Рукопись, версия от 26.04.2004

ИСПОЛЬЗОВАНИЕ МЕТОДОВ СЖАТИЯ ДАННЫХ БЕЗ ПОТЕРЬ ИНФОРМАЦИИ В УСЛОВИЯХ ЖЕСТКИХ ОГРАНИЧЕНИЙ НА РЕСУРСЫ УСТРОЙСТВА-ДЕКОДЕРА

М.А. Смирнов
Санкт-Петербургский государственный университет аэрокосмического приборостроения

Рассматривается задача эффективного сжатия данных при жестких ограничениях на ресурсы декодера, в первую очередь по памяти. Сравнивается эффективность различных методов при адаптивном и статическом подходах. Для сравниваемых программ показывается взаимосвязь достигаемого коэффициента сжатия, скорости декодирования и требуемого для декодирования объема памяти (ОЗУ или ПЗУ). Основное внимание уделяется экономному кодированию текста на естественном языке.

Введение

В настоящее время большое распространение получили мобильные вычислительные устройства и различная встраиваемая техника. Часто мобильное устройство взаимодействует с некоторым стационарным в рамках модели «клиент-сервер». Например, так может строиться работа электронных записных книжек, использующихся в качестве терминала для работы с удаленной базой данных. При этом серверу передаются главным образом запросы, а клиенту — требуемые данные, т.е. обмен асимметричен, и основная нагрузка ложится на канал связи от сервера к клиенту. Поэтому при организации информационного взаимодействия между мобильным и стационарным устройством часто возникает проблема эффективного использования канала связи. Экономное кодирование пересылаемых сообщений (сжатие данных) может существенно увеличить реальную пропускную способность канала. Но применение сжатия данных затрудняется тем, что для мобильных или управляемых встраиваемых устройств характерно наличие сравнительно малых вычислительных ресурсов. Декодирование должно требовать небольших затрат оперативной памя-

ти и процессорного времени. При этом требования к алгоритму кодирования (сжатия) значительно менее жесткие.

В настоящей статье приводятся результаты по оценке применимости разных методов сжатия без потерь информации при решении описанной задачи. Сравниваются так называемые «универсальные» методы сжатия данных, т.е. те, которые, как принято считать, не ориентированы на данные специального вида. Более подробно рассматривается вопрос эффективного сжатия текстовых данных, поскольку часто передается и обрабатывается информация именно такого типа. Основное внимание уделяется соотношению между объемом памяти, требуемым для декодирования, и коэффициентом сжатия.

1. Сравниваемые методы сжатия данных

При сравнении были использованы методы сжатия на основе предсказания по частичному совпадению (PPM), на основе преобразования Барроуза-Уилера (BWT), словарные методы семейств LZ77 и LZ78. При этом для статистического кодирования применялось арифметического сжатие. В статье не приводится описания данных методов, интересующийся читатель может обратиться к [1].

Терминология

Пусть кодируется последовательность $S = \{s_i\}^N$ длины N, составленная из символов конечного алфавита A, $s_i \in A = (a_1, a_2, \ldots a_{N_A})$. Мощность A будет обозначаться как N_A, $N_A = |A|$. Конечная последовательность $W_A = s_1 s_2 \ldots s_l$, $s_i \in A$, называется словом в алфавите A. l — длина слова. Процесс кодирования заключается в отображении отдельных слов W_A, составляющих S, на множество слов W_B в кодовом алфавите B, или кодовых слов. Совокупность всех кодовых слов $\{W_B\}$ образует код. Если для всех W_B длина l одинакова, то код называется равномерным, а l определяет длину кода. Иначе под длиной кода понимается средняя длина кодовых слов на основании вероятности их использования. При посимвольном кодировании все W_A имеют $l = 1$; соответствующий код $\{W_B\}$ часто называют неблочным.

Если исходные данные S могут быть однозначно восстановлены по массиву соответствующих W_B, то отображение $W_A \to W_B$ не приводит к потере информации, т.е. является безущербным, без потерь.

Эффективность сжатия как характеристика сокращения размера представления информации относительно исходного будет определяться коэффициентом сжатия K. С учетом сложившейся традиции, K будет измеряться в «битах на байт» (бит/байт). В этом случае показывается, с помощью какого количества битов в среднем представляется 1 байт исходных (несжатых) данных. Например, $K = 4$ бит/байт соответствует сжатию в 2 раза. Чем K меньше, тем сжатие сильнее, «лучше».

Оценки затрат памяти при декодировании для разных методов

Пусть осуществляется посимвольное кодирование $s_i \in A = (a_1, a_2, \ldots a_{N_A})$, и s_i представляются с помощью равномерного кода. Тогда длина такого кода $l = \lceil \log_2 N_A \rceil$ битов.

