Алгоритмы сжатия без потерь разностных целочисленных последовательностей при помощи оптимизации их разбиения на интервалы с постоянной битовой глубиной значений

А. Е. Хмельнов

Институт динамики систем и теории управления СО РАН, Иркутск, Россия
Контактный e-mail: hmelnov@icc.ru

Рассмотрен алгоритм сжатия без потери информации целочисленных данных, значения которых распределены преимущественно вблизи нуля. Такие данные получаются, например, при разностном кодировании целочисленных последовательностей, представляющих постепенно изменяющиеся величины (величины, принимающие близкие значения в соседних точках). По степени сжатия для этого вида данных алгоритм сравним с ZLib или превосходит его в режиме Z_BEST_COMPRESSION. Он требует значительно меньше времени как при сжатии, так и при распаковке, поскольку характеризуется линейной вычислительной сложностью.

Предложенный алгоритм является разновидностью алгоритма VSEncoding, расширяющей его возможности: рассмотрено кодирование знаковых чисел и применение алгоритма для произвольных разностных последовательностей, учтён более общий способ кодирования заголовков интервалов. Предложен критерий остановки поиска, позволяющий находить точный минимум длины кодовой последовательности, а также способ достижения точного минимума при работе с буфером ограниченного размера, в который не помещается вся кодируемая последовательность. Проведено сравнение работы рассматриваемых алгоритмов при сжатии растров большого объёма.

Ключевые слова: сжатие без потери информации, специализированный алгоритм сжатия, динамическое программирование.

Введение

Несмотря на постоянный рост аппаратных возможностей вычислительной техники, потребность в компактном хранении информации сохраняется и продолжает увеличиваться, поскольку параллельно растут объёмы собираемых и обрабатываемых цифровых данных. Область сжатия данных очень хорошо исследована, и даже при принятии решения об использовании некоторого готового алгоритма сжатия разработчики сталкиваются с достаточно непростым выбором. Если учесть особенности конкретных данных, то можно предложить специализированные алгоритмы их сжатия, которые будут превосходить готовые универсальные алгоритмы по различным характеристикам, существенным для разрабатываемого приложения.

В работе рассмотрен принцип построения алгоритмов быстрого сжатия без потери информации целочисленных данных, значения которых распределены преимущественно вблизи нуля. Такие данные получаются, например, при разностном кодировании целочисленных последовательностей, представляющих постепенно изменяющиеся величины (принимающие близкие значения в соседних точках). Для этого вида данных алгоритм, построенный по рассматриваемому принципу, по степени сжатия сравним с ZLib [1] или превосходит его в режиме Z_BEST_COMPRESSION, но он требует значительно меньше времени как при сжатии, так и при распаковке, поскольку характеризуется линейной вычислительной сложностью. Начальная стадия работы рассматриваемых алгоритмов сжатия состоит в разделении мантиссы и порядка целочисленных значений и аналогична работе алгоритма UNIC (Universal Coding of Integers) вида SEM (Separate Exponents and Mantissas) [2], однако, в отличие от UNIC, для представления порядка используется разбиение последовательности на интервалы постоянной битовой глубины. Для оптимизации разбиения последовательности на интервалы с целью минимизации объёма памяти, используемой для её представления, применяется метод динамического программирования.

Первоначально в работе использовалась оригинальная схема применения метода динамического программирования и соответствующий алгоритм назывался BCRL (Bit Count Run-Length encoding). Этот алгоритм был упомянут в работах, посвящённых форматам данных, использующим сжатие разностных последовательностей [3]. Последующее знакомство с алгоритмом VSEncoding [4], его реализация и практическое тестирование показали, что предложенная в нем схема более проста, но достаточно эффективна для задачи сжатия разностных целочисленных последовательностей. Кроме того, сделанные доработки алгоритма позволили находить точный минимум объёма сжатых данных. В результате в качестве основного алгоритма, применяемого для сжатия разностных последовательностей, был использован доработанный алгоритм VSEncoding — алгоритм VSEopt.

Таким образом, основными результатами данной работы являются: адаптация алгоритма VSEncoding к задаче сжатия разностных целочисленных последовательностей; вывод критерия прекращения поиска, позволяющего находить точный минимум объёма сжатых данных; способ поиска начального фрагмента данных, которые можно записать в поток без ущерба для нахождения оптимального разбиения последовательности на интервалы при использовании для сжатия данных буфера ограниченного размера, а также результаты тестирования рассматриваемых алгоритмов на конкретных примерах данных.

1. Разностные последовательности

Любую конечную целочисленную последовательность $\{S_i\}$ можно охарактеризовать диапазоном значений, принимаемых её элементами $[S_{\min}, S_{\max}]$:

$$S_{\min} = \min_j(S_j) \le S_i \le S_{\max} = \max_j(S_j).$$

Такой диапазон не всегда полностью использует все возможные значения n-байтных знаковых $[-2^{8n-1}\ ,\ 2^{8n-1}-1]$ или беззнаковых $([0\ ,\ 2^{8n}-1])$ целых чисел. Ещё в большей степени неполным использованием диапазона значений n-байтных целых чисел,

служащих для представления элементов последовательности, могут характеризоваться отдельные её подпоследовательности.

Часто целочисленные последовательности являются результатом измерения и дискретизации некоторой постепенно изменяющейся величины. Примеры таких величин: яркость пикселя на растровом изображении; высота рельефа, представленного на регулярной сетке; серия измерений, поступающих с внешнего аналогово-цифрового датчика; и т. д. Несмотря на то что элементы такой последовательности могут быть распределены на достаточно широком интервале, модуль разности соседних элементов, как правило, оказывается значительно меньше их абсолютных величин. Поэтому, если рассмотреть разностную последовательность $\{D_i = S_{i+1} - S_i\}$, то, хотя потенциально диапазон значений её элементов может даже увеличиться: $S_{\min} - S_{\max} \leq D_{\min}$, $D_{\max} \leq S_{\max} - S_{\min}$, появляется большое количество подпоследовательностей, для которых диапазон значений существенно сокращается. Кроме наиболее простых (разностных) последовательностей, подобными характеристиками обладают последовательности, получающиеся, например, после применения целочисленных вейвлет-преобразований (лифтинг [5]). Далее для простоты будем называть все подобные последовательности разностными.

2. Применимость алгоритмов сжатия к разностным последовательностям

В литературе, посвящённой алгоритмам сжатия данных [2, 6, 7], большое внимание уделяется статистическим и словарным методам. Статистические методы позволяют более компактно кодировать часто встречающиеся значения, стремясь добиться степени сжатия, определяемой энтропией сигнала. При этом, если метод Хаффмана даёт возможность достичь этого результата только для вероятностей символов, являющихся степенями $1/2$, то арифметическое и интервальное кодирование позволяет практически достичь задаваемого энтропией порога. При применении более сложных вероятностных методов, например PPM, используемых в архиваторе RAR, сжимаемые данные рассматриваются как результат работы марковского случайного процесса. При этом вероятность появления символа оценивается с учётом его предшественников. Словарные методы позволяют кодировать повторяющиеся подпоследовательности с использованием ссылок на уже обработанные данные. Также описываются блочные методы, при использовании которых последовательность разбивается на блоки некоторого заранее заданного и не зависящего от данных размера, после чего к ним применяется преобразование, облегчающее дальнейшее кодирование, например дискретное косинусное преобразование, или преобразование Барроуза — Уилера (BWT).

Рассмотрим применимость упомянутых методов для сжатия разностных последовательностей. Поскольку такие последовательности оказываются практически случайными, вероятность встретить в них повторяющиеся подпоследовательности достаточно низка. Это существенно снижает эффективность словарных методов, поэтому, например, в литературе по сжатию целочисленных последовательностей [4, 8] возможность использования таких методов не рассматривается. Слабая корреляция между соседними элементами не позволяет рассчитывать на эффективность при применении марковских моделей. Блочные методы могут быть применимы, однако часть из них, как, например, дискретное косинусное преобразование, преобразуют целые числа в действительные и поэтому ориентированы на сжатие с потерями. Преобразование BWT по су-

ти является конкурентом словарных методов и также не сможет дать более простую последовательность при отсутствии явных повторений. Таким образом, в нашем распоряжении остаются вероятностные методы, работа которых может быть улучшена путем учёта близости распределения сигнала к нормальному.

