным алгоритмом.

(*4*) Принцип «постадаптации» хорошо работает для тонкой подстройки параметра k, но не может противодействовать появлению сверхдлинных кодовых слов при появлении большого r при малом k. Не хватает механизма экстренного увеличения k для эффективного кодирования неожиданных больших значений r. После введения такого механизма можно отказаться от сомнительного правила быстрой адаптации «вверх», которое создает «трудные ситуации» в алгоритме `RLGR`.

4.12. Алгоритм `e10`

Алгоритм `e10` является финальным в серии экспериментальных алгоритмов по мотивам `RLGR`. Подобно `RLGR`, алгоритм `e10` имеет режимы «без прогонов» и «с прогонами», и переходит из режима в режим

в порядке адаптации к компрессируемым данным. Номенклатура кодовых слов алгоритма `e10` приведена на рисунке 6.

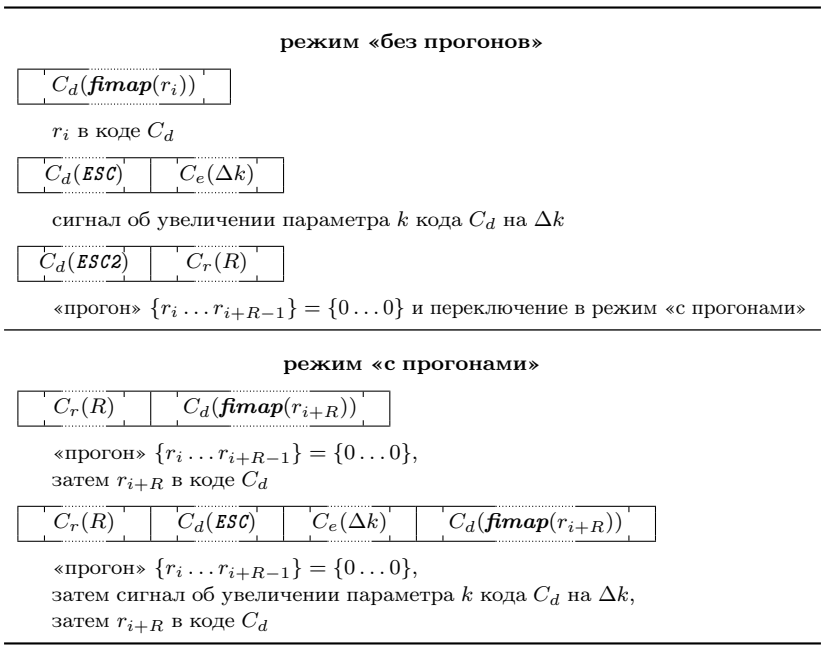


Рисунок 6. Кодовые слова алгоритма `e10`

Для кодирования значений r используется код Голомба-Райса $C_d(r) = \mathbf{GR}(\mathbf{fimap}(r), k)$. Параметр k медленно адаптируется, синхронно в кодере и в декодере. Функция **fimap** отображает множество r (целые числа) на область определения функции GR (неотрицательные целые числа), и резервирует два `ESC`-значения для сигнализации. Сигнализация используется нечасто, поэтому для `ESC`-значений выбраны не самые короткие кодовые слова.

Если r *слишком велико* для оптимального кодирования при текущем k, кодер сначала передает сигнальное кодовое слово (`ESC`) о скачкообразном увеличении k до $k_{opt} = \lceil \log_2(\mathbf{fimap}(r)/2) \rceil$, и только после этого $C_d(r) = GR(\mathbf{fimap}(r), k_{opt})$. Алгоритм продолжает работать дальше с $k = k_{opt}$. Критерий *«слишком велико»* вычисляется только в кодере, и может варьироваться из соображений удобства реализации без изменения декодера.