В табл. 1 представлены оценки затрат памяти V при декодировании для различных методов. При расчете значений в байтах принималось, что исходные s_i представляются 1 байтом, т.е. $l = 8$ битов, и ссылка на ячейку памяти (массива) занимает 2 байта. Это, в частности, предполагает, что ссылка на отдельный s_i занимает 2 байта. Ограничение разме-

ра различаемого фрагмента S до 2^{16} элементов представляется разумным для интересующей задачи.

Таблица 1. Оценки затрат памяти V при декодировании для различных методов

Структура данных	V, бит	Наиболее вероятное значение V, байт
Сжатие при использовании BWT		
1. Блок, полученный при преобразовании	$N \cdot l$	N
2. Массив для определения первого столбца отсортированной матрицы	$N_A \cdot l$	N_A
3. Вектор обратного преобразования	$N \cdot \lceil \log_2 N \rceil$	$2N$
4. Массив для декодирования «стопки книг». Структуру можно совместить с (2)	$N_A \cdot l$	–
Итого		$3N + N_A$
LZ77-код		
1. Собственно декодированный фрагмент	$N \cdot l$	N
Итого		N
LZW-код		
1. Словарь. Структура словаря: • ссылка на родительскую фразу; • последний символ фразы.	$\approx N \cdot (\lceil \log_2 N \rceil + l)$	$3N$
2. Стек для формирования слова (слово формируется справа налево, т.е. последний s_i определяется в первую очередь), $L_{\max}$ — максимальная длина фразы	$L_{\max} \cdot l$	$L_{\max}$
Итого		$3N + L_{\max}$
Кодирование по Хаффману (каноническое)		
1. $base[i]$ — число нелистовых элементов на уровне i, $L_{\max}$ — максимальная длина кодового слова в битах	$\left(\log_2 \frac{N_A}{2} \right) \cdot L_{\max}$	$L_{\max}$
2. $alph[i]$ — алфавит, упорядоченный по длине кодового слова и его числовому значению	$N_A \cdot l$	N_A
3. $offs[i]$ — индекс массива $alph[i]$ такой, что $alph[offs[i]]$ — первый листовой узел (символ) на уровне i	$L_{\max} \cdot l$	$L_{\max}$
Итого		$N_A + 2L_{\max}$
Арифметическое сжатие		
Набор счетчиков частот для каждого s_i	$N_A \cdot \lceil \log_2 N \rceil$	$2N_A$
Итого		$2N_A$

Замечания и пояснения к табл. 1:

1. Оценка не включает расходы по хранению декодированной S. При необходимости такого учета следует добавить $N \cdot l$ во всех случаях, исключая LZ77.

2. Для LZW выражение $N \cdot (\lceil \log_2 N \rceil + l)$ соответствует варианту хранения N фраз. Можно дать верхнюю оценку размера словаря как $(N - 1 + N_A) \cdot (\lceil \log_2 (N - 1 + N_A) \rceil + l)$, поскольку в наихудшем случае к имеющимся по определению в словаре N_A фразам при обработке S длиной N может добавиться $N - 1$ фраз.

3. Каноническое кодирование по Хаффману отличается тем, что, во-первых, кодовые слова W_B меньшей длины численно меньше W_B большей длины, и, во-вторых, W_B равной длины численно возрастают в алфавитном порядке соответствующих им символов алфавита $a_j \in A$ [4].

4. Выбор разрядности счетчика частот при арифметическом сжатии зависит от N_A и предполагаемой функции распределения частот $a_j \in A$. Чем меньше разрядность, тем, в общем случае, грубее оценка и, соответственно, хуже сжатие. Как правило, при побайтовой обработке используются 16-разрядные счетчики или, реже, 8-разрядные.

2. Известные публикации по теме

Необходимо признать, что вопрос эффективности алгоритма с точки зрения используемого объема памяти (ОЗУ и/или ПЗУ) и скорости кодирования-декодирования нередко остается за рамками публикаций. Часто сравнение методов и алгоритмов производится только по коэффициенту (степени) сжатия K.

Сравнительно пристальное внимание уделяется проблеме экономного декодирования кодов Хаффмана. Среди разработанных алгоритмов наиболее эффективным по критериям размера используемой при декодировании памяти V и скорости декодирования является, по-видимому, способ декодирования канонического кода Хаффмана. Различные модификации этого алгоритма рассматриваются, например, в [4, 5].

В [6, 7] описываются варианты алгоритмов с PPM-моделированием низких порядков (порядок $O \le 3$), но не показана зависимость достигаемого K от V. Также практически опущено сравнение по скорости кодирования/декодирования. Простые экономные схемы контекстного моделирования описываются в [8, 9], но при этом, опять же, не дается развернутого сравнения по скорости, V и K. В [10] рассматриваются способы ускорения PPM-сжатия за счет увеличения K. Предложено кодировать не символ, а его позицию в ранжированном списке символов, встреченных в контексте, а также использовать более быстрые, чем арифметическое сжатие, но менее эффективные по критерию K способы кодирования. Вопросу зависимости $K(V)$ особого внимания не уделяется.

