УДК 519.72

В.С. Выхованец

Верхняя и нижняя оценки Колмогоровской сложности[1]

Аннотация. Рассматривается Колмогоровская сложность, определяемая как мера вычислительных ресурсов, необходимых для восстановления строк по их минимальным описаниям на некотором формальном языке. Несмотря на неразрешимость задачи определения Колмогоровской сложности строк в общей постановке задачи, могут быть найдены ее нижние и верхние оценки. Для получения оценок Колмогоровской сложности произвольная строка представляется как последовательность значений некоторой логической функции. После нахождения формулы этой функции строится программа, которая восстанавливает исходную строку путем циклического вычисления закодированной в ее теле формулы. Для минимизации длины программы последняя снова рассматривается как строка, подлежащая восстановлению. Процесс минимизации завершается, когда следующая строка программы становится длиннее предыдущей. На основе этого и известных оценок числа операций в формулах логических функций получены нижняя и верхняя оценки Колмогоровской сложности строк. Оказалось, что Колмогоровская сложность строк является одинаковой для всех строк с длиной, превышающей некоторую максимальную.

Ключевые слова и фразы: алгоритмическая сложность, Колмогоровская сложность, сжатие данных.

Введение

В алгоритмической теории информации Колмогоровская сложность объекта [1] есть мера информации, необходимой для точной его идентификации. Исходным описанием объекта служат данные, представленные в виде строк конечной длины над конечным алфавитом, а сложность строки – это длина описания этой строки на некотором формальном языке, также представленном в виде строки.

Считается, что Колмогоровская сложность $K(s)$ строки s не может быть значительно больше длины $|s|$ самой строки s, $K(s) \le |s| + c$, где c – некоторая константа. Это результат обосновывается существованием программы P, которая содержит строку s в неизменном виде и непосредственно выводит ее в процессе выполнения. В этом случае константа c интерпретируется как вычислительные ресурсы, необходимые для организации выдачи строки, хранящейся в теле программы P.

Основной результат алгоритмической теории информации заключается теореме Соломонова-Колмогорова, утверждающей, что среди алгоритмов, восстанавливающих строки по их описанию, существует оптимальный, использующий описание наименьшей длины [2, с. 11]. В свою очередь длины программ, реализующих этот оптимальный алгоритм, отличаются не более чем на константу, зависящую от используемого способа кодирования программ.

Другой важный результат теории – это не вычислимость функции $K(s)$ [2, с. 17]. Иными словами, не существует алгоритма, который принимал бы на вход строку s и выдавал на выходе число $K(s)$. Аналогичный результат получен в [3], утверждающий о несуществовании алгоритма, который для любой логической функции строит минимальный алгоритм ее вычисления.

---

[1]

Причина не вычислимости $K(s)$ заключается в том, что для каждой строки s существует свой алгоритм, восстанавливающий эту строку из ее минимального описания. Поэтому сконструировать алгоритм вычисления $K(s)$, включающий в себя счетное множество других алгоритмов по числу строк конечной длины в конечном алфавите, не представляется возможным, так для этого требуется неограниченное время и неограниченная память. Однако для каждой конкретной строки минимальный алгоритм может быть построен и, на его основе, может быть найдена Колмогоровская сложность этой строки.

Настоящая статья посвящена оценке Колмогоровской сложности строк в двоичном алфавите на основе известных оценок сложности формул логических функций. Для решения этой задачи двоичная строка рассматривается как последовательность значений некоторой логической функции, зависящей от логических переменных. Для построения программы, вычисляющей значения этой функции, находится формула этой функции в базисе бинарных логических операций. После двоичного кодирования этой формулы строится программа, которая последовательно вычисляет элементы исходной строки. В этом случае длина программы, непосредственно восстанавливающей исходную строку, называется алгоритмической сложностью этой строки.

Для минимизации длины восстанавливающей программы последняя снова рассматривается как двоичная строка, подлежащая сжатию. Итерации завершаются, когда длина программы перестает уменьшаться. В результате находится программа минимальной длины, восстанавливающая исходную строку в результате последовательных вычислений множества восстанавливающих программ. Длина такой программы считается минимальной и называется Колмогоровской сложностью исходной строки.

1. Логические функции

Пусть задана логическая функция f, принимающая значения на множестве N_2 и существенно зависящая от n логических переменных x_0, x_1, …, x_{n-1}, также принимающих значения на множестве N_2, где $N_2 = \{0,1\}$.

