В.А. Кирюхин

Алгоритм построения линейных блоковых двоичных кодов по заданному числу информационных символов и числу исправляемых ошибок

В статье предложен алгоритм построения линейных блоковых двоичных кодов по заданному числу информационных символов и числу исправляемых ошибок. Дана теоретическая оценка сложности предложенного алгоритма, произведено экспериментальное исследование времени работы. На основе анализа результатов работы предложенного алгоритма был сделан вывод о том, что параметры построенных кодов совпадают с параметрами кодов, найденных полным перебором. Было произведено сравнение параметров построенных кодов с параметрами некоторых известных в литературе кодов (БЧХ, Голея), которое показало, что в большинстве случаев параметры построенных кодов не уступают известным, а в остальных случаях незначительно хуже.

Ключевые слова: помехоустойчивое кодирование, линейные коды.

Большинство процессов, связанных с накоплением, хранением и передачей информации, протекают в условиях воздействия разнообразных помех, способных исказить хранимые и обрабатываемые данные. Для противодействия помехам необходимо использовать методы, позволяющие обнаруживать и корректировать подобные ошибки. С математической точки зрения задача сводится к построению так называемых помехоустойчивых кодов.

Одним из классов помехоустойчивых кодов являются блоковые коды, делящие информацию на фрагменты постоянной длины и обрабатывающие каждый из них в отдельности. Практически все используемые блоковые коды являются линейными. Это связано с тем, что нелинейные коды значительно труднее исследовать, и для них трудно обеспечить приемлемую сложность кодирования и декодирования.

Несмотря на большое количество литературы, связанной с линейными кодами, вопрос существования кода с заранее заданными

параметрами (n, k, t) является открытым. Очевидно, что использование алгоритма полного перебора для построения линейного кода неэффективно из-за высокой вычислительной сложности. В данной статье рассмотрен алгоритм поиска оптимального, с точки зрения длины кодового слова, линейного двоичного блокового кода при заданных k и t.

Необходимые теоретические сведения

Корректирующая способность линейного блокового (n, k)-кода (n, k) зависит от минимального кодового расстояния d_{min}, определяемого как наименьшее расстояние Хемминга между всеми парами кодовых слов. Известно, что d_{min} может быть вычислено как минимальный вес ненулевого кодового слова[1].

Для того чтобы код исправлял ошибки кратности t, должно быть выполнено соотношение $d_{min} = 2t + 1$.

Рассмотрим границы параметров кода. Если существует код с параметрами n, k, t, то верно неравенство $2^{n-k} \geq \sum_{l=0}^{t} \binom{n}{l}$ (Верхняя граница Хемминга)[2]. С другой стороны, существует (n, k)-код с d_{min}, равным, по меньшей мере, d, параметры которого удовлетворяют неравенству $\sum_{l=0}^{d-2} \binom{n}{l} \geq 2^{n-k}$ (Нижняя граница Варшамова–Гильберта)[3]. Таким образом, имеем соотношение $\sum_{l=0}^{2t-1} \binom{n}{l} \geq 2^{n-k} \geq \sum_{l=0}^{t} \binom{n}{l}$. Отсюда, при фиксированных k и t, можем найти n_{min} и n_{max} такие, что $n \in [n_{min}, n_{max}]$.

Линейный блоковый код можно задать с помощью порождающей матрицы $G_{k \times n}$, такой, что $\vec{u}G = \vec{v}$, где $\vec{u}$ – информационное слово длины k, $\vec{v}$ – кодовое слово длины.

Каждый линейный (n, k)-код эквивалентен систематическому, т. е. может быть задан с помощью матрицы $G_{k \times n} = \left(I_k P_{k \times (n-k)}\right)$, где I_k – единичная матрица.

Полный перебор и его вычислительная сложность

Воспользуемся тем, что перебирать необходимо только систематические коды. Будем рассматривать все возможные матрицы дополнений $P_{k\times(n-k)}$, и проверять, являются ли каждые $d-1$ ее строк линейно независимыми (достаточное условие того, что d_{min} кода равно d [4]). Если для минимально возможного n ни одной такой матрицы не существует − увеличим на единицу рассматриваемый n и снова попытаемся построить требуемую матрицу. Продолжим увеличение n до тех пор, пока матрица не будет найдена.