3. Методы сжатия целочисленных последовательностей

В литературе по общим методам сжатия данных задача кодирования последовательностей целых чисел не рассматривается, описываются только способы кодирования отдельных целочисленных значений (см., например, [2, 7]). При этом алгоритмы сжатия последовательностей натуральных чисел активно используются в поисковых системах для хранения, например, списков индексов использованных в документе слов [4, 8]. Поскольку такие списки по сути представляют множества, в них отсутствуют повторения и их элементы можно упорядочить для ускорения поиска и облегчения сжатия. Таким образом, при этом рассматривается задача компактного представления возрастающей последовательности целых чисел. Для сокращения диапазона значений элементов такой последовательности вместо исходной кодируется последовательность разностей значений соседних элементов. В силу отсутствия повторений и возрастания исходной последовательности последовательность разностей индексов содержит положительные целые числа. Другая задача, которая требует сжатия подобных последовательностей, — это компактное хранение индексов значений в гиперкубе в OLAP-системе [9]. Здесь также кодируются разности между упорядоченными целыми числами, представляющими кортежи — координаты точек в гиперкубе.

Использование специализированных алгоритмов, предназначенных для кодирования последовательностей целых чисел, позволяет получить более высокое сжатие, чем это могут сделать универсальные алгоритмы. Интересно, что, согласно приведённым в работе [4] результатам тестирования алгоритма VSEncoding, рассматриваемые алгоритмы могут “победить энтропию”, т. е. сжать последовательность сильнее, чем теоретический предел для методов, выполняющих независимое кодирование значений, $-\sum_i p_i \cdot \log_2 p_i$ битов на значение, где p_i — вероятность значения i для источника информации, генерирующего последовательность независимых случайных величин. Это происходит потому, что теоретический предел получен для кодирования значений по отдельности, а алгоритмы сжатия последовательностей могут обнаруживать и использовать для лучшего сжатия общие свойства соседних значений.

Необходимо отметить, что в работах по сжатию индексов для поисковых систем рассматривается сценарий использования алгоритмов декомпрессии, который существенно отличается от применяемого в данной работе: предполагается, что индексы документов уже находятся в оперативной памяти, но в сжатом виде, при этом требуется максимально быстро их распаковывать для использования. Такая постановка задачи существенно смещает оценку качества алгоритмов в сторону быстродействия, например, иногда выполняется выравнивание сохраняемых данных в памяти (что требует записи нескольких лишних неиспользуемых битов) ради ускорения их чтения.

Основным сценарием использования алгоритмов декомпрессии в данной работе является распаковка считанных с диска данных сразу после чтения. В этом случае сокращение объёма передаваемых из медленной памяти (читаемых с диска) данных может оправдать работу более совершенного алгоритма сжатия иногда и по времени, поскольк-

ку за то время, которое не было потрачено на чтение лишних данных с диска, может быть выполнен более медленный, но использующий более хорошее сжатие алгоритм. Таким образом, основным критерием качества алгоритма далее будем считать достигаемую степень сжатия.

4. Минимальная длина представления целого числа (битовая глубина)

В n-битном беззнаковом целом числе могут быть представлены значения в диапазоне $[0, 2^n - 1]$, а в знаковом (в дополнительном коде) — $[-2^{n-1}, 2^{n-1} - 1]$. Отсюда, рассуждая в обратном направлении, получим, что для представления беззнакового числа U минимально необходимое количество битов равно

$$L_u(U) = \min_{U<2^n} n = \begin{cases} \min_{\log_2(U)<n} n = \lfloor \log_2(U) \rfloor + 1 & \text{при } U > 0, \\ 0 & \text{при } U = 0, \end{cases}$$

т. е. для значения 0 будем считать, что оно помещается в 0 битов. Действительно, при кодировании последовательности, про которую известно, что она состоит только из нулевых значений, достаточно указать её длину — сами значения сохранять не требуется.

В литературе обычно употребляется другая форма записи L_u (эквивалентная рассмотренной), позволяющая обойтись без вариантов:

$$L_u(U) = \min_{U+1\le 2^n} n = \min_{\log_2(U+1)\le n} n = \lceil \log_2(U+1) \rceil,$$

но для рассмотрения всех случаев кодирования знаковых чисел удобнее будет воспользоваться первым способом.

Аналогично для представления знакового числа I минимально необходимое количество битов

$$L_s(I) = \min_{-2^{n-1}\le I<2^{n-1}} n,$$

$$L_s(I) = \begin{cases} \min_{I<2^{n-1}} n = L_u(I) + 1 & \text{при } I > 0, \\ 0 & \text{при } I = 0, \\ \min_{-I-1<2^{n-1}} n = L_u(-I-1) + 1 & \text{при } I < 0, \end{cases}$$

и после подстановки выражения для L_u получаем

$$L_s(i) = \begin{cases} \lfloor \log_2(I) \rfloor + 2 & \text{при } I > 0, \\ 0 & \text{при } I = 0, \\ 1 & \text{при } I = -1, \\ \lfloor \log_2(-I-1) \rfloor + 2 & \text{при } I < -1. \end{cases}$$

Для значения 0, как и для беззнаковых чисел, считаем, что оно помещается в 0 битов. Представление значения 0 в 0 битах позволяет очень компактно кодировать однородные подпоследовательности, разностный образ которых состоит из одних нулей. Для представления значения -1 требуется 1 бит, поэтому, если некоторый интервал состоит только из разностей 0 и -1, то его можно закодировать, расходуя 1 бит на значение. Для знакового представления положительных чисел требуется не менее чем 2 бита. Будем называть количество битов, используемых для записи целого числа, *битовой глубиной* представления целого числа.

5. Сокращение длины кодовой последовательности за счёт сокращения битовой глубины значений

Элементы разностных последовательностей необходимо рассматривать как знаковые целые числа, поэтому для вычисления минимального количества битов, требующихся для представления их значений, следует использовать функцию L_s. При работе с разностной последовательностью можно было бы существенно сэкономить используемую для её представления память, если бы удалось закодировать каждое значение минимально необходимым для его представления количеством битов, поскольку большинство этих значений находится вблизи нуля и для их представления требуется не слишком много битов. Величина

$$L_{\min}(S) = \sum_i L_s(S_i)$$

позволяет оценить снизу возможный объём упакованных данных, связанный с таким способом сжатия. Однако в общем случае невозможно достичь эту нижнюю границу, поскольку ещё необходимо потратить память на запоминание информации о битовой глубине, используемой для представления значения каждого из элементов последовательности.

В работе [10] с учётом особенности нормального распределения гауссова белого шума предложен алгоритм сжатия rdc (Random Data Encoder), основанный на оптимальном (для кодируемой последовательности) выборе порога, разделяющего короткие и длинные значения, после чего вся последовательность разбивается на блоки, состоящие из ряда коротких и, может быть, одного длинного значения. Каждый блок начинается с заголовка, в котором задается количество коротких элементов, а также содержится признак наличия длинного элемента. Для выбора порога битовой глубины достаточно проанализировать гистограмму битовых глубин значений элементов последовательности. Несмотря на использование столь простой схемы сжатия, при её применении к данным, полученным в результате декорреляции исходной последовательности посредством применения вейвлет-преобразования, степень сжатия оказывается лучше, чем у gzip-9 (то же, что и ZLib в режиме Z_BEST_COMPRESSION).

6. Представление информации о битовой глубине посредством разбиения на интервалы

Основной принцип работы рассматриваемых в данной работе алгоритмов состоит в том, что для хранения информации об используемых для представления значений количествах битов последовательность можно разбить на интервалы постоянной глубины. Для каждого интервала запоминается заголовок с информацией о его битовой глубине D и количестве значений L, после чего следуют сами значения, занимающие $D \cdot L$ битов. При этом каждое значение S_i не обязательно должно быть представлено с глубиной $L_s(S_i)$, главное, чтобы выполнялось условие $D \geq L_s(S_i)$ для того интервала, которому принадлежит значение. На увеличение битовой глубины, используемой для представления некоторых значений, стоит пойти для того, чтобы сократить общее количество интервалов, на которые разбивается последовательность, и тем самым суммарные накладные расходы, связанные с запоминанием заголовков интервалов.

Для рассматриваемого представления упакованных данных алгоритм распаковки выглядит чрезвычайно просто.

```
1: Прочитано_значений ← 0
2: while Прочитано_значений < Упаковано_значений do
3:     (Глубина, Количество) ← ПрочитатьЗаголовокИнтервала()
4:     for i ← 1, Количество do
5:         ДобавитьЗначение(ПрочитатьЦелое(Глубина))
6:     end for
7:     Прочитано_значений ← Прочитано_значений + Количество
8: end while
```

Таким образом, декодирование значения алгоритмом распаковки выражается в простом считывании из битового потока знакового целого числа некоторой глубины. Декодирование отдельного значения выполняется за постоянное время, а общее время распаковки линейно по объёму данных.

