Сравнение алгоритмов декодирования кода Рида-Соломона, исправляющих стирания и несколько ошибок

Крещук А.А.
ИППИ РАН
krsch@iitp.ru

Аннотация

В данной работе описаны коды Рида-Соломона. Представлен алгоритм декодирования, исправляющий ошибки и стирания. Данный алгоритм разбит на четыре этапа. Для каждого из этих этапов приведено несколько методов их реализации. Для каждого метода представлена оценка числа необходимых операций для декодирования представленного в статье РС кода. Представлена новая модификация алгоритма Форни. Построена программная реализация приведённых алгоритмов. Проведена экспериментальная оценка числа требуемых операций умножения.

1. Введение

Коды Рида-Соломона являются кодами с максимальным достижимым расстоянием. К сожалению, коды Рида-Соломона существуют только для определённой длины в конкретном поле. Для поля $GF(q)$ коды РС имеют длину $q-1$. Поэтому для построения кода большой длины приходится использовать поле большой размерности. В технике часто применяются поля $GF(2^m)$, что ещё сильнее сужает выбор кода. Разумным компромиссом между скоростью декодирования и корректирующей способностью является использование РС кода длины 255 над полем $GF(2^8)$. Именно этот код мы и будем рассматривать в статье.

При выборе кода и декодера важно принимать во внимание свойства канала передачи данных. Например, в условиях канала с замираниями соотношение сигнал шум может опускаться до малого значения на небольшие интервалы времени. При этом ошибка жёсткой демодуляции может достигать значений в 40%. Эффективным выход из этой ситуации является использование декодера с мягким входом, но его скорость декодирования обычно значительно ниже аналогичного жёсткого декодера. В качестве компромисса демодулятор заменяет символы с высокой вероятностью ошибки на “стирания”. В этом случае декодер должен понимать такой вход и уметь эффективно исправлять как ошибки, так и стирания.

Мы будем рассматривать канал, в котором вероятность стирания не менее чем на порядок выше вероятности ошибки.

2. Постановка задачи

Рассмотрим РС-код [n,k,d]=[255,225,31] на полем $GF(2^8)$. Введём некоторые обозначения:

α - примитивный элемент поля,

ν - количество ошибок,

ρ - количество стираний,

$g = \nu + \rho$ - количество искажений,

$\mathbf{c} = (c_0, \ldots, c_{n-1})$ - кодовое слово,

$\mathbf{v} = (v_0, \ldots, v_{n-1})$ - принятое слово,

$\mathbf{e} = (e_0, \ldots, e_{n-1})$ - вектор ошибок,

$\mathbf{f} = (f_0, \ldots, f_{n-1})$ - вектор стираний.

$$\mathbf{v} = \mathbf{c} + \mathbf{e} + \mathbf{f}$$

Декодер должен, имея на входе вектор $\mathbf{v}, v_i \in GF(q) \cup \{\xi\}$, где ξ - символ стирания, дать на выходе либо переданное кодовое слово $\mathbf{c}$, либо неправильное кодовое слово $\tilde{\mathbf{c}}$, либо дать отказ от декодирования. При этом декодер будет “хорошим”, если вероятности последних двух событий будут малы.

Рассмотрим сначала случай отсутствия стираний.

При этом пространство $GF(q)^n$ разбивается на q^k+1 областей, первые q^k соответствуют различным кодовым словам, а последняя - отказу от декодирования. Обычно области, соответствующие кодовым словам, являются шаром, описанным вокруг кодового слова, так как это упрощает конструкцию декодера. Кроме того, будем рассматривать те декодеры, у которых радиусы этих шаров равны между собой.

В этом случае вероятность отказа от декодирования будет равна вероятности попадания вектора ошибки в соответствующую область, а вероятность правильного декодирования - в область, соответствующую нулевому кодовому слову.

Если в канале произошло ρ стираний, принятое кодовое слово из точки "превращается" в некоторую фигуру, имеющую диаметр ρ. В этом случае, если фигура пересекается только с одним "шаром ошибок", декодер выдаст в качестве результата декодирования центр этого шара. Потребуем, чтобы "фигура стираний" всегда пересекалась не более чем с одним "шаром ошибок". Это возможно только при условии: $2\nu+\rho<d$, где ν - количество ошибок.