Оценим вычислительную сложность такого алгоритма. Число двоичных векторов длины n-k равно 2^{n-1}, число матриц, составленных из таких векторов, различных с точностью до перестановки строк, равно $C^k_{2^{n-k}}$. Проверим, что каждая строка из матрицы имеет вес больше $d_{min}-1$, каждая сумма по всем возможным парам строк имеет вес больше $d_{min}-2$ и т. д. Таким образом, необходимо проверить $C^1_k + C^2_k + ... + C^{min(2t,k)}_k$ комбинаций, это число можно оценить как 2^k. Сложность суммирования двух векторов и сложность проверки веса вектора будем считать константой. Следовательно, число операций, необходимых для проверки одной матрицы $P_{k\times(n-k)}$, будет пропорционально $\sum_{i=1}^{min(2t,k)} (i-1)C^i_k$. Приблизительная оценка этой суммы − $k2^k$. В итоге получаем $O\left(k2^k \sum_{n=n_{mix}}^{n_{max}} C^k_{2^{n-k}}\right)$.

Алгоритм

Как отмечалось выше, для построения линейного блокового кода достаточно найти матрицу дополнений $P_{k\times(n-k)}$. Будем пытаться построить матрицу дополнений начиная с $n = n_{min}$.

Матрица P состоит из k векторов длины $r = n-k$. Рассмотрим множество B двоичных векторов длины r. Мощность множества B равна 2^r.

Исходя из того что все векторы в матрице P различны и того что перестановка векторов будет давать эквивалентную порожда-

ющую матрицу, будем строить матрицу дополнений следующим образом:

– пронумеруем векторы, принадлежащие множеству B. Будем отбирать только те, чей вес больше $d_{min}-1$. Выберем первый – b_1;
– выберем (из следующих за первым) второй – b_2 таким образом, чтобы вес $\omega(b_1+b_2)\geq d_{min}-2$ («+» – сложение по модулю 2);
– выберем, из следующих за вторым, третий таким образом, чтобы $\omega(b_1+b_3)\geq d_{min}-2$, $\omega(b_2+b_3)\geq d_{min}-2$ и $\omega(b_1+b_2+b_3)\geq d_{min}-3$;
– на шаге i сумма любой пары векторов должна иметь вес не меньше $d_{min}-2$, сумма любой тройки не меньше $d_{min}-3$, аналогично до комбинаций из $d_{min}-1$ векторов, вес которых должен быть ненулевым.

Если нам удалось найти по такому принципу k векторов, мы нашли код с заданными параметрами. Можем выйти из алгоритма или перейти на предыдущий шаг для поиска других кодов с такими же параметрами.

Если k векторов найти не удалось, увеличим n на единицу, создадим заново множество B. Тем самым будем рассматривать векторы, отобранные ранее, в новом пространстве с увеличенным числом измерений, считая у них нулевой координату нового измерения.

Увеличивая n мы либо построим на каком-то шаге матрицу дополнений, либо n достигнет нижней границы Варшамова–Гильберта, код с таким n удовлетворяет требуемым параметрам, следовательно, матрица будет построена.

Построенный код будет удовлетворять требуемым параметрам k и t. Для того чтобы доказать это, покажем, что вес любого ненулевого кодового слова не меньше, чем $d_{min}=2t+1$.

Рассмотрим произвольный ненулевой вектор $\vec{u}$ длины k. Если его вес больше или равен d_{min}, то кодовое слово $\vec{v}=\vec{u}G_{k\times n}=\vec{u}\left(I_k P_{k\times(n-k)}\right)$ будет, очевидно, иметь вес не меньше d_{min}, так как $\vec{u}I_k=\vec{u}$. Если вес вектора $\vec{u}$ равен $x<d_{min}$, то вес вектора $\vec{v}$ будет определяться весом суммы какого-то набора x строчек из матрицы P. Вес этой суммы векторов будет больше или равен, чем $d_{min}-x$. Таким образом, $\omega(\vec{v})=\omega(\vec{u}I_k)+\omega\left(\vec{u}I_k P_{k\times(n-k)}\right)\geq x+d_{min}-x$, $\omega(\vec{v})\geq d_{min}$. В силу того что вектор выбран произвольно, соотношение $\omega(\vec{v})\geq d_{min}$ выполнено для любого кодового слова, следовательно, построенный код действительно удовлетворяет заданным параметрам.