7. Поиск оптимального разбиения при сжатии

Информацию о глубине значений также можно рассматривать как последовательность. Идея рассматриваемого алгоритма основана на том, что последовательность значений и последовательность глубин не независимы и в некоторых случаях можно увеличить глубину значения сверх необходимой ради того, чтобы последовательность глубин стала проще (рис. 1). Алгоритмы сжатия целочисленных последовательностей находят близкое к оптимальному разбиение последовательности на интервалы, минимизирующее общий объём упакованных данных: накладные расходы (объём заголовков интервалов) плюс объём данных, используемых для представления в интервалах самих значений элементов последовательности.

Для поиска оптимального разбиения алгоритм сжатия использует метод динамического программирования. Для построения схемы применения динамического программирования необходимо выделить такую информацию, при наличии которой для предыдущего элемента последовательности аналогичную информацию можно найти и для следующего элемента, а по информации для последнего элемента — и само оптимальное разбиение.

Обозначим:

- $P(S)$ — разбиение последовательности S на интервалы;
- $L(P(S))$ — стоимость разбиения $P(S)$ (объём упакованных данных при разбиении $P(S)$);
- $S_{\leq i}$ — начальная подпоследовательность S, заканчивающаяся элементом S_i.

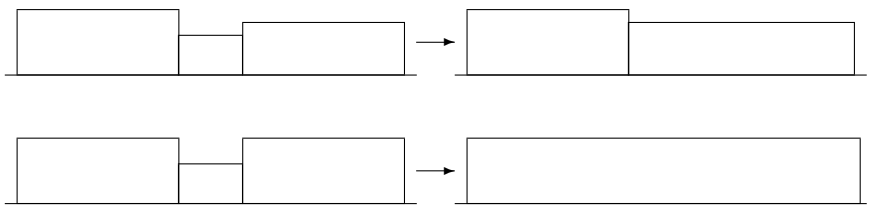

Рис. 1. Выигрыш от увеличения глубины интервала для объединения с соседними

Алгоритмы сжатия, которые строятся по рассматриваемому в данной работе принципу, могут различаться способом кодирования заголовка интервала. Пусть $h(d, l)$ означает стоимость записи заголовка интервала с глубиной d и количеством элементов l при выбранном способе кодирования заголовка (т. е. количество битов, необходимое для его представления).

При реализации алгоритма сжатия более эффективно запоминать информацию не об отдельных значениях, а об интервалах значений с постоянной минимальной глубиной ($L_s(S_i) = \text{const}$). Для хранения промежуточной информации о таких интервалах (длина, минимальная глубина, стоимость наилучшего разбиения, начало последнего интервала при наилучшем разбиении) используется вспомогательный буфер интервалов.

При выделении памяти сразу под весь буфер интервалов, достаточный для хранения информации обо всей последовательности, его объём может в несколько раз превысить объём сжимаемых данных. Если такие требования по памяти неприемлемы, то в случае переполнения буфера может выполняться сброс буфера: принудительное разбиение последовательности по границе последнего обработанного интервала, запись уже обработанной части согласно оптимальному к этому моменту разбиению, очистка буфера интервалов и обработка оставшейся части последовательности. Сброс буфера интервалов означает нарушение условия полной оптимальности найденного разбиения (если только не принять дополнительных мер для поиска безопасного места деления буфера при сбросе — см. далее).

8. Алгоритм VSEncoding

Рассмотрим алгоритм VSEncoding, предложенный в работе [4]. Поскольку для дальнейшего изложения необходимо знакомство читателя с этим алгоритмом, приведём его псевдокод с использованием применяемых в данной работе обозначений.

```
1:  C[0] ← 0
2:  for i ← 1, n do
3:      len ← 1                                          ▷ Длина интервала
4:      D ← L(S[i])                                      ▷ Глубина интервала
5:      C* ← C[i − 1] + h(D, 1) + D                      ▷ Минимальная найденная стоимость
6:      P* ← i − 1                                       ▷ Соответствующее разбиение начальной части
7:      for j ← i − 1, max(0, i − maxK) step −1 do
8:          L ← L + 1
9:          D ← max(D, L(S[j]))                          ▷ Пересчёт глубины интервала
10:         C_cur ← C[j − 1] + h(D, len) + D ∗ len        ▷ Стоимость проверяемого разбиения
11:         if C_cur < C* then                           ▷ Найден более хороший вариант
12:             C* ← C_cur                               ▷ Запомним его стоимость
13:             P* ← j                                   ▷ и место начала последнего интервала
14:         end if
15:     end for
16:     C[i] ← C*                                        ▷ Запомним в буфер стоимость наилучшего разбиения
17:     p[i] ← P*                                        ▷ и место начала его последнего интервала
18: end for
```

Алгоритм VSEncoding выполняет поиск близкого к оптимальному разбиения $P(S)$ исходной последовательности целых чисел S методом динамического программирования. Помимо S параметрами алгоритма являются способ кодирования заголовков интервалов и целое число $maxK$. Способ кодирования заголовков характеризуется функцией $h(d, l)$ стоимости записи заголовка интервала с глубиной d и количеством элементов l (т. е. его размером в битах). Число $maxK$ ограничивает глубину поиска. Пусть n — длина последовательности S. При работе алгоритма будем использовать два массива длины $n + 1$: $C[0..n]$ — стоимостей оптимальных разбиений и $p[0..n]$ — номеров предыдущих элементов разбиения. При этом $C[i]$ содержит стоимость наилучшего разбиения $S_{\leq i}$, а $p[i]$ — номер элемента, после которого начинается последний интервал этого наилучшего разбиения, дающего стоимость $C[i]$.

Таким образом, чтобы найти лучшее разбиение последовательности на интервалы, на каждом шаге метода динамического программирования выполняется сравнение стоимостей разбиений с разными длинами последнего интервала и наилучшим разбиением оставшейся начальной части последовательности (которое уже к этому моменту известно). При поиске длина последнего интервала ограничена значением параметра $maxK$.

В оригинальном алгоритме VSEncoding рассматривалась функция $h(d, l)$ вида $h(d, l) = M_1(b) + M_2(l)$, т. е. с независимым кодированием глубины и длины интервала. В данной работе мы избавимся от таких ограничений, что позволяет, например, выбирать разные кодировки длины для интервалов разной глубины, чтобы учесть то, что бóльшие по модулю значения встречаются в разностных последовательностях реже и образуют более короткие интервалы.

9. Поиск оптимального разбиения последовательности

Использование параметра $maxK$ в алгоритме VSEncoding позволяет так ограничить поиск, чтобы он мог быть выполнен за разумное время. При длине последовательности $n >> maxK$ вычислительная сложность такого алгоритма составит $O(n \cdot maxK)$. Если же взять $maxK \geq n$, то получим сложность $O(n^2)$, что вряд ли будет приемлемо при больших n. Сами авторы алгоритма VSEncoding используют $maxK$, не превосходящий 64, а также кодировку длины интервала M_2, позволяющую представить интервалы с длиной не более 64. Однако в разностных последовательностях могут появляться интервалы, например из нулевых значений, гораздо большей длины, поэтому искусственное ограничение длины интервала может ухудшить сжатие.

С другой стороны, попытка рассмотреть интервал слишком большой длины в качестве окончания оптимального разбиения в подавляющем большинстве случаев обречена на неудачу, поскольку высока вероятность того, что на таком интервале встретится достаточно большое значение, появление которого существенно повысит стоимость кодирования всех значений длинного интервала.

Для того чтобы найти истинно (а не приближённо) оптимальное разбиение последовательности, необходимо предложить критерий остановки поиска, который должен позволить определить, что дальнейшая проверка интервалов большей длины уже не имеет смысла, поскольку она может дать только худшие результаты. Так, если количество битов в кодировке значений последнего интервала $MaxD \cdot l$ превзойдет стоимость наилучшего найденного варианта разбиения C', то можно прекращать поиск, поскольку любое дальнейшее увеличение длины интервала увеличит стоимость его кодирования не менее чем на $MaxD$ битов. Конечно, приведённый пример критерия остановки по-

иска слишком грубый, но он показывает, что такие критерии в принципе существуют, поэтому попытаемся найти более точную оценку.