Поставим задачу определить $L_2(n)$ – наибольшее число бинарных операций, которых будет достаточно, чтобы представить логическую функцию f в виде формулы. Будем предполагать, что доступны все бинарные операции.

В булевой алгебре число бинарных операций, существенно зависящих от своих переменных, равно десяти [4]. Для кодирования одной такой операции можно использовать 4-х разрядное двоичное число. Однако вместо номера бинарной операции будем использовать ее матричное описание, задаваемое числом той же длины. Например, для булевой импликации $x_0 \to x_1$ матрица операции $\to$ и соответствующее ей двоичное число запишем так:

$$x_0 \to x_1 = x_0 \begin{matrix} \\ \end{matrix}^{\begin{matrix} x_1 \end{matrix}} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \leftrightarrow 1011, \quad x_0 \to x_1 = x_0 (1011) x_1 .$$

В свою очередь формулу ψ функций f будем задавать в виде

$$\psi \to (\chi \otimes \varphi) \oplus ((\chi \otimes \varphi) \oplus \ldots \oplus (\chi \otimes \varphi)), \tag{1}$$

где χ – терминальное вхождение переменной, $\oplus$ и $\otimes$ – терминальные вхождения бинарных операций (функций, существенно зависящих от своих двух переменных), φ – формула от меньшего числа переменных той же конструкции.

Следует заметить, что аналитическая конструкция (1) не содержит вхождения констант и унарных операций (функций, существенно зависящих от одной переменной). Это связано с тем, что формула с вхождением констант или унарных операций может быть тождественно преобразована в формулу, их не содержащую, путем изменения рядом расположенных бинарных операций. Доказательство этого факта содержится в [4].

Приведем известные верхние оценки числа операций, которые необходимо выполнить, чтобы вычислить значение произвольной дискретной функции при заданных значениях ее переменных.

Теорема 1 [5, 6**]**. Максимальное число операций $L_2(n)$, необходимых для выражения формулой произвольной логической функции от n переменных, удовлетворяет следующим оценкам:

$$\frac{1}{\log_2 3}\frac{2^n}{n} < L_2(n) < 2(1+\log_2 n)\frac{2^n}{n}\,. \quad (2)$$ ♦

Для оценки числа операций, необходимых для вычисления не полностью определенных функции, у которой число значений N меньше 2^n, но больше 2^{n-1}, воспользуемся линейной интерполяцией,

$$L_2(\log_2 N) = L_2(n-1) + (L_2(n) - L_2(n-1)) \times (\log_k N - n + 1)\,, \quad (3)$$

где $n = \lceil \log_k N \rceil$, а $\lceil q \rceil$ – наименьшее целое, большее или равное q. В этом случае получается несколько завышенная оценка числа операций, так как $L_2(n)$ растет ускоренно как $N/\log_2 N$ [7].

На рис. 1 приведен график зависимости нижней и верхней оценки $L_2(n)$ от числа переменных n с линейной интерполяцией (3) в промежуточных точках.