Оценка сложности алгоритма

Дадим оценку сверху для вычислительной сложности алгоритма. Необходимо перебрать 2^{n-k} векторов. Максимальное значение n найдем из нижней границы Варшамова–Гильберта (В.–Г.) $\sum_{l=0}^{d-2}\binom{n}{l} \geq 2^{n-k}$. (Согласно результатам экспериментов значение n ближе к верхней границе Хемминга, см. рис. 1.)

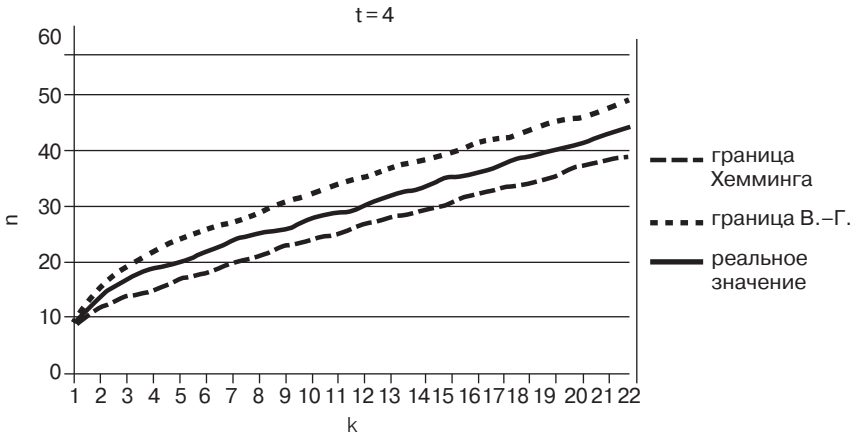

Рис. 1. Зависимость параметра n от k при $t = 4$

Для каждого рассматриваемого вектора нужно подсчитать его вес за счет использования вычисленной заранее таблицы весов и ограничения на длину вектора, время подсчета будет постоянным – $O(1)$. Если вес вектора подходит, проверим: может ли вектор принадлежать матрице дополнений? Для этого просуммируем его со всеми возможными комбинациями из уже отобранных векторов и вычислим вес полученных комбинаций. Максимальное число таких комбинаций равно $C_k^1 + C_k^2 + ... + C_k^{min(2t,k)}$, это число можно оценить как 2^k. (Естественно, число комбинаций будет принимать такое значение только к самому концу работы алгоритма.) Сложность операции сложения – $O(1)$, вычисление веса – $O(1)$. Таким образом, сложность проверки равна $O(2^k)$. Таким образом получаем, что необходимо перебрать 2^{n-k} векторов, часть из них будет отсеяна за счет проверки веса, для других надо осуществить проверку со сложностью $O(2^k)$. Вычислим число векторов в первой и во второй части. Общее число векторов 2^{n-k}, число векторов с весом, меньшим $d_{min}-1=2t$,

равно $C^0_{2^{n-k}} + C^1_{2^{n-k}} + ... + C^{2t-1}_{2^{n-k}}$. Таким образом, вычислительная сложность предложенного алгоритма – $O\left(\sum_{i=0}^{2t-1} C^i_{2^{n-k}} + 2^k \sum_{i=2t}^{k} C^i_{2^{n-k}}\right)$. Дадим оценку сверху, не учитывая отбор векторов с неподходящим весом, — $O\left(2^{n-k}2^k\right) = O\left(2^n\right)$, где $n \in \left[n_{min}, n_{max}\right]$, а n_{min} и n_{max} определяются из границ Хемминга и Варшамова–Гильберта. Нужно отметить, что при увеличении параметра t значение n становится ближе к n_{min}.