Обозначим

$$L_{\max}(j,i) = \max_{j<k\le i} L(S(k)).$$

Заметим, что для $j \le k \le j$

$$L_{\max}(j,i) = \max_{j<l\le i} L(S(l)) = \max(\max_{j<l\le k} L(S(l)), \max_{k<l\le i} L(S(l))) = \\ = \max(L_{\max}(j,k), L_{\max}(k,i)), \tag{1}$$

также отсюда следует, что для $j \le k \le j$

$$L_{\max}(j,i) \ge L_{\max}(j,k) \quad \text{и} \quad L_{\max}(j,i) \ge L_{\max}(k,i). \tag{2}$$

Стоимость оптимального разбиения $S_{\le i}$ можно выразить следующим образом:

$$C_i = \min_{0\le j<i} C_j + h(L_{\max}(j,i), i-j) + (i-j)\cdot L_{\max}(j,i).$$

Пусть в процессе поиска C_i найдено приближение C'. При этом уже пройдена точка разбиения k, поэтому $C' \le C_k + h(L_{\max}(k,i), i-k) + (i-k)\cdot L_{\max}(k,i)$.

Рассмотрим некоторую следующую за k (в порядке уменьшения индекса) точку разбиения j, т. е. $0 \le j < k$. Поскольку C_k соответствует оптимальному разбиению $S_{\le k}$, при его поиске был рассмотрен вариант с прохождением разбиения через j, поэтому

$$C_k \le C_j + h(L_{\max}(j,k), k-j) + (k-j)\cdot L_{\max}(j,k).$$

Отсюда можно получить оценку варианта разбиения с интервалом от j до i

$$C_k + (i-k)\cdot L_{\max}(k,i) \le$$

$$\le C_j + h(L_{\max}(j,k), k-j) + (k-j)\cdot L_{\max}(j,k) + (i-k)\cdot L_{\max}(k,i).$$

С учётом неравенств (2) для $L_{\max}$ получаем

$$C_k + (i-k)\cdot L_{\max}(k,i) \le C_j + h(L_{\max}(j,k), k-j) + (i-j)\cdot L_{\max}(j,i). \tag{3}$$

Сначала для упрощения изложения сделаем достаточно обоснованное предположение, что функция $h(d,l)$ является неубывающей по обоим аргументам, т. е. размер заголовка не может уменьшаться с ростом как длины l, так и глубины d (далее будет рассмотрен случай произвольной функции h). При монотонности $h(d,l)$ можно заменить в правой части (3) с сохранением неравенства $h(L_{\max}(j,k), k-j)$ на $h(L_{\max}(j,i), i-j)$:

$$C_j + h(L_{\max}(j,k), k-j) + (i-j)\cdot L_{\max}(j,i) \le$$

$$\le C_j + h(L_{\max}(j,i), i-j) + (i-j)\cdot L_{\max}(j,i).$$

Сравнивая полученное неравенство с (3), находим оценку снизу для стоимостей всех разбиений, заканчивающихся интервалом $j+1..i$ при $0 \le j < k$:

$$C_k + (i-k)\cdot L_{\max}(k,i) \le C_j + h(L_{\max}(j,i), i-j) + (i-j)\cdot L_{\max}(j,i).$$

Таким образом, если эта величина окажется больше C', то поиск можно заканчивать. Искомый критерий бессмысленности продолжения дальнейшего поиска после проверки варианта k имеет вид

$$\boxed{C' \le C_k + (i-k)\cdot L_{\max}(k,i)} \tag{4}$$

Заметим, что проверяемое выражение отличается от выражения для стоимости варианта k

$$C_k + h(L_{\max}(k,i), i-k) + (i-k)\cdot L_{\max}(k,i)$$

одним слагаемым — стоимостью заголовка, поэтому, чтобы искать оптимальное разбиение, требуется сделать минимальные изменения в коде алгоритма.

Вернемся к рассмотрению произвольной функции $h(d,l)$, которая может оказаться немонотонной. В этом случае по h найдем вспомогательную величину

$$dH = \max_{d,\ l,\ l_1\ge l,\ d_1\ge d} h(d,l) - h(d_1,l_1). \tag{5}$$

Для неубывающей h выполняется $dH = 0$, поскольку все разности неположительные (0 достигается при $d = d_1$ и $l = l_1$). В общем случае для используемого в правой части (3) подвыражения получим

$$h(L_{\max}(j,k), k-j) \le h(L_{\max}(j,i), i-j) + dH.$$

Отсюда для само́й правой части (3)

$$C_j + h(L_{\max}(j,k), k-j) + (i-j)\cdot L_{\max}(j,i) \le$$

$$\le C_j + h(L_{\max}(j,i), i-j) + dH + (i-j)\cdot L_{\max}(j,i).$$

В результате находим оценку снизу для стоимостей всех разбиений, заканчивающихся интервалом $j+1..i$ при $0 \le j < k$,

$$C_j + h(L_{\max}(j,i), i-j) + (i-j)\cdot L_{\max}(j,i) \ge C_k + (i-k)\cdot L_{\max}(k,i) - dH$$

и критерий остановки поиска после проверки варианта k

$$\boxed{C' + dH \le C_k + (i-k)\cdot L_{\max}(k,i)} \tag{6}$$

Таким образом, при немонотонной h приходится компенсировать эту немонотонность в оценке, прибавляя к порогу стоимости некоторую зависящую от h неотрицательную величину dH. При этом в ходе поиска придется зайти немного дальше, чтобы убедиться в бесперспективности его продолжения, поскольку появляется шанс, что с увеличением длины или глубины интервала размер его заголовка уменьшится. Однако размер, занимаемый значениями интервала, всё равно увеличится, поэтому поиск не продлится слишком долго.

Приведём результирующий алгоритм поиска оптимального разбиения (далее будем его называть VSEopt).

```
 1: C[0] ← 0
 2: for i ← 1, n do
 3:     len ← 1                                   ▷ Длина интервала
 4:     D ← L(S[i])                               ▷ Глубина интервала
 5:     C* ← C[i − 1] + h(D, 1) + D               ▷ Минимальная найденная стоимость
 6:     P* ← i − 1                                ▷ Соответствующее разбиение начальной части
 7:     for j ← i − 1, 0 step −1 do
 8:         len ← len + 1
 9:         D ← max(D, L(S[j]))                   ▷ Пересчёт глубины интервала
10:         C_cur ← C[j − 1] + D ∗ len            ▷ Значение для критерия остановки
11:         if C_cur ≥ C* + dH then
12:             break                             ▷ Выход по критерию остановки поиска
13:         end if
14:         C_cur ← C_cur +h(D, len)              ▷ Стоимость проверяемого разбиения
15:         if C_cur < C* then                    ▷ Найден более хороший вариант
16:             C* ← C_cur                        ▷ Запомним его стоимость
17:             P* ← j                            ▷ и место начала последнего интервала
18:         end if
19:     end for
20:     C[i] ← C*                                 ▷ Запомним в буфер стоимость наилучшего разбиения
21:     p[i] ← P*                                 ▷ и место начала его последнего интервала
22: end for
```

Измененные по сравнению с алгоритмом поиска при ограничении $maxK$ фрагменты показаны на сером фоне.

10. Поиск оптимального разбиения при ограничении на размер буфера

При сжатии разностных последовательностей используется буфер, в котором для каждого значения запоминается информация о необходимом для его представления количестве битов $L(S[i])$, стоимости найденного оптимального разбиения $C[i]$ и точке начала последнего интервала $p[i]$. При выделении памяти сразу под весь буфер интервалов, достаточный для хранения информации обо всей последовательности, его объём может в несколько раз превысить объём сжимаемых данных. Если такие требования по памяти неприемлемы, то в случае переполнения буфера может выполняться его сброс. Понятно, что сброс буфера интервалов может означать нарушение полной оптимальности найденного разбиения.

Для сохранения оптимальности получаемого разбиения при ограниченном буфере можно попытаться найти обязательный префикс всех оптимальных разбиений — точку согласования разбиений, т. е. максимальный из таких индексов элементов последовательности, что все разбиения, которые могут быть выбраны в качестве оптимальных для любых возможных продолжений последовательности, будут содержать деление на

интервалы в этом месте. В качестве такой точки согласования может, например, выступать "выброс" — значение, кодируемое существенно бóльшим числом битов, чем его соседи: любое оптимальное разбиение должно учитывать такой выброс, выделив его в отдельный интервал.