Для того, чтобы уменьшить вероятность неправильного декодирования, наш декодер будет выдавать отказ от декодирования при $2\nu+\rho>2t$. При этом декодер примет неправильное решение, только если $2\nu+\rho>2d-2t$. А значит вероятности ошибки и отказа от декодирования определяются параметром t.

Для упрощения нахождения значений искажений (пункт 4 в нижеприведённом списке) отдельно ограничим количество ошибок и стираний величинами ν_{max} и ρ_{max} соответственно. Если хотя бы одна из них будет превышена, декодер откажется от декодирования.

Введём также следующие многочлены:

$$c(x)=c_{n-1}x^{n-1}+\ldots+c_1x+c_0$$

$$\nu(x)=\nu_{n-1}x^{n-1}+\ldots+\nu_1x+\nu_0$$

$$e(x)=e_{n-1}x^{n-1}+\ldots+e_1x+e_0$$

$$f(x)=f_{n-1}x^{n-1}+\ldots+f_1x+f_0$$

Введём локаторы ошибок X_i, стираний U_i и искажений L_i [2], а также их значения Y_i, Z_i и D_i соответственно.

В терминах локаторов ошибок и стираний можно сказать, что декодер должен по данным $\mathbf{v}$ и U_i определить X_i и D_i.

Рассмотрим основные шаги алгоритма декодирования:

1. Нахождение синдрома S_j и синдромного многочлена

$$S(x)=\sum_{j=1}^{2t}S_jx^{j-1}=\sum_{j=1}^{2t}\nu(\alpha^j)x^{j-1}.$$

2. Нахождение многочлена локаторов искажений $\Lambda(x)=\prod_{j=1}^{g}(1-X_jx)$ и многочлена значений искажений

$$\Omega(x)=S(x)\Lambda(x)(\mathrm{mod}\,x^g)=\sum_{i=1}^{g}Y_iX_i\prod_{j\neq i}(1-X_ix).$$

3. Вычисление локаторов ошибок.

4. Вычисление значений искажений.

Мы будем рассматривать различные реализации отдельных этапов и сравнивать их по количеству операций сложения и умножения, возможности параллельного выполнения и скорости работы на компьютере. К сожалению, не существует универсального параметра, позволяющего сравнивать алгоритмы по времени работы их реализаций. Существуют три элементарных операции, относительная скорость выполнения которых не определена: сложение в $GF(q)$, умножение в $GF(q)$ и умножение элементов $GF(p)\subset GF(q)$ на элементы $GF(q)$. Например, в случае прямого вычисления произведения элементов $GF(2^m)$ требуется m сложений. Использование таблиц логарифмов позволяет использовать единственную операцию сложения и 3 подстановки по таблице. В случае использования компьютера, подстановка по таблице может быть выполнена примерно за то же время, что и три операции сложения. Однако при реализации на ПЛИС использование памяти значительно увеличивает количество используемых элементов.

3. Преобразование Фурье

Многие этапы декодирования можно эффективно реализовать с помощью быстрого преобразования Фурье. Приведём 2 эквивалентные формулировки задачи вычисления преобразования Фурье:

$$V_i=\sum_{j=0}^{n-1}\alpha^{ij}v_j=v(\alpha^i),i=\overline{1,n} \qquad (1)$$

Вычисление синдрома является вычислением первых $2t$ компонент преобразования Фурье принятого слова (в дальнейшем мы будем называть эту операцию *частичным преобразованием Фурье*). Прямое вычисление синдрома по формуле 1 требует $2t(n-1)$ операций сложения и $2tn$ операций умножения. Это, с учётом параметров нашего кода, составляет $30*254=7620$ операций сложения и 7720 операций умножения.

Существует множество реализаций быстрого преобразования Фурье. К сожалению, большинство из них не дают возможности вычислить частичное преобразование Фурье за меньшее время. Рассмотрим наиболее известные из них.