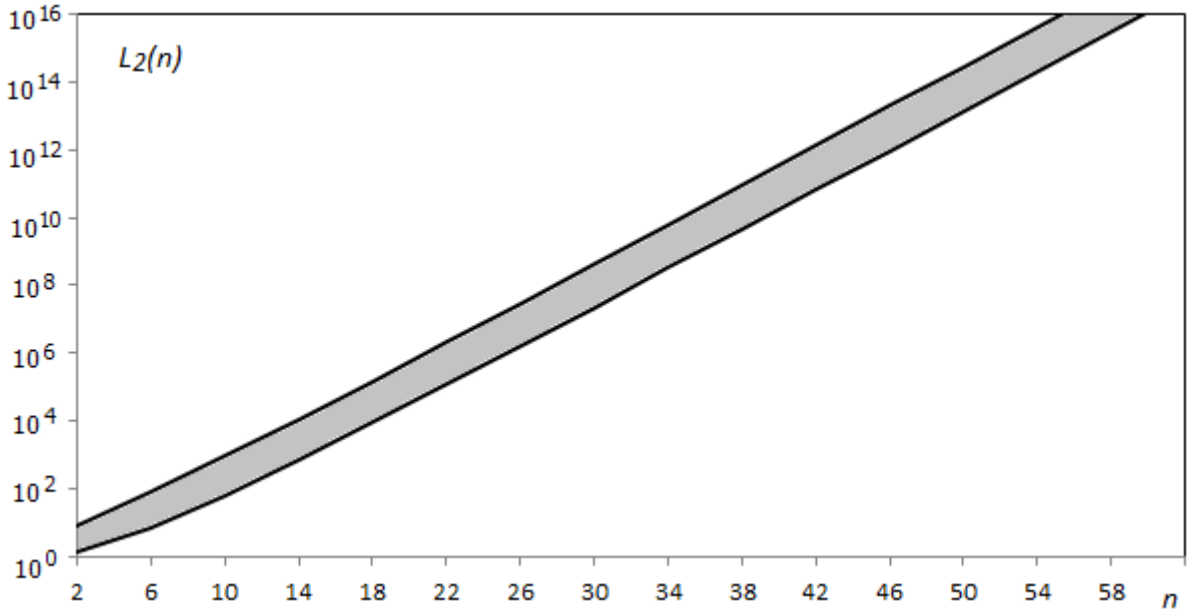

Рис. 1. Число операций для выражения логической функций

2. Вычисление логических функций

Из теоремы 1 следует, что программа, вычисляющая значение произвольной логической функции от n переменных, выполняет не более $L_2(n)$ операций. Для вычисления значения функции по ее формуле будем использовать вычислительное средство со стековой организацией вычислений. В этом случае программа состоит из команд двух типов: команд загрузки значения переменной на вершину стека и команд бинарных операций, два операнда которых находятся в стеке.

Для декодирования команд в начале каждой команды предусмотрим специальное поле типа команды – одноразрядное двоичное число. Для указания на команду загрузки стека туда запишем единицу, а для указания на команду бинарных операций – число ноль.

Очевидно, что для кодирования номера загружаемой переменной требуется $\lceil \log_2 n \rceil$ ячеек памяти. Так как последовательность номеров загружаемых переменных можно представить в виде одного многоразрядного числа, задаваемого в системе счисления с основанием n, но записанного в двоичной системе счисления, то для кодирования последовательности m переменных потребуется $\lceil m \log_2 n \rceil$ двоичных пазрядов.

Из анализа аналитической конструкции (1) следует, что число загрузок переменных в стек на единицу больше числа бинарных операций. Отсюда следует, что область памяти, выделяемая для кодирования последовательности переменных в формуле, должна иметь объем $\lceil (L_2(n)+1) \log_2 n \rceil$ двоичных разрядов.

Команда загрузки стека кодируется только одним разрядом – полем типа команды единичного значения. Для кодирования всех команд загрузки стека потребуется $L_2(n)+1$ разрядов, а для кодирования всех бинарных команд – $5L_2(n)$ разрядов.

Для передачи вычислительному средству числа переменных n в начале каждой программы запишем n нулевых разрядов. Так как первой командой любой программы должна быть команда загрузки стека с единичным кодом, то начальная часть программы с перечислением переменных легко отделяется от следующей за ней последовательности команд.

Для завершения выполнения программы предусмотрим команду останова, состоящую из пяти нулевых разрядов, представляющую собой бинарную операции с нулевым определением или константу ноль. Таким образом, число команд загрузки стека и бинарных операций будет одинаковым и равным $L_2(n)+1$. В итоге получаем объем памяти программы $V_2(n)$,

$$V_2(n) = n + (L_2(n)+1)(6+\log_2 n). \tag{4}$$

Структура программы приведена на рис. 2, где B – ячейки памяти с определением бинарных операций, а X – ячейки памяти с закодированной последовательностью номеров переменных для команд загрузки стека. Из рисунка видно, что программа является самоограниченной, т.е. из строки программы легко могут быть выделены ее составные части:

– область для хранения текущих значений переменных, откуда также может быть получено их число n;

– область, содержащая последовательность команд, начинающаяся с команды загрузки стека и заканчивающаяся командой останова;

– область, содержащая последовательность номеров загружаемых переменных, за-

кодированная в виде многоразрядного числа.

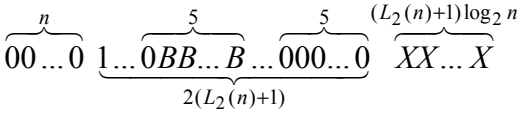

Рис. 2. Структура программы вычисления функции

Пример 1. Построим программу для вычисления логической функции с формулой

$$f(x_2, x_1, x_0) = \left(x_0 \otimes_1 \left(x_2 \otimes_3 x_1\right)\right) \oplus_1 \left(x_1 \otimes_2 \left(x_2 \otimes_4 x_0\right)\right),$$

где $\otimes_1 = 0111$, $\otimes_2 = 1001$, $\otimes_3 = 1101$, $\otimes_4 = 0100$, $\oplus_1 = 0110$. Перепишем формулу в обратной польской нотации [8, с. 336],