Для поиска точки согласования k_{con} требуется сначала найти минимальную точку остановки перебора k^* — такой индекс, за который критерий остановки поиска (6) не должен пропустить цикл поиска оптимального разбиения для любого продолжения последовательности. Будем использовать следующие индексы: i — номер элемента последовательности, на котором был достигнут конец буфера; $l > i$ — номер произвольного следующего элемента последовательности.

Поиск оптимального разбиения для большинства $l > i$ (достаточно далеких от i) может закончиться ещё до i, но для не слишком больших l будет сделана попытка использовать разбиение с последним интервалом, начинающимся перед i. В первом случае (для больших l) префиксом оптимального разбиения будет одно из разбиений, полученных для меньших l, поэтому достаточно рассмотреть второй случай, для которого согласно найденному в предыдущей главе критерию остановки (6) при достижении k^* должно выполняться условие

$$C'_l + dH \le C_{k^*} + (l - k^*) \cdot L_{\max}(k^*, l) \quad \forall l > i. \tag{7}$$

Здесь C'_l — стоимость наилучшего известного к рассматриваемому моменту разбиения $P_{\le l}$.

При поиске оптимального разбиения для l должно быть рассмотрено и разбиение, проходящее через i, поэтому

$$C'_l \le C_i + h(L_{\max}(i, l), l - i) + (l - i) \cdot L_{\max}(i, l). \tag{8}$$

Для выполнения (7) достаточно, чтобы выполнялось условие

$$C_i + h(L_{\max}(i, l), l - i) + \underline{(l - i) \cdot L_{\max}(i, l)} + dH \le C_{k^*} + (l - k^*) \cdot L_{\max}(k^*, l) \quad \forall l > i. \tag{9}$$

С учётом неравенств (2) для $L_{\max}$ можно оценить снизу правую часть (9), разбив последний интервал $k^*..l$ на две части в точке i:

$$C_{k^*} + (l - k^*) \cdot L_{\max}(k^*, l) \ge C_{k^*} + (i - k^*) \cdot L_{\max}(k^*, i) + \underline{(l - i) \cdot L_{\max}(l, i)}.$$

Поэтому неравенство (9) будет выполнено при

$$C_i + h(L_{\max}(i, l), l - i) + dH \le C_{k^*} + (i - k^*) \cdot L_{\max}(k^*, i) \quad \forall l > i \tag{10}$$

(подчеркнутые фрагменты сокращаются).

Обозначим через $H_{\max}$ максимальный размер заголовка интервала. Эту вспомогательную величину так же, как и dH, можно найти по функции h:

$$H_{\max} = \max_{d,\ l} h(d, l). \tag{11}$$

Заменяя в (10) $h(L_{\max}(i, l), l - i)$ на $H_{\max}$, слегка усиливаем неравенство, избавляемся от зависимости условия от l и получаем окончательное условие на k^*:

$$\boxed{C_i + H_{\max} + dH \le C_{k^*} + (i - k^*) \cdot L_{\max}(k^*, i)} \tag{12}$$

Полученное условие отличается от критерия остановки поиска (6) для i слагаемым $H_{\max}$ в левой части, поэтому поиск k^* может зайти несколько дальше, чем поиск оптимального разбиения для $P_{\leq i}$.

Приведём алгоритм поиска минимальной точки остановки перебора k^*.

```
1: k* ← 0                                  ▷ Искомая точка остановки перебора (пока не найдена)
2: len ← 1                                 ▷ Длина интервала
3: D ← 0                                   ▷ Глубина интервала
4: Lim ← C_i + H_max + dH                  ▷ Предел — стоимость, которую надо превзойти
5: for j ← i, i/2 step −1 do               ▷ Ограничиваем поиск последней половиной буфера
6:     len ← len + 1
7:     D ← max(D, L(S[j]))                 ▷ Пересчёт глубины интервала
8:     if C[j − 1] + D * len ≥ Lim then    ▷ Предельная стоимость достигнута
9:         k* ← j                          ▷ Точка остановки перебора найдена
10:        break
11:    end if
12: end for
```

В данном алгоритме используется эвристика для определения возможного предела поиска точки остановки перебора: здесь поиск ограничивается последней половиной буфера. В случае неудачи поиска k^* придётся отказаться от мысли о нахождении истинно оптимального разбиения, сбросить весь буфер и ограничиться нахождением лишь близкого к оптимальному результата. Уменьшение предельного индекса точки остановки перебора может повысить шансы её нахождения. Однако в случае, если такая точка будет найдена где-то в начале буфера, придётся сбросить лишь малую его часть, а частое повторение такой ситуации может привести к существенному замедлению работы алгоритма. Впрочем, вычислительный эксперимент на реальных данных большого объёма показывает, что k^* обычно находится совсем не далеко от конца буфера.

Если минимальную точку остановки перебора $k^* > 0$ удаётся найти, то поиск оптимального разбиения $\forall l > i$ закончится до k^*. Поэтому оптимальное разбиение $\forall l > i$ пройдёт через одну из точек $k^*+1 \,..\, i$. Отсюда следует, что точка k_{con} согласования всех оптимальных разбиений $\forall l > i$ на самом деле является точкой согласования конечного числа оптимальных разбиений, соответствующих $C[j]$ и $p[j]$ $\forall j : k^* < j \leq i$. Для поиска точки согласования добавим вспомогательный флаг $Used[j]$ в элементы буфера. Если флаг установлен, это означает, что по границе j проходит одно из потенциально используемых разбиений. Если же флаг $Used[j]$ не будет установлен к моменту рассмотрения позиции j, то на данную границу не вышло ни одно из ранее рассмотренных используемых разбиений, поэтому точку j можно пропустить. Перед началом поиска задаем нижнюю границу k^* и устанавливаем флаг $Used[j]$ у всех элементов буфера в интервале $k^* + 1..i$ — все они могут быть использованы при построении оптимальных разбиений $\forall l > i$. Далее при обнаружении $p[j] < k^*$ уменьшаем границу поиска, устанавливая флаг $Used[p[j]]$ и сбрасывая его для остальных элементов в диапазоне $p[j] + 1..k^*$.

Из приведённого ниже алгоритма поиска точки согласования k_{con} видно, что работа алгоритма заканчивается в случае достижения циклом поиска точки согласования k_{con}, поскольку это означает, что ни одна из рассмотренных последовательностей интервалов не “перепрыгнула” через неё.

```
1: k_con ← k*                          ▷ Устанавливаем начальное значение точки согласования
2: for j ← k* + 1, i do                ▷ Устанавливаем флаги использования для разбиений до k*
3:     Used[j] ← true
4: end for
5: for j ← i, 1 step −1 do                                        ▷ Поиск точки согласования
6:     if Used[j] then
7:         if j = k_con then
8:             break                   ▷ Найдена точка согласования, через которую прошли все разбиения
9:         end if
10:        if p[j] < k_con then        ▷ Используется разбиение за пределами зоны поиска
11:            for k ← p[j] + 1, k_con do      ▷ Флаги для этой части не были заданы
12:                Used[k] ← false     ▷ Пометим как пока не используемые все добавляемые в рассмотрение точки
13:            end for
14:            k_con ← p[j]                                   ▷ Расширим зону поиска
15:        end if
16:        Used[p[j]] ← true
17:    end if
18: end for
```

При обнаружении точки согласования выполняется сохранение начального фрагмента буфера до этой точки включительно со сдвигом в начало оставшейся части буфера. Экспериментальная проверка предложенного алгоритма на реальных данных показала, что он действительно позволяет найти оптимальное разбиение при использовании ограниченного буфера (выполнялось сравнение с оптимальными результатами, полученными без ограничения объёма). При этом размеры перемещаемых в начало буфера фрагментов оказываются небольшими (в среднем менее 100 элементов).

11. Конкретизация способа кодирования заголовков интервалов

Рассмотренный в данной работе метод сжатия определяет схему построения алгоритмов. Конкретный алгоритм будет получен при задании определённого способа кодирования заголовков интервалов. При выборе способа кодирования заголовков интервалов должны учитываться битовая глубина сжимаемой последовательности и распределение длин интервалов постоянной глубины. Так, в большинстве выполненных вычислительных экспериментах было использовано кодирование длины интервала блоками по $k = 3$ битов с дополнительным 1 битом — признаком продолжения, при котором

$$h_{gr\,k}(d, l) = 4 + (k + 1) \cdot \lceil \log_2(l)/k \rceil,$$

где первые 4 бита расходуются на глубину интервала, а далее на каждые 3 бита в представлении длины интервала тратится ещё по 4 бита (далее будем называть такое кодирование split-n). Выбор разбиения на группы по 3 бита основан на результатах вычислительного эксперимента с конкретными данными, в котором для такого размера групп битов длины получен лучший результат.