Если кодер встречает «прогон» $\{r_i \ldots r_{i+R-1}\} = \{0 \ldots 0\}$, он может поступить одним из двух способов:

(1) передать прогон буквально $\{C_d(r_i) \ldots C_d(r_{i+R-1})\}$. Этот способ применяется при небольшой длине прогона R;

(2) передать сигнал ESC2, за ним длину прогона $C_r(R)$, где C_r — код, подходящий для кодирования произвольных значений R. После этого алгоритм переходит в режим «с прогонами».

В режиме «с прогонами» перед каждым $C_d(r, k)$ передается длина прогона в коде C_r (см. раздел 4.12.2). Выход из режима «с прогонами» происходит (синхронно в кодере и декодере) по тайм-ауту: после $T = 4$ шагов с пустыми прогонами ($R = 0$).

В качестве кода C_e для кодирования Δk используем vbinary2x(1,2,3x), см. примеры в приложении 1.

4.12.1. *Адаптация в алгоритме e10*

Условие оптимальности кода Голомба-Райса [**36**] можно записать в виде $\lfloor 2^{k-1} \rfloor \leqslant r < 2^{k+1} + 2^k$. При соблюдении такого соотношения r и k получаются кратчайшие кодовые слова $\mathbf{GR}(r, k)$. Для кода C_d алгоритм адаптации изменяет k следующим образом:[11]

$$k_{i+1} = \begin{cases} k_i, & \lfloor 2^k \rfloor \leqslant r < 2^{k+1} + 2^k \\ k_i + 0.75, & r \geqslant 2^{k+1} + 2^k \\ k_i - 0.25, & r < \lfloor 2^k \rfloor. \end{cases}$$

Выбрана более высокая скорость адаптации «вверх», чем «вниз», поскольку, как мы видели на рисунке 2, при $k > k_{opt}$ эффективность кодов Голомба-Райса снижается не так быстро, как при $k < k_{opt}$.

4.12.2. *Выбор кода для длин прогонов RLE в алгоритме e10*

Для уменьшения накладных расходов на пребывание в режиме «с прогонами» при отсутствии прогонов желательно выбрать такой C_r, чтобы $C_r(R = 0) = \min = 1$.

Также от кода C_r требуется эффективное кодирование R в широком диапазоне: $[0, {\sim}10^6]$, поэтому коды Голомба-Райса не годятся. В качестве кандидатур рассматривались экспоненциальные коды Elias γ и Elias ω. Выбран Elias γ за лучшую эффективность в области небольших R: $R \in \{4, \ldots, 7\} \cup \{16, \ldots, 63\}$. Код Elias ω становится эффективнее

[11] получаются дробные значения k, которые при использовании округляются вниз: $\mathbf{GR}(r, \lfloor k \rfloor)$

кода qElias γ при $R \geqslant 128$ (рисунок 7), но при таких больших R и код Elias γ обеспечивает коэффициент компрессии в режиме RLE

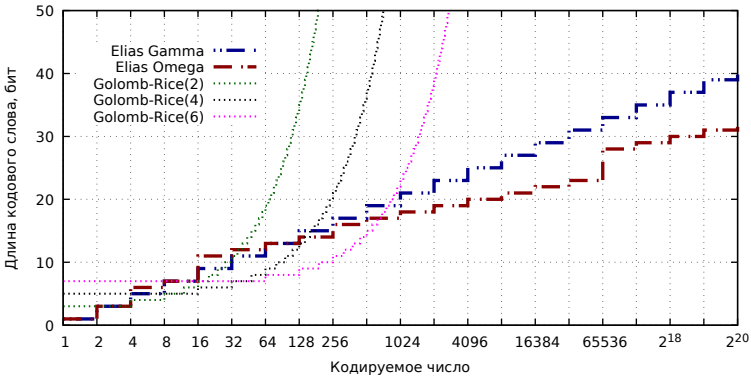


Рисунок 7. Сравнительная эффективность кандидатов на роль кода C_r для кодирования длины прогонов в алгоритме `e10`

на порядки больше, чем в принципе возможно в режиме статистического кодирования одиночных значений r.