(5) $$f(x_2, x_1, x_0) = x_2 x_1 \otimes_3 x_0 \otimes_1 x_2 x_0 \otimes_4 x_1 \otimes_2 \oplus_1 .$$

Сформируем тело программы в соответствии с рис. 2 и формулой (5), где вхождения переменных заменим командами загрузки стека с кодом 1, а перед каждой бинарной операцией запишем код 0 и четыре разряда ее определения. После последней команды формулы запишем команду останова 00000 и число, кодирующее последовательность загрузки переменных: 2, 1, 0, 2, 0, 1. В итоге имеем следующую программу:

000 1 1 01101 1 00111 1 1 00100 1 01001 00110 00000 110111110,

где последняя группа из девяти разрядов – представление троичного числа 102012 в двоичной системе счисления. ♦

После загрузки и раскодирования программы в ее первые n ячеек записываются значения переменных, для которых требуется вычислить значение функции. Программа после запуска преобразует эти значения в выходной результат, размещаемый на вершине стека. Таким образом, показана справедливость следующей теоремы.

Теорема 2. Для вычисления значения логической функции, существенно зависящей от n переменных, существует программа длиной $V_2(n)$ двоичных разрядов. ♦

На рис. 3 показана зависимость нижней и верхней оценки относительной длины программы

$$\nu_2(n) = V_2(n) / 2^n$$

от числа переменных n. Из рисунка видно, что при малых значениях n длина программы $V_2(n)$ всегда превосходит длину строки, состоящей из 2^n значений вычисляемой логической функции.

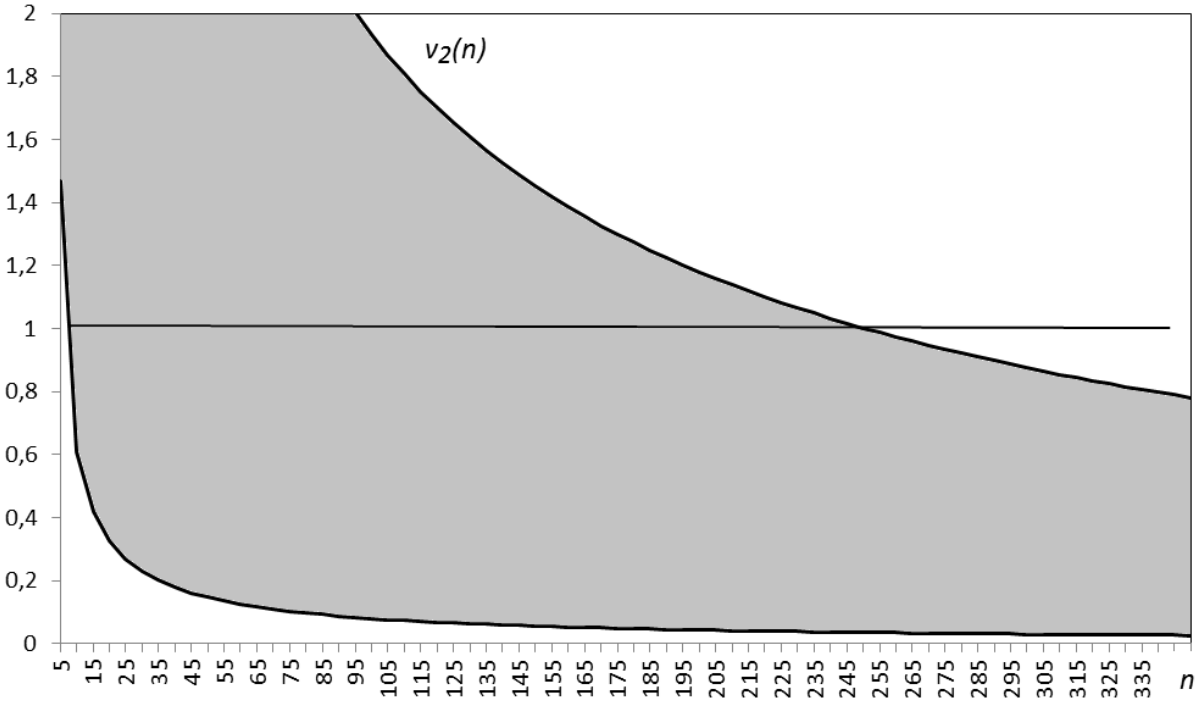

Рис. 3. Относительная длина программ

Следует заметить, что с ростом n относительная длина программ стремится к нулю,

$$\lim_{n\to\infty} v_2(n) = \lim_{n\to\infty} \frac{V_2(n)}{2^n} = \lim_{n\to\infty} \left\{ \begin{array}{c} \dfrac{1}{\log_2 3} \dfrac{\log_2 n}{n} \\ 2\dfrac{(\log_2 n)^2}{n} \end{array} \right\} = 0,$$

где в фигурных скобках приведены промежуточные выражения для нижней и верхней оценки $L_2(n)$.