Экспериментальные результаты

Сравним скорость работы рассмотренного алгоритма и алгоритма полного перебора. Нужно отметить, что время работы алгоритма полного перебора в первую очередь зависит от того, насколько близко значение параметра n к минимальной границе. В случае если искомый код является совершенным (или близок к этому), т. е. $n = n_{min}$ и соответственно $2^{n-k} = \sum_{l=0}^{t} \binom{n}{l}$ – время алгоритма полного перебора гораздо меньше за счет отсутствия необходимости проверять огромное число вариантов с меньшими значениями n, для которых ни одного линейного кода не существует. У рассмотренного в текущей статье алгоритма такой зависимости нет. Время работы зависит только от значения n, при котором код может быть построен, а не зависит от n_{min}. В силу того что время работы алгоритма полного перебора гораздо больше, чем у предложенного алгоритма, не представляется возможным сравнивать время их работы при одинаково большом наборе параметров. Поэтому на рис. 2 и 3 график, соответствующий алгоритму полного перебора, охватывает меньше значений.

Рассмотрим график, показывающий зависимость времени работы алгоритма от заданных параметров k и t (рис. 4). Нужно сказать, что, например, при переходе от некоторых k к $k+1$ рост графика опережает рост 2^n; это объясняется тем, что рост параметра n не находится в линейной зависимости от k и t (рис. 5).

Одним из важнейших критериев сравнения алгоритмов является величина $R = \frac{k}{n}$ – скорость построенного кода. Как видно из

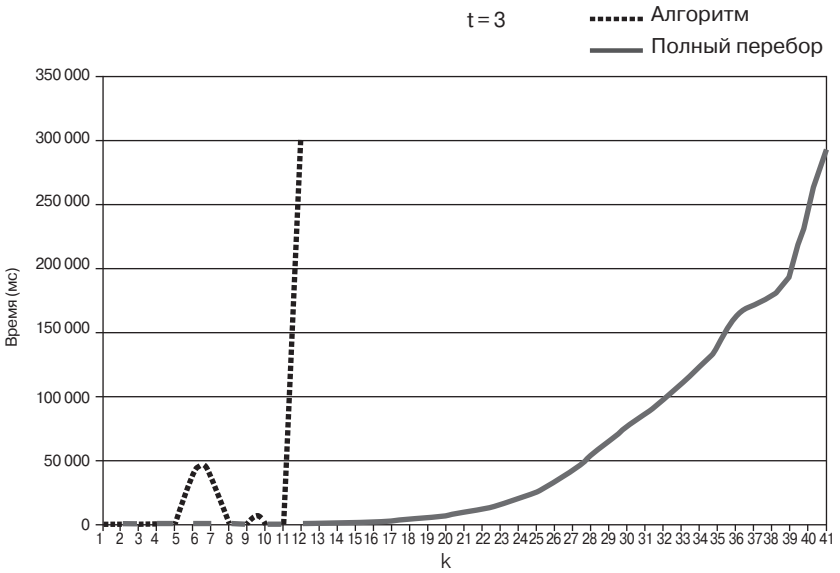

Рис. 2. Сравнение скорости работы двух алгоритмов при $t = 3$

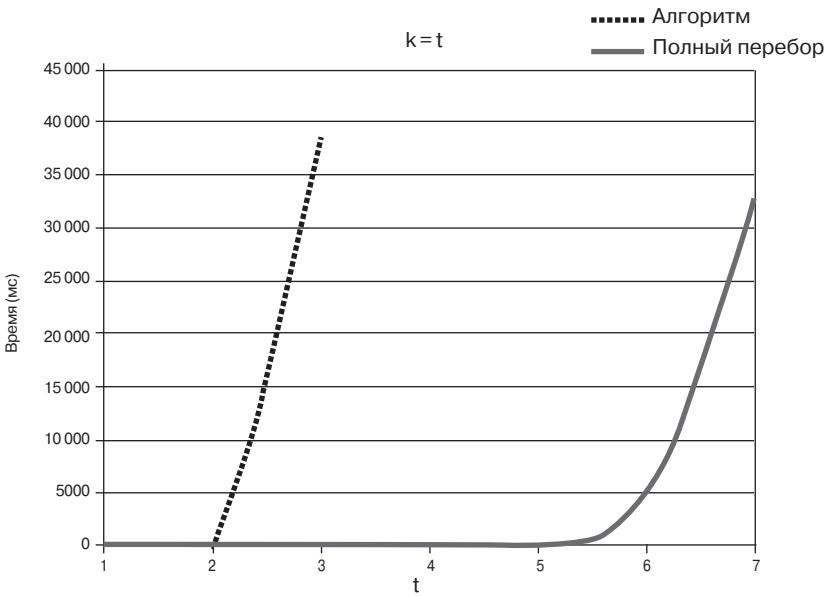

Рис. 3. Сравнение скорости работы двух алгоритмов при $k = 7$, число исправляемых ошибок $t \in [1,7]$