3. Описание программных реализаций сравниваемых методов

Сравнение эффективности разных методов производилось в двух режимах — адаптивном и статическом. В первом случае обработка адаптивна, и соответствующие структуры данных должны находиться в ОЗУ. Во втором случае модель (словарь) фиксирована и заранее задана, поэтому структуры данных могут размещаться в ПЗУ. Задача оптимизации алгоритмов и реализаций по скорости явным образом не ставилась и не решалась.

PPM

Использовалась авторская реализация PPM. Во всех случаях применялся априорный способ оценки вероятности ухода по методу D [1]. Применялись различные PPM-модели. Пусть CM^L — это контекстная модель порядка $L \le O$, соответствующая некоторому контексту C^L длины L. Параметр O — максимальный порядок контекстных моделей, определяющий порядок PPM-модели источника данных. PPM 2-0 обозначена модель, состоящая только из набора CM^2 и одной CM^0. Аналогично, PPM 3-0 включает совокупность CM^3 и одну CM^0. Обозначения PPM 2-1-0 и PPM 3-1-0 расшифровываются сходным образом.

Были рассмотрены два варианта организации структур данных для PPM, именуемые далее как (A), (B). Для обоих вариантов поиск контекстной модели для s_i выполнялся с использованием хеширования. В случае (A) контекстная модель CM^L всегда соответствует ровно одному контексту C^L, в случае (B) — набору похожих C^L, где «похожесть» задается хеш-функцией. Для (A) значению хеш-функции может соответствовать хеш-цепочка контекстных моделей CM^L, для (B) хеш-цепочка вырождается в одну контекстную модель.

Для варианта (A) использовались структуры:

1) «контекст» (экземпляр создается для встреченного контекста CM^L), имеет поля:
   - `snum` — число различных символов, встреченных в контексте, 1 байт;
   - `sptr` — указатель на массив описаний символов, встреченных в контексте, 2 байт;
   - `cnext` — указатель на следующий «контекст» в хеш-цепочке, 2 байт;
   - `cnt[]` — собственно контекст как массив символов, L байт;
2) описание символа (экземпляр создается для символа, встреченного в контексте C^L):
   - `sym` — собственно символ s_i, 1 байт;
   - `freq` — число появлений s_i в соответствующем контексте C^L, 1 байт.

В целях экономии памяти использовались 8-битовые счетчики. Если CM^L содержала 1 символ, то он описывался непосредственно в поле `sptr` структуры «контекст».

Для варианта (B) нет необходимости в полях `cnext` и `cnt[]` структуры «контекст». Поэтому количество памяти, доступной для хранения описаний контекстов и описаний символов, больше. Это может компенсировать возможное снижение точности оценки, обусловленное использованием одной CM^L для более чем одного контекста.

Во всех случаях требуется хеш-таблица, в ячейках которой записываются ссылки на хеш-цепочки контекстных моделей, имеющих одинаковое хеш-значение контекста. Для адаптивного режима требуются также структуры менеджеров контекстов и символов. Они должны обеспечивать выделение памяти при создании новых контекстных моделей (описаний символов) и сборку «мусора» при удалении. Структура менеджера должна включать, по крайней мере, массив (цепочку) указателей на свободные блоки памяти с указанием их размера. Поэтому каждый элемент структуры данных менеджера имел поля:

- `ptr` — ссылка на свободный блок памяти, 2 байт;
- `size` — размер свободного блока памяти, 2 байт.

Для обеспечения сборки мусора неиспользуемые элементы «контекст» помечались нулевым значением `snum` и `sptr`. Неиспользуемые описания символа — нулевым значением `freq`. При исчерпании памяти устранялись редко используемые CM^L и описания символов. При этом в качестве счетчика частоты употребления CM^L применялись старшие биты поля `cnext`.

Для статического режима менеджеры не нужны. Не требуется и поле `sptr`, поскольку число встреченных в контексте символов известно, и их описание может быть присоединено непосредственно к экземпляру структуры «контекст».

Собственно кодирование выполнялось с помощью интервального кодера (разновидности арифметического кодера) [1]. Для предотвращения переполнения разрядной сетки все счетчики CM^L делились на 2 при достижении максимально допустимой суммы значений всех счетчиков и при достижении счетчика некоторого символа значения 2^8.

LZW

Использовалась авторская реализация LZW. Фразы кодировались равномерно, длина кода определялась текущим размером словаря. При достижении заданного максимального размера словарь очищался. Для декодирования использовался массив элементов с полями:

- `parent` — ссылка на родительскую фразу (родительский элемент), 2 байт;
- `sym` — последний символ фразы, 1 байт.