Из (4) может быть получена оценка числа переменных, при которых относительная длина программы равна единице:

$$7 < n < 251.$$

3. Алгоритмическая сложность

Построим программу, которая будет последовательно вычислять первые $N \le 2^n$ значения логической функции. Для этого команду останова (рис. 2) будем интерпретировать как команду цикла, а число итераций N закодируем n-разрядным двоичным числом, которое запишем непосредственно после этой команды (рис. 4).

$$\overbrace{00\ldots0}^{n}\ \underbrace{1\ldots\overbrace{0BB\ldots B}^{5}\ldots\overbrace{00000}^{5}}_{2(L_2(n)+1)}\ \overbrace{NN\ldots N}^{n}\ \overbrace{XX\ldots X}^{(L_2(n)+1)\log_2 n}$$

Рис. 4. Структура программы порождения строк

Команда цикла увеличивает на единицу n-разрядное число в начальных ячейках программы, хранящее текущие значения переменных, и возобновляет вычисления с первой команды. Если увеличенное число становится равным N, то итерации завершаются.

Пример 2. Модифицируем программу из примера 1 для вычисления всей строки, а не одного ее элемента. Для этого команду останова заменим командой цикла, после которой поместим число повторений цикла, равное длине исходной строки в двоичной системе счисления:

000 1 1 01101 1 00111 1 1 00100 1 01001 00110 01111 111 110111110. ♦

Таким образом, построена программа, которая последовательно вычисляет значения логической функции при различных значениях ее переменных, т.е. записывает в стек последовательность первых N значений этой функции. В итоге показана справедливость следующей теоремы.

Теорема 3. Для порождения произвольной двоичной строки длины N существует программа длиной $S_2(N)$:

$$S_2(N) = 2n + (L_2(n) + 1)(6 + \log_2 n), \tag{6}$$

где $n = \lceil \log_2 N \rceil$.

Зависимость коэффициента алгоритмического сжатия строк $Z_2(n) = 2^n / S_2(2^n)$ от числа логических переменных n приведена на рис. 5.

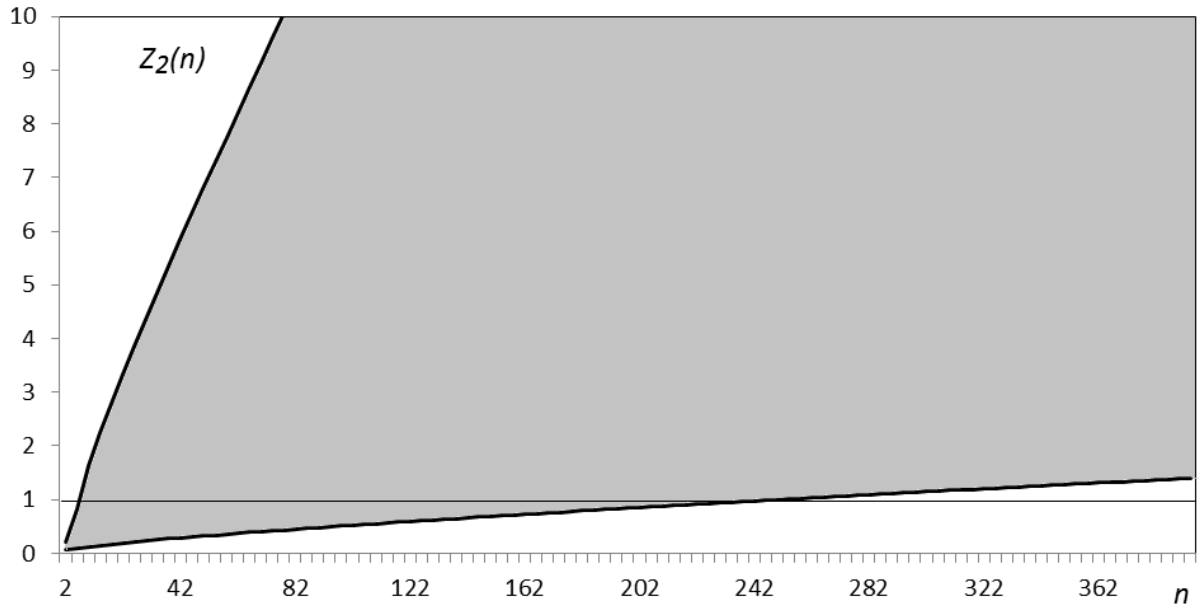

Рис. 5. Коэффициент алгоритмического сжатия строк

Длина программы $S_2(N)$ является алгоритмической сложностью двоичной строки длины N, а относительная алгоритмическую сложность $\sigma_2(N)$,