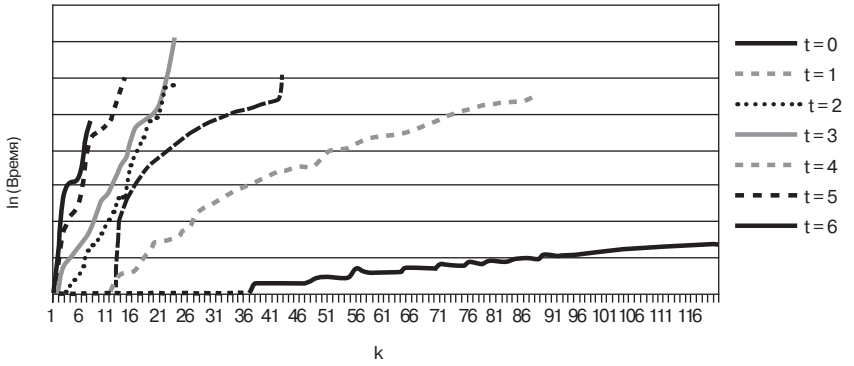

Рис. 4. Зависимость времени работы от параметров k, t

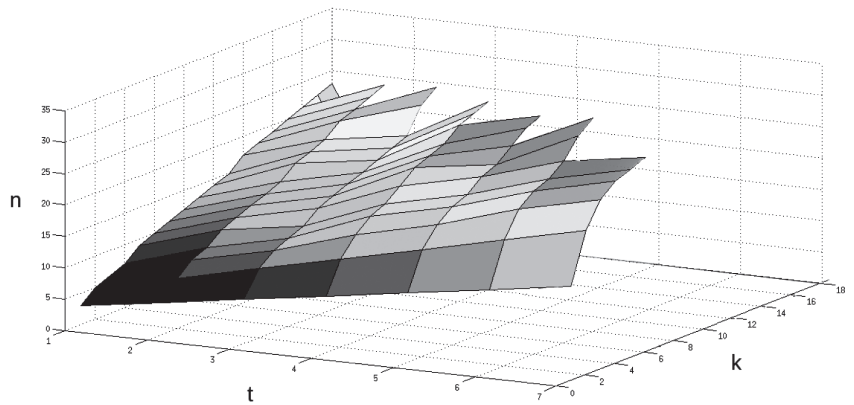

Рис. 5. Зависимость значения n от параметров k, t

графиков (рис. 6 и 7), этот параметр одинаков и у кодов, найденных с помощью полного перебора, и у кодов, построенных предложенным алгоритмом. Т. е. несмотря на то что предложенный алгоритм рассматривает гораздо меньше вариантов, эффективность кодов, построенных с его помощью, не уступает лучшим вариантам, найденным полным перебором.