Максимальная длина фразы $L_{\max} = 128$, что определило размер стека для декодирования. Поэтому для декодирования требовалось примерно $3N + 128$ байт, где N — размер словаря во фразах.

LZ77

Применялась авторская реализация так называемого метода LZari, при котором кодовые слова LZ экономно представляются с помощью арифметического сжатия. Использовалась типовая схема, при которой длины совпадения len и литералы s_i кодировались адаптивно в одном алфавите. Старшие биты pos кодировались адаптивно, младшие — статически, равномерным кодом. Максимальная $len = 256$.

В статическом режиме использовались заранее построенные таблицы частот для старших битов pos и для алфавита len и s_i.

Сжатие с использованием BWT

Применялась авторская модификация программы bzip2, позволившая гибко задавать размер блока для преобразования. В bzip2 для экономного представления результата преобразования используется кодирование «стопка книг» и кодирование по Хаффману. Следует заметить, что реализация кода Хаффмана в bzip2 не рассчитана на блок малой длины, что должно было систематически исказить оценки K.

Возможна такая модификация bzip2, что для декодирования будет требоваться примерно $3N + 2N_A$ байт. Приведенные ниже экспериментальные результаты основываются на этом. Соответствующая программа помечена как bzip2*.

Словарное сжатие Dict

Для статического сжатия текстов был разработан словарный алгоритм, именуемый ниже Dict. Фиксированный словарь Dict состоит из упорядоченных по частоте употребления слов, выделенных из опорного текста. При ограничениях на V в словарь вносятся такие слова, использование которых в качестве фраз, как ожидается, даст наименьший K. W_A, не найденные в словаре, кодируются посимвольно с помощью арифметического кодера. Найденные W_A арифметически кодируются через номер в словаре, при этом старшие биты номера кодируются исходя из их частоты появления. Младшие биты номера кодируются как равновероятные. Предварительно выполняется обратимое устранение заглавных букв (см. ниже). Самые частые символы-разделители — пробелы — кодируются только при необходимости. По умолчанию считается, что перед словом стоит пробел.

Обратимые преобразования текстовых данных

Известно несколько обратимых преобразований (способов препроцессинга), часто позволяющих существенно улучшить сжатие текстовых данных.

1. Замена по фиксированному словарю часто встречающихся последовательностей символов. Если некоторый фрагмент последовательности S совпадает с фразой словаря, то он заменяется кодовым словом фразы. Для английского языка использовался словарь,

предложенный автором в [1]. Он содержит 45 двухбуквенных фраз, 25 трехбуквенных и 16 четырехбуквенных. Для хранения такого словаря и собственно декодирования требуется примерно 300 байт. В качестве кодовых слов применялись незадействованные в ASCII байтовые значения 128…255.

2. Устранение заглавных букв. Если слово W_A, рассматриваемое как последовательность s_i, ограниченная некоторыми символами-разделителями (пробел, знаки препинания и т.п.), начинается с заглавной буквы a_j, то:

$$W_A \rightarrow < флаг >< строчная\ a_j >< остаток\ W_A > .$$

3. Модификация символов-разделителей. Перед символом-разделителем добавляется пробел, что позволяет улучшить сжатие при использовании PPM и, иногда, BWT.

4. Результаты при адаптивном подходе

Следует отметить, что, безусловно, все приводимые сравнительные результаты целесообразно рассматривать как довольно грубую оценку. Во-первых, характеристики существенно зависят от типа обрабатываемых данных. Во-вторых, возможна модификация каждого алгоритма и реализации, позволяющая достичь лучших характеристик, и нет оснований считать, что в рамках проведенных экспериментов все методы «были в равных условиях». В-третьих, равенство по использованию V было приблизительным.

Сравнения проводилось на наборе файлов, традиционно используемых для сравнения программ сжатия данных без потерь. Тест был сформирован из файлов классического тестового набора Calgary Compression Corpus[1] и альтернативного набора VYTEST[2]. Описания файлов приведены в табл. 2.

Таблица 2. Описание тестового набора

Название	Размер, байт	Описание
bib	111261	библиографический список в формате UNIX “refer”, ASCII
book1	768771	художественная книга на английском языке: T. Hardy “Far from the madding crowd”, неформатированный текст ASCII
book2	610856	техническая книга на английском языке: I. Witten “Principles of computer speech”, формат UNIX “troff”, ASCII
fileware	427520	руководство пользователя на английском языке к набору утилит Fileware версии 3.0, формат Microsoft Word, содержит разнородные данные
os2	594821	конфигурационный файл для операционной системы OS/2, содержит разнородные данные
paper2	82199	техническая статья на английском языке: I. Witten “Computer (in)security”, формат UNIX “troff”, ASCII
paper4	13286	техническая статья на английском языке: John G. Cleary “Programming by example revisited”, формат UNIX “troff”, ASCII
progc	39611	программа на языке C, ASCII
progp	49379	программа на языке Паскаль, ASCII
stand	1639139	художественная книга на русском языке: С. Кинг “Армагеддон”, форматированный текст ASCII.
trans	93695	расшифровка терминальной сессии, формат редактора “EMACS”, ASCII
wcc386	536624	исполнимый файл Watcom версии 10.0 (компилятор с языка C)