Однако впоследствии был принят во внимание тот факт, что при этом допускается избыточность вариантов кодирования одного и того же значения длины. Поскольку

для кодирования длины потребовалось несколько интервалов, это значение не может быть представлено меньшим количеством интервалов. В результате для кодирования длины интервалов были использованы старт-шаг-стоп-коды [2] (с совпадением длин начальной и последующих групп битов — далее будем называть такое кодирование step-n). При использовании step-n значения, кодируемые n интервалами, начинаются сразу после всех значений, представимых меньшим числом интервалов. В этом случае при использовании групп по k битов получаем немного меньший размер заголовка:

$$h_{s\,k}(d,l) = 4 + (k+1) \cdot \lceil \log_2(l * (1 - 1/2^k) + 1)/k \rceil.$$

Несмотря на незначительное сокращение длины интервала при использовании старт-шаг-стоп-кодов, вычислительный эксперимент показал, что для них лучшее сжатие получается при $k = 2$. Именно этот вариант кодирования заголовков будет далее использоваться в большинстве вычислительных экспериментов.

Другой проверенный экспериментально способ кодирования заголовков интервалов основан на использовании статических кодов Хаффмана для кодирования длины и/или глубины интервала. При этом дерево Хаффмана строится на основе статистики разбиения на интервалы, полученной при предварительно выполненном сжатии данных с помощью другого способа кодирования заголовков, не требующего знания статистики. Здесь возникает циклическая зависимость: статистика частоты использования интервалов с различными значениями длины и глубины определяет способ кодирования заголовков интервалов, а способ кодирования заголовков — статистику использования интервалов. Выполнение нескольких итераций сжатия для уточнения статистики позволяет ещё сократить размер файла.

Исследованы три варианта кодирования заголовков с использованием кодов Хаффмана:

- L — общая кодировка длин интервалов независимо от глубины, глубина кодируется фиксированным числом битов;
- L_D — отдельная кодировка длин интервалов для каждого значения глубины, глубина кодируется фиксированным числом битов;
- $L_D D$ — отдельная кодировка длин интервалов для каждого значения глубины, глубина также кодируется по Хаффману.

Во всех вариантах коды Хаффмана использовались для кодирования количества битов длины, т. е. величины $n = L_u(L-1)$ (кодируем $L-1$, поскольку интервалов длины 0 не бывает), за этим кодом при $L > 2$ необходимо ещё записать $n-1$ младший бит в двоичном представлении самого числа $L-1$; старший бит всегда является единицей для ненулевых значений, поэтому его не храним. В варианте $L_D D$ коды Хаффмана используются непосредственно для представления глубины интервала. Во всех случаях деревья кодов Хаффмана строились для всего файла и записывались в его заголовок.

Все многообразие возможных способов кодирования заголовков интервалов не исчерпывается выполненными экспериментами и нуждается в дальнейших исследованиях.

12. Тестирование алгоритмов сжатия разностных последовательностей

Алгоритм BCRL разработан при реализации иерархического растрового формата MRG, позволяющего эффективно работать с растрами большого объёма (впоследствии в этом

качестве применялись алгоритмы семейства VSEopt). Первоначальной целью создания формата MRG было получение слитного иерархического растра высот земной поверхности по данным SRTM (Shuttle Radar Topography Mission [11]), которые распространяются блоками 1° по широте ×1° по долготе с шагом 3″. При представлении пирамиды разрешений в формате MRG для более детальных блоков сохраняются разности между реальными значениями и результатами интерполяции этих значений по данным менее детальных блоков, кроме того, могут использоваться алгоритмы декорреляции значений. Таким образом, сохраняемые в формате MRG данные представляют собой разностные последовательности в том смысле, в каком они рассматриваются в этой статье. Впоследствии формат MRG был доработан для хранения произвольных растров высот, а также многоканальных растровых изображений большого объёма.

Первоначально разностные данные сжимались при помощи ZLib, но в дальнейшем возникла идея использовать алгоритм, способный получить более высокую степень сжатия путем учёта природы разностных данных, а также ускорить чтение блоков за счёт использования простого формата хранения упакованных значений.

Для оценки производительности алгоритмов сжатия разностных последовательностей при кодировании разностных растров выполнено их тестирование на 110 блоках SRTM, описывающих территорию вокруг Байкала. Для сравнения была использована упаковка тех же разностных последовательностей средствами библиотеки ZLib с различными значениями уровня сжатия.

Используемый фрагмент данных, с одной стороны, не слишком велик по сравнению со всем массивом данных SRTM, что позволяет выполнить над ним достаточно большое число вычислительных экспериментов. С другой стороны, объём обрабатываемых при этом данных достаточно велик (300 МБ в неупакованном виде, более 150 миллионов значений), поэтому получаемые для него результаты являются представительными и применимыми к другим данным.

В табл. 1 содержатся основные результаты проведённого тестирования алгоритмов. В первых двух строках приведена информация об исходных данных, остальные строки относятся к файлам MRG, полученным из этих данных с использованием различных алгоритмов сжатия (ZLib и VSEopt с различными вариантами кодирования заголовков интервалов). Отметим, что файлы SRTM распространяются в архивах ZIP, которые созданы со степенью сжатия, превышающей максимальную для имеющейся у автора

Т а б л и ц а 1. Результаты тестирования алгоритмов сжатия на 110 блоках SRTM(50 – 60° с.ш.) × (100 – 109° в.д.)

Способ сжатия	Размер, байт	Коэф. сжатия, %	% от ZLib max	Время, с	Время, мин
Исходные данные без сжатия	317 328 220	100.00	—	—	—
Исходные данные в Zip	113 427 015	35.74	—	—	—
MRG с ZLib fastest	88 752 112	27.97	—	13.29	0:00:13
MRG с ZLib default	78 999 264	24.90	—	50.065	0:00:50
MRG с ZLib max	76 738 672	24.18	100	541.79	0:09:01
MRG с VSEopt, split-3	66 662 320	21.01	86.87	22.97	0:00:22
MRG с VSEopt, step-2	66 409 088	20.93	86.54	22.509	0:00:22
MRG с VSEopt, HT L_DD	64 207 328	20.23	83.67	36.676	0:00:36
MRG с VSEopt, HT L_DD I5	63 905 280	20.14	83.28	40.081	0:00:40

версии архиватора PKZIP. Например, архив N50E100.hgt.zip имеет размер 1 376 212 Б, а после упаковки его содержимого при помощи архиватора PKZIPC получаем 1 418 255 Б. Этот же файл после максимального сжатия средствами библиотеки ZLib, используемой для сравнения, занимает 1 430 423 Б. Таким образом, исходные данные были сжаты очень хорошей версией архиватора ZIP.

Тем не менее при помощи разностного кодирования блоков эти результаты превосходит даже ZLib с минимальным для библиотеки уровнем сжатия Fastest. Также видно, что алгоритм сжатия разностных последовательностей смог улучшить результаты ZLib с максимальным уровнем сжатия ещё на 13–16 % от объёма после сжатия при помощи ZLib. Конкретные цифры определяются способом кодирования заголовков интервалов — далее будут более подробно рассмотрены результаты сравнения различных представлений заголовков.

По времени работы алгоритм VSEopt уступает только результатам ZLib с уровнем сжатия fastest, однако по степени сжатия BCRL существенно превосходит ZLib в этом режиме (на 25–28 % от объёма упакованных ZLib данных).

Теперь выполним сравнение различных способов кодирования заголовка. В табл. 2 приведены результаты тестирования на тех же данных SRTM представлений заголовков split-n и step-n при различных значениях n, которые обосновывают выбор split-3 и step-2 для проведения тестов. Во всех тестах использовался алгоритм сжатия VSEopt. Видно, что метод step-2 дает наилучшее сжатие среди рассмотренных, не зависящих от конкретных данных методов, поэтому он использован для проведения большинства тестов.

Рассмотрим, какое преимущество может быть получено за счёт использования VSEopt по сравнению с VSEncoding при различных значениях параметра $maxK$, ограничивающего глубину перебора. В табл. 3 приведены результаты тестирования на тех же данных SRTM при кодировке заголовков step-2. Из таблицы видно, что при $maxK = 64$, действительно, достигается неплохое сжатие. С другой стороны, даже при $maxK = 1024$ оптимальное разбиение не находится. При этом по времени работы на рассматриваемых данных с кодировкой заголовка step-2 алгоритм VSEopt сравним с VSEncoding при $maxK = 16$. Таким образом, использование критерия остановки поиска оказывается очень выгодным: оно позволяет не только найти оптимальное разбиение, но и сократить время работы даже по сравнению с достаточно небольшими значениями $maxK$.