Число $n=255$ имеет следующие простые множители: 3, 5, 17. Воспользовавшись алгоритмом Кули-Тьюки [2], мы можем выполнить преобразование Фурье за $255(3+5+17+2)=6885$ операций умножения. Алгоритм Гуда-Томаса [2] позволяет уменьшить это количество до 6375. Оба этих алгоритма сводят вычисление преобразования Фурье длины 255 к нескольким применениям преобразований

Фурье длины 3, 5 и 17. В алгоритме Гуда-Томаса исходный вектор записывается в таблицу на диагонали и её продолжении. Результат так же считывается "нелинейно". Это не позволяет нам вычислить частичное преобразование Фурье за меньшее число операций.

Но алгоритм Кули-Тьюки записывает входные значения в таблицу вдоль строк и столбцов. Это позволяет нам вычислять только первые 2 элемента преобразования Фурье длины 17. Это сокращает количество операций умножения до $255*(3+1)+255*(5+1)+255*2=3060$.

Алгоритм Ванга-Жу [7] так же позволяет вычислять преобразование Фурье. Он строит дерево из многочленов $q_{ij}(x)$, где i-высота текущего элемента (для листьев она равна 0). При этом $q_{ij}=\prod_{k=0}^{p-1} q_{i-1,jp+k}, i=\overline{1,m}, j=\overline{0,p^{m-i}-1}$. Кроме того, степень многочлена $q_{ij}(x)$ равна p^i, и $q_{ij}(x)=q_{i0}+Q_{ij}$, где Q_{ij} - некоторая константа. Для входного многочлена $v(x)$ вычисляется $r_{i-1,jp+k}(x)=r_{i,j}(x)\mathrm{mod}\,(q_{i-1,jp+k}(x)), r_{m,0}=v(x)$. В случае $m=2^r$ существует такая перестановка элементов $q_{0j}(x)$, что все коэффициенты многочленов $q_{i0}(x)$ будут принадлежать полю $GF(p)$.

В результате количество операций умножения элементов поля $GF(q)$ будет равно для нашего кода 1665, а количество операций умножения элементов $GF(p)\subset GF(q)$ на элементы $GF(q)$ равно 3712. Количество операций сложения равно 5377. По всем трём параметрам алгоритм Ванга-Жу быстрее алгоритмов Гуда-Томаса и Кули-Тьюки, но, в отличие от последнего, не предоставляет возможности вычислять частичное преобразование Фурье. В случае вычисления корней многочлена малой степени алгоритм Ванга-Жу позволяет сократить количество операций, благодаря вычислению только нижней части дерева.

Ещё один быстрый алгоритм преобразования Фурье описан в [5]. К сожалению, вычисление частичного преобразования Фурье не позволяет понизить число операций. Тем не менее данный алгоритм всё равно использует меньше операций умножения, чем все представленные ранее. Для вычисления преобразования Фурье длины 256 требуется лишь 1024 операции умножения.

Другим эффективным алгоритмом преобразования Фурье является циклотомический алгоритм [3]. Он основывается на разложении поля на циклотомические классы и группировке элементов исходного вектора. В результате вычисление преобразования Фурье сводится к умножению исходного вектора на двоичную и блочно-циркулянтную матрицы. Данный алгоритм требует $O(n\log n)$ памяти для хранения предвычесленных матриц, однако он использует значительно меньше операций сложения и умножения. Подробный обзор этого алгоритма не представлен в данной работе. Более подробно про применение циклотомического алгоритма можно прочитать в [4].

4. Многочлен локаторов искажений

Построим многочлен локаторов стираний: $\Psi(x)=\prod_{i=1}^{\rho}(1-xU_i)$. Воспользуемся итеративной процедурой, описанной в [6, раздел 3.3], для поиска многочлена локаторов ошибок и стираний $\Lambda(x)$ (в [6] используется обозначение $\Omega(x)$, но мы резервируем это обозначение для многочлена значений искажений). Внесём в эту процедуру небольшое изменение: ограничим степень многочлена $\Lambda(x)$ значением $\rho+v_{max}$.

Найдём многочлен локаторов ошибок $\tilde{\Lambda}(x)=\frac{\Lambda(x)}{\Psi(x)}$, $\deg\tilde{\Lambda}<v_{max}$. Таким образом, деление многочленов потребует $(\rho+v_{max}/2)v_{max}$ операций умножения, столько же операций сложения и v_{max} операций взятия обратного элемента.