При проведении экспериментов не учитывались затраты на хранение декодированной S. При необходимости этого надо учесть соответствующую поправку для всех реализаций, исключая LZari. Интегральный коэффициент сжатия $\overline{K}$ в зависимости от объема V

[1] http://links.uwaterloo.ca/calgary.corpus.html
[2] http://www.compression.ru/ybs/

ОЗУ, доступного при декодировании, приведен для всего теста в табл. 3. $\overline{K}$ рассчитывался как обычное среднее коэффициентов K для отдельных файлов. $\overline{K}$ не указан для тестов, не проводившихся в силу их очевидной избыточности. Для сравнения даны результаты для архиватора PKZIP версии 2.04g, запускавшегося в режиме максимального сжатия (опция «-ex»). В PKZIP реализован метод LZH с блочной адаптацией кода Хаффмана, что обеспечивает высокую скорость декодирования. Возможна модификация декодера PKZIP, требующая немногим более 32 кбайт ОЗУ для декодирования.

Таблица 3. Зависимость $\overline{K}(V)$ для тестового набора

V, кбайт	PPM 3-0 (A)	PPM 3-0 (B)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-0 (B)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8			3.29	3,40	3,55	3.20	3,89	4,28	**3,13**	
16	3,29	3,38	**2.91**	3,13	3,19	3.03	3,46	4,05	3,01	
32	3,05	3,20	**2.75**	3,00	3,14	2.94	3,11	3,90	2,93	2,74
64	2,84	3,08	**2.61**	2,98	3,10	2.93	2,87	3,77	2,89	
128			**2.55**			2.93	2,68	3,72	2,87	

Из табл. 3 видно, что при $V \geq 16$ кбайт преимущество по критерию $\overline{K}$ имеют реализации PPM. Максимум достигается для PPM 3-1-0 (A). Следует также отметить, что вариант реализации (A) стабильно обеспечивает лучшее сжатие, чем (B). При V порядка 8 кбайт и, очевидно, менее доминирует LZari, что можно было предположить заранее. В табл. 4 приведено сравнительное время декодирования $\overline{T}_{дек}$ всего набора, при этом время декодирования для PKUNZIP (декодер для PKZIP) принято за 1. Как следовало ожидать, декодирование для PPM в разы медленнее, чем для словарных методов.

Таблица 4. Зависимость $\overline{T}_{дек}(V)$ для тестового набора

V, кбайт	PPM 3-0 (A)	PPM 3-0 (B)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-0 (B)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8			26,3	16,5	16,0	14,8	3,89	3,0	**2,6**	
16	16,3	14,0	11,5	11,3	9,0	10,0	3,46	2,7	**1,7**	
32	13,8	11,0	9,8	8,5	8,5	7,5	3,11	2,5	**1,6**	1,00
64	12,0	10,3	8,3	8,5	9,0	6,8	2,87	2,2	**1,5**	
128			8,0			6,8	2,68	2,1	**1,5**	

В табл. 5 дана детальная статистика для $V = 64$ кбайт. LZari имеет преимущество над PPM только на высоко избыточных файлах. С увеличением V отставание может быть устранено за счет использования PPM-модели большего порядка O.

Таблица 5. K для отдельных файлов при $V = 64$ кбайт

Файл	PPM 3-0 (A)	PPM 3-0 (B)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-0 (B)	PPM 2-1-0 (A)	BZIP2*	LZW	LZari	PKZIP -ex (справочно)
bib	2,47	2,66	**2,17**	2,70	2,76	2,63	2,65	3,47	2,53	2,53
book1	2,75	2,86	**2,61**	2,95	2,96	2,93	3,24	3,68	3,17	3,25
book2	2,50	2,77	**2,37**	2,88	2,94	2,87	2,86	3,56	2,69	2,70
fileware	2,87	3,58	**2,65**	2,97	3,44	2,91	2,69	4,16	2,67	2,36
os2	2,49	2,81	2,35	2,61	2,75	2,57	1,92	2,91	**1,83**	1,60
paper2	2,64	2,79	**2,47**	2,88	2,89	2,85	2,87	3,45	2,94	2,88
paper4	3,17	3,29	**2,93**	3,27	3,29	3,19	3,12	4,06	3,59	3,31
progc	2,82	3,00	**2,55**	2,93	3,01	2,85	2,72	3,78	3,00	2,69
progp	2,08	2,32	**1,87**	2,33	2,38	2,26	2,03	3,05	2,08	1,81