4.12.3. *Заключительные замечания по алгоритму e10*

Благодаря механизму экстренной адаптации «вверх» (`ESC` Δk) алгоритм `e10` защищен от «трудных ситуаций» алгоритма `RLGR`. Механизма экстренной адаптации «вниз» не предусмотрено, потому что кодирование малых r при больших k не приводит к появлению сверхдлинных кодовых слов: при $r < 2^k$ $\mathbf{bitlength}(\mathbf{GR}(r,k)) = k+1$.

Использование экспоненциального кода для кодирования длин прогонов R и фиксированное время пребывания в режиме «с прогонами» при отсутствии прогонов обеспечивают эффективное кодирование в случае редких больших прогонов. При кодировании коротких прогонов возможно снижение эффективности по сравнению с `RLGR`, хотя на имеющихся тестовых данных этот эффект не проявился. Коэффициент сжатия прогонов в режиме «с прогонами» будет не хуже

$$K = mR/(G + \mathbf{bitlength}(\gamma(R+1))).$$

где $G < T$ — число шагов с $R = 0$ между шагами с $R > 0$. В худшем случае (прогоны длиной $R = 1$ при малой разрядности входных данных $m = 8$ с интервалом $G = T - 1 = 3$ шага), $K = 8/(3+3) = 1.25$.

По результатам тестирования алгоритм `e10` превосходит алгоритм `RLGR′` на 69 источниках из 75 (таблица 4), уступает на 5 источниках, и 1 без изменений. Алгоритм `e10` превосходит алгоритм `e4` на 64 источниках, уступает на 10 источниках, и 1 без изменений.

4.13. Критерий сравнения алгоритмов

Сравнение алгоритмов непосредственно по данным таблицы 4 затруднительно, поскольку результаты тестирования каждого алгоритма представляют собой вектор из 75 значений. Почленное сравнение векторов работает только в редких случаях, когда алгоритм A превосходит алгоритм B на данных от всех источников. Во всех остальных случаях для сравнения нужна скалярная характеристика, вычисленная исходя из данных вектора.

Стандартные скалярные характеристики распределения коэффициентов сжатия для каждого алгоритма на множестве тестовых источников приведены в таблице 5: минимум $\min K$, максимум $\max K$, среднее арифметическое $\operatorname{avg} K$ и медиана $\operatorname{med} K$. Средний коэффициент сжатия полезен для оценки времени автономной работы сенсорного узла в режиме накопления данных при известном количестве и характере источников данных — но как видно из таблицы 5, среднее арифметическое коэффициентов сжатия не подходит для этой роли из-за сильного влияния больших максимальных значений. Медиана не подвержена влиянию больших максимальных значений, но может существенно отличаться от искомого среднего в случае несимметричного распределения.

Искомый средний коэффициент сжатия для смеси источников можно определить как

$$\overline{K} = \frac{\sum_{i=1}^{n} S_{0i}}{\sum_{i=1}^{n} S_{ci}},$$

где S_{0i} — объем несжатых данных, поступивших от источника i, S_{ci} — объем данных от источника i после сжатия, n — число источников в смеси. Вектор $(S_{01}, \ldots, S_{0n})$ характеризует сравнительную интенсивность источников (объем порции данных и частоту поступления порций).

По определению коэффициента сжатия,

$$S_{ci} = S_{0i}/K_{ij},$$

где K_{ij} — коэффициент сжатия данных источника i алгоритмом j

(данные таблицы 4). Следовательно,

$$\overline{K_j} = \frac{\sum_{i=1}^{n} S_{0i}}{\sum_{i=1}^{n} S_{0i}/K_{ij}}.$$

Данные о сравнительной интенсивности появляются только в конкретных приложениях; для сравнения алгоритмов в общем случае будем предполагать одинаковую интенсивность (равный объем данных от каждого источника)

$$S_{01} = S_{02} = \cdots = S_{0n}.$$

Тогда

$$\overline{K_j} = \frac{\sum_{i=1}^{n} S_{01}}{\sum_{i=1}^{n} S_{01}/K_{ij}} = \frac{\sum_{i=1}^{n} 1}{\sum_{i=1}^{n} 1/K_{ij}},$$

что совпадает с известной формулой среднего гармонического

$$H(x_1, \ldots, x_n) = \frac{n}{1/x_1 + \cdots + 1/x_n}.$$

Значения среднего гармонического для смеси данных, включающей все 75 тестовых источников, приведены в колонке $H(K)$ таблицы 5, и используются в качестве основного критерия сравнения алгоритмов.