Сжатие может быть улучшено за счёт применения методов кодирования заголовков интервалов, учитывающих особенности конкретных данных. Для этих целей реализованы три варианта представления заголовков с использованием кодов Хаффмана: L, L_D,

Таблица 2. Сравнение методов кодирования заголовков split-n и step-n по результатам работы на 110 блоках SRTM

n	split-n		step-n	
	Размер, байт	Коэф. сжатия, %	Размер, байт	Коэф. сжатия, %
1	66 557 008	20.97	67 744 096	21.35
2	66 409 104	20.93	66 690 512	21.02
3	66 630 368	21.00	66 662 336	21.01
4	66 780 672	21.04	66 781 264	21.04
5	66 832 272	21.06	66 832 304	21.06

Т а б л и ц а 3. Сравнение методов сжатия VSEopt и VSEncoding по результатам работы на 110 блоках SRTM

$maxK$	Размер, байт	Коэф. сжатия, %	Время, с	Время, мин
8	70 798 896	22.31	17.598	0:00:17
16	67 913 120	21.40	22.290	0:00:22
24	67 273 056	21.20	26.540	0:00:26
32	66 890 384	21.08	31.010	0:00:31
40	66 683 360	21.01	35.354	0:00:35
48	66 567 520	20.98	40.128	0:00:40
56	66 505 184	20.96	44.056	0:00:44
64	66 476 416	20.95	48.640	0:00:48
72	66 462 960	20,94	53.074	0:00:53
80	66 451 952	20.94	57.659	0:00:57
88	66 443 296	20.94	62.667	0:01:02
96	66 436 432	20.94	66.024	0:01:06
128	66 420 320	20.93	84.077	0:01:24
256	66 410 096	20.93	154.367	0:02:34
512	66 409 744	20.93	292.853	0:04:52
1024	66 409 696	20.93	561.020	0:09:21
opt	66 409 088	20.93	22.509	0:00:22

Т а б л и ц а 4. Сравнение методов кодирования заголовков с использованием кодов Хаффмана по результатам работы на 110 блоках SRTM

Кодировка	Размер, байт	Коэф. сжатия, %	Время, с	Время, мин
step-2	66 409 088	20.93	22.509	0:00:22
L	65 910 416	20.77	28.909	0:00:28
L_D	65 779 152	20.73	35.241	0:00:35
$L_D D$	64 207 328	20.23	36.676	0:00:36

$L_D D$. Результаты применения этих методов приведены в табл. 4. Сжатие выполнялось с помощью алгоритма VSEopt и статистики, полученной при использовании алгоритма VSEopt и кодировки заголовка step-2. По результатам тестирования видно, что наибольший выигрыш (0.7 % от объёма исходных данных по сравнению с кодировкой step-2) достигается при использовании кодировки $L_D D$.

Для наиболее эффективного варианта кодирования заголовков $L_D D$ исследуем возможность улучшения сжатия за счёт поиска кодов Хаффмана, соответствующих статистике получаемого при их использовании распределения интервалов. Для этого выполним несколько итераций, начиная со статистики для step-2 и продолжая использованием статистики для разбиений, полученных на предыдущей итерации, для выбора кодов, используемых на следующей. В табл. 5 показана динамика изменения размера файла по итерациям. Видно, что в ходе поиска размер файла монотонно уменьшается. Судя по совпадению размеров, можно предположить, что на 5-й итерации процесс сошелся. Это предположение подтверждается тем, что файлы статистики, полученные на 4-й и 5-й итерациях, совпадают.

Т а б л и ц а 5. Динамика изменения размера файла в ходе улучшения статистики распределения интервалов методом итераций при сжатии 110 блоков SRTM

№ итерации	Размер, байт	Коэф. сжатия, %	Время, с	Время, мин
0 (по step-2)	64 207 328	20.23	36.676	0:00:36
1	63 968 688	20.16	39.605	0:00:39
2	63 928 768	20.15	39.963	0:00:39
3	63 910 992	20.14	40.228	0:00:40
4	63 905 280	20.14	40.097	0:00:40
5	63 905 280	20.14	40.081	0:00:40

Не ясно, насколько типичен этот случай, т. е. насколько часто процесс поиска согласованного со статистикой представления заголовков методом итераций будет сходиться, — этот вопрос требует дальнейшего исследования. Сам метод поиска наилучших параметров сжатия посредством генерации нескольких вспомогательных вариантов файла также выглядит не бесспорно — необходимо еще предложить сценарий его применения, обосновывающий целесообразность таких трудозатрат (например, оптимальную для фрагмента данных статистику можно использовать при сжатии всех данных). Тем не менее именно этот метод позволил найти наиболее компактное представление рассматриваемых в работе данных, которое приводится в последней строке табл. 1.

Рассмотренные ранее представления заголовков split-n и step-n являются монотонными, чего нельзя сказать о представлениях с помощью кодов Хаффмана: здесь реже встречающимся меньшим значениям могут быть присвоены более длинные коды. Поэтому на примере представлений заголовков с использованием кодов Хаффмана можно проверить необходимость введения поправки dH в критерии остановки поиска (6). В табл. 6 приведены результаты такого тестирования. При этом вместо значения dH, вычисленного по формуле (5) (здесь будем обозначать его dH^*), в условии остановки поиска (6) использованы альтернативные значения. Во всех тестах применялась статистика для кодировки заголовков step-2, при этом получены ненулевые значения поправок для всех рассматриваемых вариантов кодирования заголовка, приведённые в столбце dH^*.

Тестирование показало, что поправка dH действительно необходима в (6) для достижения оптимального результата, поскольку при $dH = 0$ для всех вариантов получается результат хуже оптимального. Однако найденная по формуле (5) поправка либо оказывается слишком грубой, либо учитывает маловероятное сочетание интервалов, которое не реализуется на используемых для тестирования данных, поэтому уже при $dH = 1$ для кодировок L и L_D и при $dH = 2$ для кодировки L_DD находится оптимальное разбиение. Дополнительно проведено тестирование с $dH = dH^* + 1$, которое показало, что

Т а б л и ц а 6. Результаты тестирования алгоритма VSEopt с заменой вычисленного значения dH^* на альтернативные при сжатии 110 блоков SRTM с использованием статистики для step-2

Кодировка	dH^*	Размер, байт		
		$dH = 0$	$dH = 1$	$dH = 2..dH^* + 1$
L	2	65 912 560	65 910 416	
L_D	8	65 781 184	65 779 152	
L_DD	10	64 212 688	64 207 344	64 207 328

Т а б л и ц а 7. Результаты тестирования работы алгоритма VSEopt с ограничениями на размер буфера при сжатии 110 блоков SRTM с кодировкой заголовков step-2

Размер буфера	Число делений	Число ошибок	$i - k^*$			$i - k_{con}$			Увеличение размера буфера	t, с
			средн.	σ	max	средн.	σ	max		
256	513639	18407	49.09566	29.56092	192	66.39054	36.06395	255	17872	22.409
512	223039	1745	54.09093	44.03588	384	71.23359	49.63309	511	1680	22.345
768	141332	295	55.45633	51.03141	576	72.7885	56.38411	762	272	22.422
1024	103181	50	55.99229	54.35135	768	73.48568	59.57874	1002	32	22.328
1536	66823	2	55.9353	55.42749	1017	73.24717	60.78797	1041	16	22.507
2048	49343	—	55.7629	55.519	1523	73.35563	60.85825	1553	0	22.329
4096	23890	—	55.37472	53.15844	907	72.56769	58.95732	1181	0	22.307
16384	5359	—	54.3357	51.40663	763	72.57772	56.65134	772	0	22.327
65536	1137	—	77.14776	82.33685	758	96.97186	85.8242	758	0	22.209
262144	—	—	—	—	—	—	—	—	0	22.328

при этом не происходит дальнейшего улучшения, поскольку поправки dH^* достаточно для нахождения минимума.

Проверим работу алгоритма VSEopt при ограничении размера буфера с использованием алгоритма поиска точки согласования разбиений. Во всех ранее приводившихся тестах использован буфер достаточного для представления всех кодируемых данных размера (поскольку в файлах MRG изображение разбивается на блоки 400×400, требуется, чтобы размер буфера был не меньше 160 000 элементов).