Файл	PPM 3-0 (A)	PPM 3-0 (B)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-0 (B)	PPM 2-1-0 (A)	BZIP2*	LZW	LZari	PKZIP -ex (справочно)
stand	3,09	3,20	**2,86**	3,13	3,17	3,12	3,41	3,90	3,26	3,36
trans	2,04	2,32	1,83	2,37	2,49	2,32	2,12	3,23	**1,76**	1,66
wcc386	5,12	5,36	4,70	4,77	5,08	**4,61**	4,79	5,94	5,19	4,67
$\overline{K}$	2,84	3,08	**2,61**	2,98	3,10	2,93	2,87	3,77	2,89	2,74
$\overline{T}_{дек}$	13,8	7,0	9,5	5,8	5,5	5,5	4,9	3,8	**2,9**	1,0

Зависимость $\overline{K}(V)$ носит убывающий экспоненциальный характер и может быть хорошо аппроксимирована функцией $c+e^{aV}$, где c и a константы. Например, экспериментальные значения $\overline{K}(V)$ для PPM 3-1-0 (A), приведенные выше в табл. 3, достаточно точно ложатся на график кривой $2.57+e^{-0.056V}$, так что остаются неучтенными всего 4% от вариации $\overline{K}(V)$. Для рассмотренного тестового набора и диапазона изменения V наибольшая скорость убывания $\overline{K}(V)$ — определяется параметром a — отмечена у PPM и BWT[3].

На рисунке показана зависимость K от объема обработанных данных для book1 при $V=64$ кбайт. При небольшом размере текста разница между K для bzip2*, PPM 3-1-0 (A) и PPM 2-1-0 (A) несущественная и составляет менее 10%.

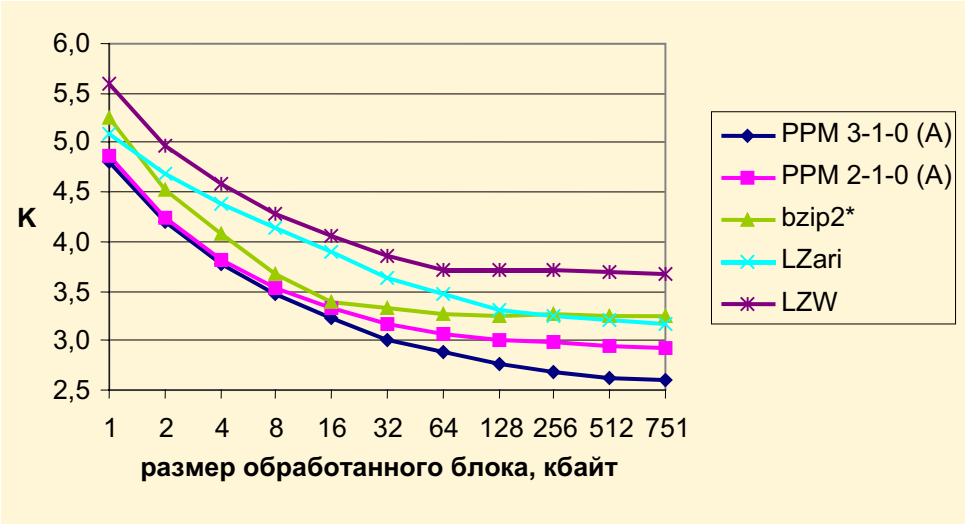

Рисунок. Изменение K по мере обработки book1 при $V=64$ кбайт

На примере book1 были рассмотрены варианты схем сжатия с использованием обратимых преобразований текстовых данных. В табл. 6 приведены K для вариантов без препроцессинга (далее «БП»). Значения $\overline{K}$, полученные при устранении заглавных букв и словарной замене (препроцессинг 1, далее «П1»), представлены в табл. 7, полученные при устранении заглавных букв, словарной замене и модификации символов-разделителей (препроцессинг 2, далее «П2») — в табл. 8.

Таблица 6. $K(V)$ для book1 без препроцессинга

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8		3,46	3,41	**3,28**	4,20	4,34	3,62	
16	3,50	3,13	3,07	**3,04**	3,81	4,09	3,44	

[3] Что согласуется с теоретическими оценками скорости сходимости к энтропии источника информации, известными для рассматриваемых методов.