Таблица 5. Обобщающие характеристики алгоритмов сжатия. Строки отсортированы в порядке убывания среднего гармонического коэффициентов сжатия $H(K)$

Цвет фона: вне конкурса | лучшее в столбце | худшее в столбце

Алгоритм	min K	max K	avg K	med K	$H(K)$
z22	3.61	10000	218.1	6.85	6.98
z1	3.04	10000	201.1	4.96	5.15
e10	1.53	8.8e5	11817	4.19	3.79
e4	1.48	1.0e6	13748	3.95	3.54
RLGR′	1.52	6.0e5	8036	3.85	3.48
e5	1.40	1.0e6	13747	3.89	3.48
e1	1.43	9.9e5	13263	3.95	3.46
DD	1.46	1.1e6	14256	3.82	3.45
e2	1.50	9.9e5	13268	3.68	3.43
e3	1.50	9.9e5	13267	3.68	3.42
EG	0.97	9.4e5	12593	3.19	2.92
F256	1.08	16.70	4.100	2.91	2.92
EW	1.10	1.1e6	14807	2.91	2.82
F16	1.06	8.00	3.55	2.64	2.64
RLGR	0.001	7.2e5	9646	2.25	0.04

5. Заключение

Мы провели эмпирическое исследование нескольких известных и вновь предложенных экспериментальных алгоритмов сжатия сенсорных данных, использующих минимальный объем памяти и работающих в потоковом режиме без предварительного накопления блока некомпрессированных данных. Все алгоритмы основаны на классической схеме сжатия с прогнозированием (дельта-кодирование); статья посвящена выбору способа кодирования ошибок прогнозирования. Рассмотрено несколько подходов, в частности коды переменной длины с динамическими префиксами, полученными с помощью MTF-преобразования, кодирование адаптивным бинарным кодом, кодирование адаптивным кодом Голомба-Райса. На 75 тестовых наборах данных достигнуты коэффициенты сжатия порядка 1.5/4/1000000 (мин./медиана/макс.) при расходе памяти 256 байт на выходной буфер компрессированных данных и порядка 10 байт на данные состояния алгоритма сжатия. Большие максимальные коэффициенты сжатия достигаются на постоянных данных в режиме RLE благодаря потоковому режиму сжатия.

Было замечено, что коэффициент сжатия медленно меняющих сенсорных данных может быть сильно снижен из-за присутствия в данных шумов с небольшой амплитудой, препятствующих использованию режима RLE. Для таких источников гистерезисный шумоподавитель с порогом, выбранным исходя из характеристик датчика так, чтобы не ухудшить качество данных, позволил увеличить коэффициент сжатия в 11–20 раз. Применение шумоподавления к быстро меняющимся данным не улучшает сжатия.

По результатам сравнения коэффициентов сжатия на множестве тестовых источников лучшими кандидатами для практического применения представляются экспериментальные алгоритмы `e10` (лучшее сжатие) и `e4` (проще в реализации). Алгоритмы предназначены в первую очередь для сжатия данных в сенсорных узлах, но могут пригодиться и в других случаях, когда нужен потоковый алгоритм сжатия с малым объемом данных состояния — например, на сервере, принимающем данные от большого числа источников.

Использованные в статье данные, прототипы кодеров и тестовая оснастка свободно доступны (*https://github.com/yysizif/sencomp*).

Автор благодарит редакцию журнала «Программные системы: теория и приложения» и лично проф. С. В. Знаменского за ценные замечания, позволившие значительно улучшить раздел 4.13.