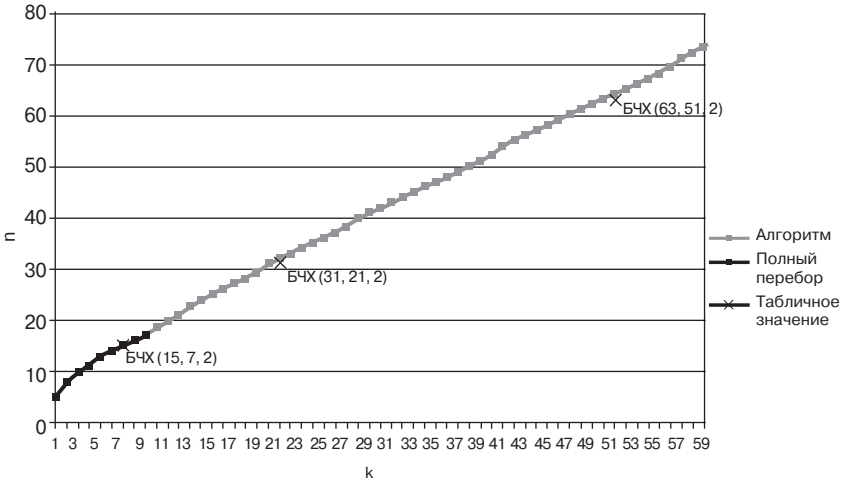

Рис. 6. Сравнение значения *n* у 2-х алгоритмов и табличных данных при $t = 2$

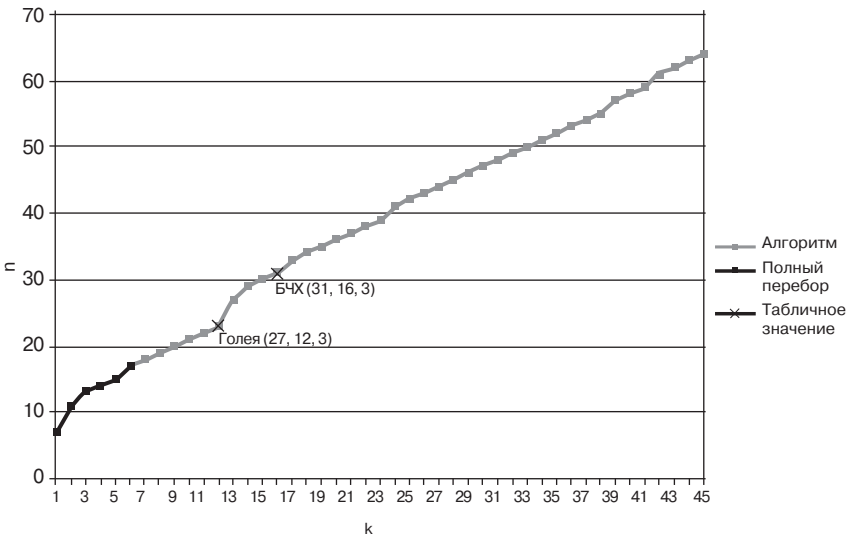

Рис. 7. Сравнение значения n у двух алгоритмов и табличных данных при t = 3

Вместе с тем алгоритм не всегда может найти самый лучший код из теоретически существующих. Это видно, если сравнить его результаты с известными на текущий момент кодами. В некоторых случаях параметр n у найденных решений несколько больше, чем у известных кодов. Например, известно о существовании БЧХ кодов с (n, k, t)-параметрами: (15,7,2), (31,21,2), (63,51,2), (31,16,3); совершенного кода Голея с параметрами (23,12,3)[5]. Предложенный в статье алгоритм находит соответственно коды со следующими параметрами: (15,7,2), (*32*,21,2), (*64*,51,2), (31,16,3), (23,12,3). Т. е. в некоторых случаях алгоритмом найден самый лучший вариант, в других значение $R = \frac{k}{n}$ незначительно хуже.

Рассмотренные алгоритмы были реализованы на языке C++. В качестве компилятора использовался gcc версии 4.8.2. Тесты проводились на ЭВМ с операционной системой LinuxMint 17 на процессоре IntelCore i7 2.2ГГц с 6 Гб ОЗУ.

В статье был предложен алгоритм построения линейных блоков двоичных кодов при заданных параметрах k — числе информационных символов и t — числе исправляемых ошибок. Теоретические и экспериментальные оценки показывают, что предложенный алгоритм обладает существенно меньшей вычислительной сложностью, чем алгоритм полного перебора. На основе анализа результатов работы предложенного алгоритма был сделан вывод о том, что параметры построенных кодов совпадают с параметрами кодов, найденных полным перебором. Было произведено сравнение параметров построенных кодов с параметрами некоторых известных в литературе кодов (БЧХ, Голея), которое показало, что в большинстве случаев параметры построенных кодов не уступают известным, а в остальных случаях незначительно хуже.

Примечания

[1] *Вернер М.* Основы кодирования: Учеб. для вузов. М.: Техносфера, 2004.

[2] *Духин А.А.* Теория информации: Учеб. пособие. М.: Гелиос АРВ, 2007.

[3] Там же.

[4] Там же.

[5] *Гаранин М.В. и др.* Системы и сети передачи информации: Учеб. пособие для вузов. М.: Радио и связь, 2001.