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
32	3,11	**2,85**	2,95	2,94	3,49	3,88	3,29	3,25
64	2,75	**2,61**	2,95	2,93	3,24	3,68	3,17	
128		**2,49**		2,93	3,02	3,53	3,08	

Таблица 7. $K(V)$ для book1 при препроцессинге 1

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8		3,34	3,47	**3,27**	4,14	4,18	3,59	
16	3,65	3,24	3,17	**3,01**	3,73	3,93	3,45	
32	3,42	3,06	2,80	**2,68**	3,38	3,71	3,29	3,04
64	3,11	2,77	2,55	**2,51**	3,10	3,50	3,14	
128		2,52		**2,38**	2,89	3,33	3,02	

Таблица 8. $K(V)$ для book1 при препроцессинге 2

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8		3,29	3,43	**3,19**	4,25	4,28	3,59	
16	3,56	3,11	3,03	**2,92**	3,79	4,01	3,43	
32	3,29	2,92	2,71	**2,65**	3,41	3,77	3,27	3,12
64	2,92	2,64	**2,49**	**2,48**	3,13	3,57	3,13	
128		**2,41**		2,43	2,91	3,39	3,02	

В табл. 9 показана зависимость $T_{дек}(V)$ для book1 без препроцессинга. Табл. 10 иллюстрирует изменение $T_{дек}(V)$ при использовании препроцессинга.

Таблица 9. $T_{дек}(V)$ для book1 без препроцессинга

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	bzip2*	LZW	LZari	PKZIP -ex (справочно)
8		41,0	23,8	30,0	5,2	4,8	**3,8**	
16	31,3	21,0	11,0	12,5	5,5	4,7	**3,3**	
32	21,0	12,0	6,3	6,8	5,7	4,2	**3,1**	1,0
64	13,8	9,5	5,8	6,3	5,8	3,8	**2,9**	
128		9,0		6,0	6,0	3,6	**2,4**	

Таблица 10. $T_{дек}(V)$ для book1 при препроцессинге

V, кбайт	PPM 3-1-0 (A)			PPM 2-1-0 (A)			PKZIP -ex (справочно)
	БП	П1	П2	БП	П1	П2	
8	41,0	**37,3**	39,0	**30,0**	33,3	32,8	
16	21,0	**20,8**	21,8	**12,5**	20,0	18,3	
32	**12,0**	14,8	14,8	**6,8**	13,0	11,0	1,0
64	**9,5**	11,5	11,5	**6,3**	9,5	7,8	
128	**9,0**	11,3	11,3	**6,0**	7,5	6,8	

Использование препроцессинга в среднем существенно улучшает сжатие для всех алгоритмов. Наибольший эффект достигается для PPM 2-1-0 — в ряде случаев отмечено уменьшение K на 25%. Но в рамках рассмотренных алгоритмов PPM и диапазона изменения V применение препроцессинга, как правило, замедляет декодирование в 1.1-1.9 раза.

5. Результаты при статическом подходе

Статический режим предполагает хранение и использование фиксированной модели (словаря). Это специфическая задача, которая, по-видимому, должна решаться с максимальным учетом специфики данных, имеющихся ресурсов, особенностей аппаратного и, возможно, программного обеспечения. Поэтому даются результаты, полученные на примере сжатия только текстового файла book1. Сжатие на основе обычного BWT в статическом режиме невозможно, метод LZW не рассматривался в силу сравнительно плохих характеристик. Во всех случаях первые 512 кбайт исходного файла использовались для создания и настройки модели (словаря), которая затем применялась без адаптации при сжатии оставшейся части файла размером $751-512=239$ кбайт. Результаты для вариантов без препроцессинга сведены в табл. 11, результаты для сжатия с препроцессингом — в табл. 12 (для Dict препроцессинг не имеет смысла, поэтому вариант не рассматривался).

Таблица 11. $K(V)$ в статическом режиме без препроцессинга

V, кбайт	PPM 3-0 (A)	PPM 3-1-0 (A)	PPM 2-0 (A)	PPM 2-1-0 (A)	LZari	Dict	
8		3,44	3,46	**3,34**	3,59	3,39	3,35
16	3,47	3,17	3,03	**2,99**	3,40	3,17	3,15
32	3,04	2,98	2,92	**2,91**	3,24	3,00	2,97
64	2,65	**2,61**	2,91	2,90	3,10	2,85	2,84
128		**2,41**		2,90	2,99	2,73	

Таблица 12. $K(V)$ в статическом режиме с препроцессингом

V, кбайт	PPM 3-0 (A)		PPM 3-1-0 (A)		PPM 2-0 (A)		PPM 2-1-0 (A)		LZari	
	П1	П2	П1	П2	П1	П2	П1	П2	П1	П2
8			3,61	3,33	3,47	3,43	3,25	**3,19**	3,58	3,58
16	3,66	3,59	3,14	2,97	3,10	3,00	2,94	**2,88**	3,42	3,40
32	3,38	3,24	2,92	2,78	2,88	2,77	2,67	**2,64**	3,25	3,23
64	2,97	2,80	2,62	2,53	2,53	**2,44**	2,51	**2,44**	3,08	3,07
128			2,35	**2,27**			2,31	2,38	2,93	2,93

Алгоритмы PPM 2-1-0 и PPM 3-1-0 обеспечивают лучшее сжатие. При использовании препроцессинга любого типа сжатие, как правило, улучшается (соответствует уменьшению K). Зависимость $K(V)$ имеет убывающий экспоненциальный характер.