В табл. 7 приведён размер буфера, который был использован при сжатии, число выполненных делений буфера (в связи с его переполнением), число ошибок — для какого числа из этих делений не удалось найти точку согласования разбиений k_{con} (в результате был сброшен весь буфер и получено неоптимальное разбиение). Приводятся статистические характеристики двух величин: $i - k^*$ (расстояние от конца буфера до точки разбиения) и $i - k_{con}$ (расстояние от конца буфера до точки согласования). Для каждой из величин показаны ее среднее значение, стандартное отклонение σ и максимальное значение (столбец "max"). Отображено увеличение размера буфера (в байтах), вызванное ошибками поиска точки согласования, относительно оптимального размера (66 409 088 байтов для используемой кодировки заголовка) и приводится время работы алгоритма сжатия.

По результатам выполненного тестирования видно, что использование буфера ограниченного размера не вызывает статистически значимого замедления работы алгоритма (время работы при разных размерах буфера колеблется в диапазоне погрешности измерений) и позволяет получить оптимальное разбиение в том случае, когда размер буфера превышает $\max(i - k_{con})$. Для рассматриваемого примера достаточный для нахождения оптимального разбиения размер буфера составил 2048 элементов.

Заключение

Рассмотренный принцип построения алгоритмов сжатия данных без потери информации ориентирован на работу с последовательностями целочисленных значений, распределённых преимущественно вблизи нуля. Для получения конкретного алгоритма сжа-

тия необходимо задать способ кодирования заголовков интервалов. Работа алгоритма направлена на снижение избыточности количества битов, используемых для представления целочисленных значений.

Особенностью алгоритмов сжатия разностных последовательностей является асимметрия по времени работы между этапами кодирования и декодирования: при всех вариантах алгоритма декодирование выполняется значительно быстрее и не вызывает существенного расхода памяти. Таким образом, потратив некоторое время на сжатие данных, мы получаем выигрыш по времени при каждом их чтении.

Основными результатами работы являются: алгоритм VSEopt (разработанный на основе алгоритма VSEncoding), позволяющий находить оптимальное сжатие разностных последовательностей; алгоритм поиска точки согласования разбиений, позволяющий находить оптимальное разбиение при использовании для хранения вспомогательной информации буфера ограниченного размера; тестирование конкретных алгоритмов сжатия, получаемых при выборе способа кодирования заголовков интервалов и, в частности, алгоритмов с использованием кодов Хаффмана, учитывающих статистические особенности обрабатываемых данных.

Имеется опыт использования рассматриваемых алгоритмов для сжатия без потерь растровых изображений и звуковых файлов, который показывает, что в сочетании с подходящим методом вычисления разностных значений алгоритм позволяет повысить степень сжатия по сравнению с популярными универсальными архиваторами.

Список литературы / References

[1] **Gailly, J.-L., Adler, M.** ZLib home site. Available at: http://www.zlib.net (accessed: 13.05.2015).

[2] Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. М.: ДИАЛОГ-МИФИ, 2003. 384 с.
Data compression methods. Construction of archivers, image and video compression / D. Vatolin, A. Ratushnyak, M. Smirnov, V. Yukin. Moscow: DIALOG-MIFI, 2003. 384 p. (in Russ.)

[3] **Хмельнов А.Е.** Формат файлов MRG для компактного представления и высокоскоростной декомпрессии матриц высот большого объёма // Вычисл. технологии. 2015. Т. 20, № 1. С. 63–74.
Hmelnov, A.E. The MRG file format for compact representation and fast decompression of large digital elevation models // Computational Technologies. 2015. Vol. 20, No. 1. P. 63–74. (in Russ.)

[4] **Silvestri, F., Venturini, R.** VSEncoding: efficient coding and fast decoding of integer lists via dynamic programming // Proceeding CIKM '10 Proceedings of the 19th ACM International Conference on Information and Knowledge Management. USA, New York, 2010. P. 1219–1228.

[5] **Calderbank, A.R., Daubechies, I., Sweldens, W., Yeo, B.-L.** Wavelet transforms that map integers to integers. Technical report, Department of Mathematics. USA: Princeton University, 1996. Available at:
ftp://ftp.ldv.ei.tum.de/pub/lossless/wavelet_transf_integers.pdf

[6] **Ватолин Д.** Все о сжатии данных, изображений и видео. Адрес доступа:
http://www.compression.ru (дата обращения: 13.05.2015).
Vatolin, D. All about data, image and video compression. Available at:
http://www.compression.ru (accessed: 13.05.2015). (in Russ.)

[7] **Сэломон Д.** Сжатие данных, изображений и звука. М.: Техносфера, 2004.
Salomon, D. A Guide to data compression methods. New York: Springer-Verlag, 2002.

[8] **Lemire, D., Boytsov, L.** Decoding billions of integers per second through vectorization // Software: Practice and Experience. 2015. Vol. 45, iss. 1. P. 1–29.

[9] **Deveaux, J.-P., Rau-Chaplin, A., Zeh, N.** Adaptive tuple differential coding // Database and Expert Systems Applications. Lecture Notes in Computer Science. 2007. Vol. 4653. P. 109–119.

[10] **Klimenko, S., Mitselmakher, G., Sazonov, A.** Losless compression of LIGO data. Technical Note LIGO-T000076- 00- D, MIT 07.08.2000. Available at: http://www.phys.ufl.edu/~klimenko/wavelets/lossless.pdf

[11] **Дубинин М.** Описание и получение данных SRTM. Адрес доступа: http://gis-lab.info/qa/srtm.html (дата обращения 13.05.2015).
Dubinin, M. The SRTM data description and sources. Available at: http://gis-lab.info/qa/srtm.html (accessed 13.05.2015). (in Russ.)

Поступила в редакцию 29 сентября 2014 г.,
с доработки — 25 мая 2015 г.

A lossless compression algorithm for integer difference sequences by optimization of their division into intervals of constant bit depth values

Hmelnov, Alexei E.

Institute for System Dynamics and Control Theory SB RAS, Irkutsk, 664033, Russia

Corresponding author: Hmelnov, Alexei E., e-mail: hmelnov@icc.ru

Purpose. A principle for construction of algorithms for fast lossless compression of integer data values, which are distributed mainly near zero, is considered. Such data come from, e. g., differential encoding of integer sequences, which represent gradually varying quantities (the quantities, which take similar values in neighboring points). According to the degree of compression for the given kind of data, the algorithms considered are of superior performance compared to ZLib [1] in its strongest compression mode Z_BEST_COMPRESSION, but they take considerably less time for both compression and unpacking, since the proposed compression algorithm is characterized by a linear computational complexity in terms of the processed data size.

Design/methodology/approach. To find the best (most compact) representation of an integer sequence by series of intervals for values of constant bit depth we use the dynamic programming approach. The dynamic programming scheme used is of the VSEncoding algorithm [4] type. A concrete compression algorithm is defined by specifying a particular method for the interval header representation.

Findings. The VSEopt algorithm — an optimized version of VSEncoding algorithm is proposed, which allows to find the true minimum of compressed size without limiting the depth of the interval length search by the $maxK$ parameter. A theoretical justification of the VSEopt algorithm is presented. The performed tests have demonstrated that VSEopt works as fast as VSEncoding with impractically small $maxK = 16$, but even with $maxK = 1024$ (and a lot of time spent) the VSEncoding algorithm can‘t achieve the optimal compression found by VSEopt.

Another theoretical inference allows to use for the VSEopt compression the buffer of limited size without losing the capability of finding the true optimal solution. The performed tests show that the algorithm still be capable to find the best solution even when the buffer may be as small as 2000 items (when compressing much larger blocks of 160 000 values). Some particular methods of interval header encoding were considered and tested. The best results were obtained for encodings, which use the static Huffman codes obtained with the help of statistics for interval depths and lengths distribution. For the test performed, the process of finding the header encoding which matches the statistics resulting from its usage have converged to the smallest of all the tests executed compressed data size.

Originality/value. The VSEncoding algorithm is the well known tool for compression of document indexes. The optimized version of the algorithm — VSEopt is a new result. Our formulation and solution addresses the problem how to use a buffer of limited capacity but still find the best compression. The methods used for the algorithms of compression of differential integer sequences are also new and extend the field of the algorithm usage.

Keywords: lossless compression, specialized data compression algorithm, dynamic programming.

Received 29 September 2014
Received in revised form 25 May 2015