$$\sigma_2(N) = \frac{S_2(N)}{N},$$

является величиной, обратной коэффициенту алгоритмического сжатия.

Теорема 4. Относительная алгоритмическая сложность двоичных строк $\sigma_2(N)$ при увеличении длины строки N стремится к нулю.

Доказательство.

$$\lim_{N\to\infty}\sigma_2(N)=\lim_{n\to\infty}\frac{S_2(2^n)}{2^n}=\frac{2n+(L_2(n)+1)(6+\log_2 n)}{2^n}=0\,.\ \blacklozenge$$

Если не рассматривать вопросы кодирования программ, то длина программы $\Pi_2(n)$, необходимой для восстановления булевой строки длиной 2^n, удовлетворяет следующей оценке:

$$\Pi_2(n)>\log_2 n+(L_2(n)+1)\log_2 40n\,, \tag{7}$$

где учтено хранение в теле программы числа переменных n, последовательности из $L_2(n)+1$ команд загрузки стека и $L_2(n)+1$ команд бинарных операций, кодирующих структуру формулы, а также последовательности номеров используемых бинарных операция длиной $(L_2(n)+1)\log_2 10$ и номеров загружаемых переменных длиной $(L_2(n)+1)\log_2 n$ (рис. 6).

$$\overbrace{n\ldots n}^{\log_2 n}\ \overbrace{110\ldots0}^{2(L_2(n)+1)}\ \overbrace{BB\ldots B}^{(L_2(n)+1)\log_2 10}\ \overbrace{XX\ldots X}^{(L_2(n)+1)\log_2 n}$$

Рис. 6. Структура минимальной программы восстановления строк

4. Колмогоровская сложность

Колмогоровская сложность строки – это минимальная длина программы, восстанавливающей эту строку. Ранее при оценке алгоритмической сложности строки порождающая ее программа строилась как двоичная строка. Нет никаких препятствий для построения программы, порождающей строку самой порождающей программы. В некоторых случаях это позволяет уменьшить ранее полученные оценки алгоритмической сложности.

Для получения Колмогоровской сложности $K_2(N)$ двоичной строки длины N процесс построения программы выполняется до тех пор, пока длина следующей программы $V^{(i)}$ будет меньше предыдущей $V^{(i-1)}$:

$$\begin{cases} V^{(0)}=S_2(N); \\ V^{(i)}=S_2(V^{(i-1)})\ \ (V^{(i)}<V^{(i-1)}, i=1,2,\ldots); \\ K_2(N)=V^{(i-1)}. \end{cases} \tag{8}$$

Для определения необходимости продолжить восстановление строки команду цикла с кодом 00000 будем заменять командой цикла с кодом 01111, которая также как и первая команда не является бинарной операцией, существенно зависящей от своих переменных (рис. 7).

$$\overbrace{00\ldots0}^{n}\ \underbrace{1\ldots\overbrace{0BB\ldots B}^{5}\ldots\overbrace{01111}^{5}}_{2(L_2(n)+1)}\ \overbrace{NN\ldots N}^{n}\ \overbrace{XX\ldots X}^{(L_2(n)+1)\log_2 n}$$

Рис. 7. Структура промежуточной программы восстановления строк

Теорема 5. Процедура вычисления Колмогоровской сложности (8) сходится для всех строк.

Доказательство. Функция $S_2(N)$ с учетом аппроксимации (3) является монотонно убывающей при уменьшении N. Также известно, что при $n=2$ формула логической функции состоит всего из одной операции, которая непосредственно содержит вычисляемую строку. В этом случае из формулы (6) получаем $V^{(i)} > V^{(i-1)}$. Следовательно, последовательность $V^{(i)}$ является монотонной убывающей и ограниченной. Это доказывает сходимость (8). ♦

Рассмотрим относительную Колмогоровскую сложность двоичных строк $\mu_2(N)$,

$$\mu_2(N) = \frac{K_2(N)}{N},$$

которая интерпретируется как коэффициент Колмогоровского сжатия строк. На рис. 8 приведены графики зависимости от n относительной алгоритмической и относительной Колмогоровской сложности двоичных строк исходной длины 2^n.