Список литературы

[1] Fall K. *A delay tolerant network architecture for challenged internets // SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (August 25–29, 2003, Karlsruhe, Germany), New York: ACM.– 2003.– ISBN 978-1-58113-735-4.– pp. 27–34. ↑

[2] Sadler C. M., Martonosi M. *Data compression algorithms for energy-constrained devices in delay tolerant networks // SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems* (31 October 2006–3 November 2006, Boulder, Colorado, USA), New York: ACM.– 2006.– ISBN 978-1-59593-343-0.– pp. 265–278. ↑

[3] Arrabi S., Lach J. *Adaptive lossless compression in wireless body sensor networks // BodyNets '09: Proceedings of the Fourth International Conference on Body Area Networks* (April 1–3, 2009, Los Angeles, CA, USA), Brussels: ICST.– 2010.– ISBN 978-963-9799-41-7. ↑

[4] Kolo J. G., Shanmugam S. A., Lim D. W. G., Ang L. M. *Fast and efficient lossless adaptive compression scheme for wireless sensor networks* // Computers & Electrical Engineering.– 2015.– Vol. **41**.– pp. 275–287. ↑

[5] Huang F., Liang Y. *Towards energy optimization in environmental wireless sensor networks for lossless and reliable data gathering*, IEEE International Conference on Mobile Adhoc and Sensor Systems (08–11 October 2007, Pisa, Italy).– 2007.– pp. 1–6. ↑

[6] Coalson J., Description of the FLAC format. ↑

[7] Robinson T. *SHORTEN: Simple lossless and near-lossless waveform compression*, Technical report CUED/F-INFENG/TR.156.– 1994.– 16 pp. ↑

[8] Pelkonen T., Cavallaro P., Huang Q., Franklin S., Meza J., Teller J., Veeraraghavan K. *Gorilla: a fast, scalable, in-memory time series database* // Proceedings of the VLDB Endowment.– 2015.– Vol. **8**.– No. 12.– pp. 1816–1827. ↑

[9] Lazin E. *Compression algorithms in Akumuli.* ↑

[10] Jazizadeh F., Afzalan M., Becerik-Gerber B., Soibelman L. *EMBED: A dataset for energy monitoring through building electricity disaggregation // e-Energy '18: Proceedings of the Ninth International Conference on Future Energy Systems* (June 12–15, 2018, Karlsruhe, Germany), New York: ACM.– 2018.– ISBN 978-1-4503-5767-8.– pp. 230–235. ↑

[11] *Digital Humidity Sensor SHTW2 (RH/T)*, Datasheet.– Switzerland: Sensirion AG.– 14 pp. ↑

[12] Ватолин Д., Ратушняк А., Смирнов М., Юкин В. *Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео.*– М.: ДИАЛОГ-МИФИ.– 2002.– ISBN 5-86404-170-X.– 384 с. ↑

[13] Ratanaworabhan P., Ke J., Burtscher M. *Fast lossless compression of scientific floating-point data // Data Compression Conference*, DCC'06 (28–30 March 2006, Snowbird, UT, USA).– IEEE Computer Society.– 2006.– ISBN 0-7695-2545-8.– pp. 133–142. ↑

[14] Burtscher M., Ratanaworabhan P. *High throughput compression of double-precision floating-point data* // *Data Compression Conference*, DCC'07 (27–29 March, 2007, Snowbird, UT, USA).– IEEE Computer Society.– 2007.– ISBN 0-7695-2791-4.– pp. 293–302. ↑

[15] Hans M., Schafer R. W. *Lossless compression of digital audio*, HPL-1999-144.– Hewlett-Packard Company.– 1999.– 37 pp. ↑

[16] Reznik Yu. A. *Coding of prediction residual in MPEG-4 standard for lossless audio coding (MPEG-4 ALS)* // *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing* (17–21 May 2004, Montreal, QC, Canada).– 2004.– ISBN 0-7803-8484-9. ↑