В табл. 13 показано, как изменяется $T_{дек}(V)$. За 1 принято $T_{дек}$ при использовании PKUNZIP. Видно, что, как и при адаптивном подходе, $T_{дек}$ при использовании препроцессинга может увеличиваться на десятки процентов.

Таблица 13. $T_{дек}(V)$ в статическом режиме

V, кбайт	PPM 3-1-0 (A)			PPM 2-1-0 (A)			LZari			Dict
	БП	П1	П2	БП	П1	П2	БП	П1	П2	БП
8	8,5	7,7	8,1	6,2	6,9	6,8	3,5	3,7	3,7	3,9
16	9,1	9,0	9,4	5,4	8,6	7,9	2,9	3,2	3,2	3,1
32	7,5	9,2	9,2	4,2	7,8	6,8	2,7	2,9	2,9	2,8
64	5,9	7,2	7,2	3,9	5,9	5,3	2,4	2,6	2,7	2,5
128	5,5	7,0	7,0	4,4	5,5	5,1	2,0	2,2	2,3	2,1

Следует отметить, что в реализации Dict словарь представлялся в исходном виде, без сжатия. За счет компактного представления словаря возможно уменьшение K для Dict при $V=const$. Скорость при этом, вероятнее всего, уменьшится, так как потребуется декодировать фразы словаря при каждой словарной подстановке.

Выводы

При адаптивном сжатии в случае наличия до 16 кбайт ОЗУ в общем случае целесообразно применение алгоритма типа LZ77. При большем V возможны варианты, и выбор зависит от типа данных и требований к $T_{дек}$. Если нет очень жестких ограничений по $T_{дек}$, и обрабатываются текстовые данные, целесообразно использовать алгоритм сжатия на основе контекстного моделирования типа PPM. При объеме $V < 64$ кбайт разумно применять модель с длинами контекстов 2, 1, 0 (PPM 2-1-0), при $V \geq 64$ имеет смысл рассмотреть PPM 3-1-0 или даже более сложные схемы. При этом превосходство по коэффициенту сжатия K алгоритмов PPM над схемами типа LZ77 может составлять 25% и более.

При статическом подходе, предполагающем хранение модели источника данных (словаря) в ПЗУ и минимальное использование ОЗУ, наблюдается в целом сходная картина в случае сжатия текстовых данных. В случае жестких ограничений на $T_{дек}$ и доступный объем ПЗУ V следует применять словарные алгоритмы. Иначе целесообразно проанализировать оправданность использования алгоритма на основе PPM. Но, скорее всего, даже при тщательном построении алгоритма и эффективной реализации скорость декодирования для PPM будет в разы меньше, чем для словарных схем. Перспективным является словарный алгоритм, при котором фразами являются естественные слова языка.

Как при адаптивном, так и при статическом подходе использование специальных обратимых преобразований совместно с основным алгоритмом сжатия может существенно улучшить сжатие текстовых данных — отмечается уменьшение K на 5-25%. Но, как правило, это замедляет декодирование в 1.1-1.9 раза.

Благодарности

Спасибо Вадиму Юкину за ряд полезных замечаний и советов.

Литература

1. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. – 384 с.
2. Gallager R.G. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668-674, Nov. 1978.
3. Burrows M., Wheeler D.J. A Block-sorting Lossless Data Compression Algorithm. *SRC Research Report 124, Digital Systems Research Center, Palo Alto*, May 1994.
4. Moffat A., Turpin A. On the Implementation of Minimum Redundancy Prefix Codes. *IEEE Transactions on Communications*, 45(10):1200-1207, Oct. 1997.
5. Hirschberg D., Lelewer D. Efficient decoding of prefix codes. *Communications of the ACM*, 33(4):449-459, 1990.
6. Lelewer D.A., Hirschberg D.S. Streamlining Context Models for Data Compression. *Proceedings of IEEE Data Compression Conference*, Snowbird, Utah, Apr. 1991, pp. 313-322.
7. Lelewer D.A., Hirschberg D.S. An Order-2 Context Model for Data Compression With Reduced Time and Space Requirements. *Technical Report N90-33*, Department of Information and Computer Science, University of California, Irvine, 1990.
8. Langdon G., Rissanen J. A Double-Adaptive File Compression Algorithm. *IEEE Transactions on Communications*, 31(11):1253-1255, Nov. 1983.
9. Bell T., Witten I., Cleary J. Modeling for Text Compression. *ACM Computing Surveys*, 21(4):557-591, Dec. 1989.
10. Howard P.G. The design and analysis of efficient lossless data compression systems. *PhD thesis*, Brown University, Providence, Rhode Island, 1993.