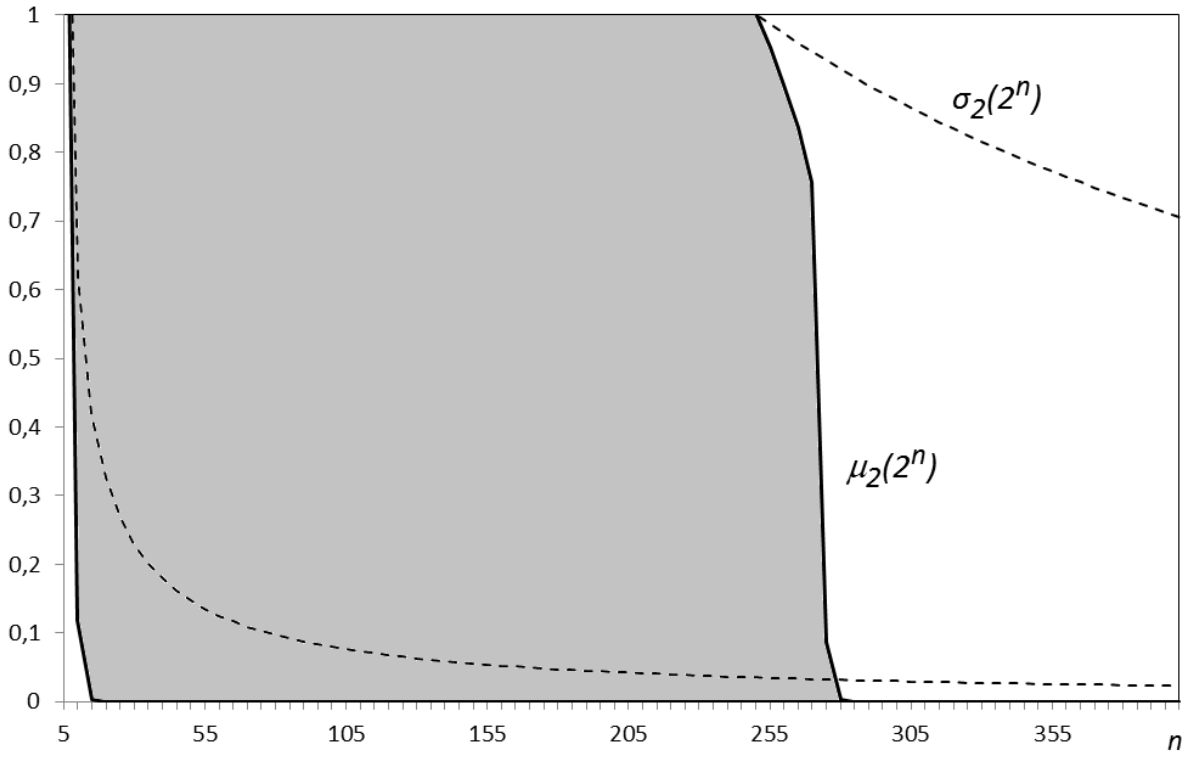

Рис. 8. Относительные алгоритмическая и Колмогоровская сложности строк

На рис. 9 приведены результаты вычисления максимальной Колмогоровской сложности K.