[17] Cutler C. C. *Differential quantization of communication signals*, U.S. patent 2605361.– 1950. ↑

[18] Salomon D. *Data Compression. The Complete Reference*, 4th ed..– London: Springer-Verlag.– 2007.– ISBN 978-1-84628-602-5.– xxviii+1092 pp. ↑

[19] Golomb S. W. *Run-length encodings* // IEEE Transactions on Information Theory.– 1966.– Vol. **12**.– No. 3.– pp. 399–401. ↑

[20] Rice R. F., Plaunt J. R. *Adaptive variable-length coding for efficient compression of spacecraft television data* // IEEE Transactions on Communication Technology.– 1971.– Vol. **19**.– No. 6.– pp. 889–897. ↑

[21] Faller N. *An Adaptive System for Data Compression*, 7th Asilomar Conference on Circuits, Systems, and Computers.– 1973.– pp. 593–597. ↑

[22] Marpe D., Schwarz H., Wiegand T. *Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard* // IEEE Transactions on Circuits and Systems for Video Technology.– 2003.– Vol. **13**.– No. 7.– pp. 620–636. ↑

[23] Рябко Б. Я. *Сжатие данных с помощью стопки книг* // Проблемы передачи информормации.– 1980.– Т. **XVI**.– № 4.– с. 16–20. ↑

[24] Malvar H. M. *Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics* // *Data Compression Conference*, DCC'06 (28–30 March 2006, Snowbird, UT, USA).– IEEE Computer Society.– 2006.– ISBN 0-7695-2545-8.– pp. 23–32. ↑

[25] Durbin J. *The fitting of time-series models* // JSTOR: Revue de l'Institut International de Statistique.– 1960.– Vol. **28**.– No. 3.– pp. 233–344. ↑

[26] Schiopu I., Munteanu A. *Deep-learning-based lossless image coding* // IEEE Transactions on Circuits and Systems for Video Technology.– 2020.– Vol. **30**.– No. 7.– pp. 1829–1842. ↑

[27] Goyal M., Tatwawadi K., Chandak S., Ochoa I. *DZip: improved general-purpose loss less compression based on novel neural network modeling* // *2021 Data Compression Conference*, DCC (23–26 March 2021, Snowbird, UT, USA).– IEEE.– 2021.– pp. 153–162. ↑

[28] Collet Y., Kucherawy M. (eds.) *Zstandard compression and the 'application/zstd' media type*, RFC 8878.– 2018.– 54 pp. ↑

[29] Ziv J., Lempel A. *A universal algorithm for sequential data compression*

// IEEE Transactions on Information Theory.– 1977.– Vol. **23**.– No. 3.– pp. 337–343. ↑

[30] Duda J. *Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding.*– 2014.– 24 pp. arXiv 1311.2540 ↑

[31] Rice R. F., Yeh P. S., Miller W. *Algorithms for a very high speed universal noiseless coding module*, JPL Publication 91-1.– Pasadena: Jet Propulsion Laboratory.– 1991.– 30 pp. ↑

[32] Elias P. *Universal codeword sets and representations of the integers* // IEEE Transactions on Information Theory.– 1975.– Vol. **21**.– No. 2.– pp. 194–203. ↑

[33] Bentley J. L., Sleator D. D., Tarjan R. E., Wei V. K. *A locally adaptive data compression scheme* // Communications of the ACM.– 1986.– Vol. **29**.– No. 4.– pp. 320–330. ↑

[34] Shevchuk Yu. *Vbinary: variable length integer coding revisited* // Program Systems: Theory and Applications.– 2018.– Vol. **9**.– No. 4(39).– pp. 239–252. ↑

[35] Dietz H. G. *The Aggregate Magic Algorithms*, Aggregate.Org online technical report.– University of Kentucky.– 2021. ↑

[36] Kiely A. *Selecting the Golomb parameter in Rice coding*, IPN Progress Report 42-159.– 2004.– 18 pp. ↑

Поступила в редакцию 18.01.2022;
одобрена после рецензирования 28.02.2022;
принята к публикации 04.04.2022.