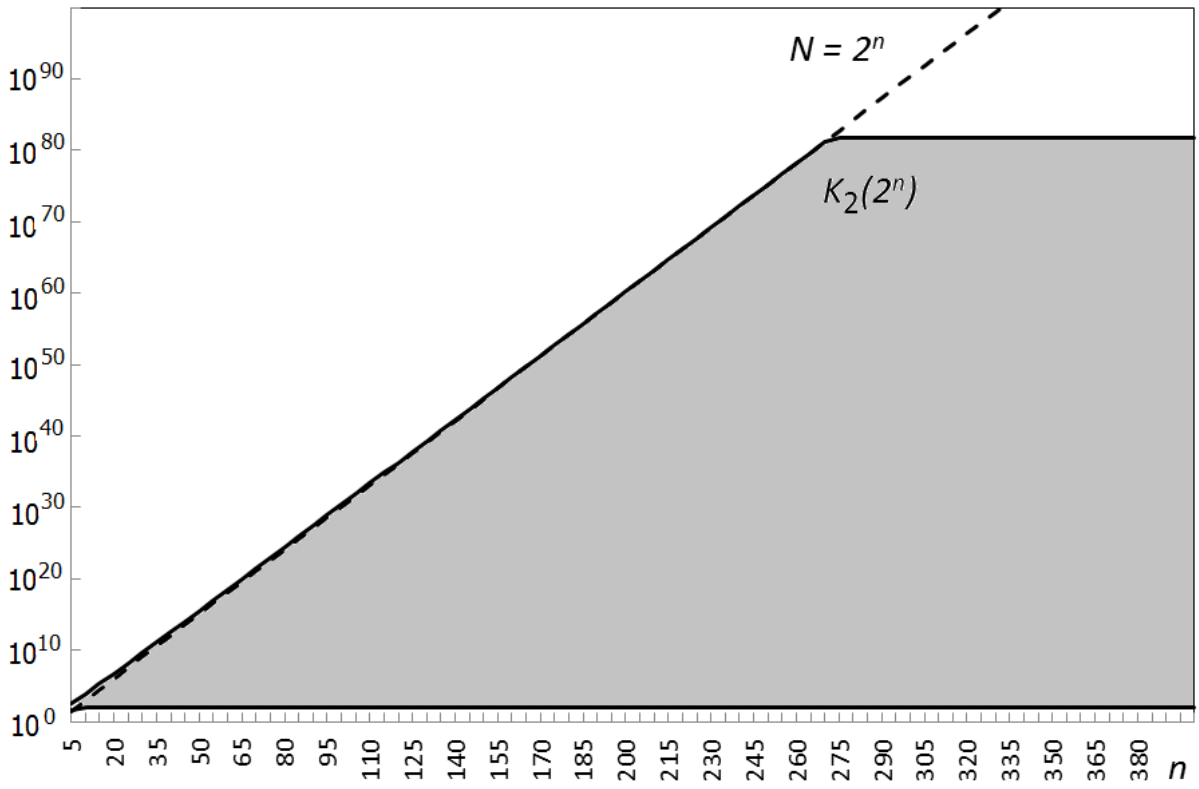

Рис. 9. Максимальная Колмогоровская сложность

Из рисунка видно, что все строки, имеющие длину, большую некоторой константы K, сжимаются до длины, равной K. Однако при этом число повторных сжатий может быть достаточно большим при большой начальной длине строки (см. рис. 5).

Заключение

В настоящей статье доказано существование константы K – максимальной Колмогоровской сложности строк – такой, что все строки большей длины сжимаются до длины, равной K. Это связано с тем, для таких строк может быть построена восстанавливающая строку программа меньшей длины. При этом сама построенная программы также является строкой, для порождения которой может быть построена другая восстанавливающая программа меньшей длины. В пределе получается строка с длиной K.

Практическая применимость полученных результатов зависит от получения уточненных нижних и верхних оценок максимального числа операций, необходимых для вычисления произвольной логической функции, а также нахождения методов синтеза формул, удовлетворяющих этим оценкам. Второй актуальной задачей является нахождения точного значения максимальной Колмогоровской сложности K.

Список литературы

1. Колмогоров, А.Н. *Три подхода к определению понятия «количество информации»* // Проблемы передачи информации. 1965. Т. 1. Вып. 1. С. 3-11.
2. Верещагин, Н.К. *Колмогоровская сложность и алгоритмическая случайность* / Н.К. Верещагин, В.А. Успенский, А. Шень. М.: Изд. МЦНМО, 2013. 575 с.

3. Марков, А. А. *О нормальных алгоритмах, связанных с вычислением булевых функций* // Изв. АН СССР. 1967. Сер. математическая. Т. 31, № 1. С. 161-208.
4. Выхованец, В.С. *Аналитическая идентификация дискретных объектов* // Автоматика и телемеханика. 2011. № 7. С. 159-189.
5. Even, S. *On minimal modulo 2 sums of products for switching functions* / S. Even, I. Kohavi, A. Paz // IEEE Trans. Elect. Comput. 1967. P. 671-674.
6. Кириченко К.Д. *Верхняя оценка сложности полиномиальных нормальных форм булевых функций* // Дискретная математика. 2005. Т. 17, вып. 3. С. 80-88.
7. Андреев, А. Е. *О сложности реализации частичных булевых функций схемами из функциональных элементов* // Дискретная математика. 1989. Том 1, вып. 4. С. 36–45.
8. Пратт Т., Зелковиц М. *Языки программирования: разработка и реализация* / Т. Пратт, М. Зелковиц. Спб.: Питер, 2002. 688 с.