Рекомендовал к публикации *к.ф.-м.н. С. А. Романенко*

Информация об авторе:

Юрий Владимирович Шевчук

с.н.с. Центра мультипроцессорных систем Института программных систем им. А.К. Айламазяна РАН, к.т.н. Сфера интересов: сенсорные сети, эксплуатационный мониторинг, распределенные информационно-вычислительные системы, отказоустойчивые системы, распределенное программирование

0000-0002-2327-0869
e-mail: sizif@botik.ru

Автор заявляет об отсутствии конфликта интересов.

Приложение 1. Используемые в статье коды

dec	binary	expbinary	Elias γ	Elias ω	GR(3)	vbinary2x	vbinary2x1x	vbinary2x(1,2,3x)
0	0	—	—	—	0000	00	00	00
1	1	пусто	1	0	0001	01	01	010
2	10	0	010	10 0	0010	10	10	011
3	11	1	011	11 0	0011	1100	110	1000
4	100	00	00100	101 00 0	0100	1101	1110	1001
5	101	01	00101	101 01 0	0101	1110	11110	1010
6	110	10	00110	101 10 0	0110	111100	111110	1011
7	111	11	00111	101 11 0	0111	111101	1111110	11000
8	1000	000	0001000	111 000 0	10000	111110	11111110	11001
9	1001	001	0001001	111 001 0	10001	11111100	111111110	11010
10	1010	010	0001010	111 010 0	10010	11111101	<10 бит>	11011
11	1011	011	0001011	111 011 0	10011	11111110	<11 бит>	11100
12	1100	100	0001100	111 100 0	10100	<10 бит>	<12 бит>	111010
13	1101	101	0001101	111 101 0	10101	<10 бит>	<13 бит>	111011
14	1110	110	0001110	111 110 0	10110	<10 бит>	<14 бит>	1111000
15	1111	111	0001111	111 111 0	10111	<12 бит>	<15 бит>	1111001
16	10000	0000	000010000	101001 0000 0	110000	<12 бит>	<16 бит>	1111010
17	10001	0001	000010001	101001 0001 0	110001	<12 бит>	<17 бит>	1111011
18	10010	0010	000010010	101001 0010 0	110010	<14 бит>	<18 бит>	11111000
19	10011	0011	000010011	101001 0011 0	110011	<14 бит>	<19 бит>	11111001
20	10100	0100	000010100	101001 0100 0	110100	<14 бит>	<20 бит>	11111010
21	10101	0101	000010101	101001 0101 0	110101	<16 бит>	<21 бит>	11111011
22	10110	0110	000010110	101001 0110 0	110110	<16 бит>	<22 бит>	11111100
23	10111	0111	000010111	101001 0111 0	110111	<16 бит>	<23 бит>	111111010
24	11000	1000	000011000	101001 1000 0	1110000	<18 бит>	<24 бит>	111111011
25	11001	1001	000011001	101001 1001 0	1110001	<18 бит>	<25 бит>	1111111000
...	...	...	...	...	...	...	...	...
100	1100100	100101	0000001100100	10110 1100100 0	<16 бит>	<68 бит>	<100 бит>	<30 бит>
101	1100101	100110	0000001100101	10110 1100101 0	<16 бит>	<68 бит>	<101 бит>	<30 бит>
102	1100110	100111	0000001100110	10110 1100110 0	<16 бит>	<70 бит>	<101 бит>	<31 бит>
...	...	...	...	...	...	...	...	...
1000	1111101000	111101001	0000000001111101000	111001 1111101000 0	<129 бит>	<668 бит>	<1001 бит>	<275 бит>
1001	1111101001	111101010	0000000001111101001	111001 1111101001 0	<129 бит>	<668 бит>	<1002 бит>	<275 бит>
1002	1111101010	111101011	0000000001111101010	111001 1111101010 0	<129 бит>	<670 бит>	<1003 бит>	<276 бит>
...	...	...	...	...	...	...	...	...