Для случаев $v_{max}=2,3,4$ аналитические методы вычисления корней многочлена описаны в [3] и [1, с 272]. Мы не будем приводить точное количество операций для них. В программной реализации использовался переборный алгоритм, в силу простоты его реализации.

5. Многочлен значений искажений

Многочлен значений искажений

$$\Omega(x)=\sum_{k=0}^{g-1}=S(x)\Lambda(x)\mathrm{mod}\,x^g=\sum_{i=1}^{g}D_iL_i\prod_{j\neq i}(1-L_jx)$$

Можно рассматривать $\Omega(x)$ по модулю x^{2t}, как это описано в [2, Глава 7], но при этом коэффициенты $\Omega_k=0, k\geq g$. Для нахождения коэффициентов Ω_k требуется k умножения и k сложений. Тогда для вычисления всех коэффициентов необходимо $(g^2-g)/2$ умножений и $(g^2-g)/2$ сложений. Характерным значением параметра g является 28, что соответствует 2 ошибкам и 26 стираниям. При этом нам потребуется по 378 операций сложения и умножения.

Алгоритм Форни[2] позволяет найти значения искажений по следующей формуле:

$$D_i=\frac{\Omega(L_i^{-1})}{\Lambda'(L_i^{-1})}=\frac{L_i^{-1}\Omega(L_i^{-1})}{\prod_{j\neq i}(1-L_i^{-1}L_j)}=\frac{\Omega(L_i^{-1})}{\sum_{j=1}^{\lfloor(g-2)/2\rfloor}\Lambda_{2j+1}L_i^{-2j}},$$

где $\Lambda'(x)$ - производная многочлена локаторов искажений. Коэффициенты при нечётных членах $\Lambda'(x)$ равны нулю.

Оценим сложность вычисления значений искажений по этой формуле. Во-первых требуется $2g$ операций нахождения обратного элемента. Кроме того,

необходимо вычислить значения двух многочленов в g точках. Если считать их по схеме Горнера, для вычисления значения числителя понадобится $g-1$ умножений и $g-1$ сложений в поле $GF(q)$. На вычисление знаменателя - ещё $(g-2)/2$ умножений и $(g-2)/2$ сложений. Итого требуется $3g^2/2-g$ операций умножения и $3g^2/2-2g$ операций сложения. Подставив $g=28$ получим по 1148 операций сложения и умножения. Вместе с вычислением коэффициентов многочлена значений искажений это даёт по 2000 операций сложения и умножения.

Построим итеративную модификацию алгоритма Форни. Мы доказали, что значения искажений можно определить по формуле

$$D_i = \frac{\sum_{j=0}^{g-1} \overline{\Lambda}_{ij} S_{g-j}}{L_i \prod_{k \neq i} (L_i - L_k)} = \frac{\sum_{j=0}^{g-1} \overline{\Lambda}_{ij} S_{g-j}}{\sum_{j=0}^{g-1} \overline{\Lambda}_{ij} L_i^{g-j}}, \quad (2)$$

где $\overline{\Lambda}_i(x) = \prod_{j \neq i}(1+L_j x)$ - "усечённый" синдром, не содержащий локатора L_i.

Так же доказана следующая формула:

$$D_i = \frac{\sum_{j=0}^{i-1} \Lambda_j^{(i-1)} S_{i-j}^{(i)}}{L_i \prod_{j=1}^{i} (L_i - L_k)} = \frac{\sum_{j=0}^{i-1} \Lambda_j^{(i-1)} S_{i-j}^{(i)}}{\sum_{j=0}^{g-1} \Lambda_j^{(i-1)} L_i^{i-j}}, \quad (3)$$

где $\Lambda^{(i)}(x) = \prod_{k=1}^{i}(1+L_k x)$ - "частичный" многочлен локаторов искажений, и $S_j^{(i)}(x) = \sum_{k=1}^{i} L_k^i D_k$ - "частичный" синдром.

Формула 3 позволяет построить итеративную процедуру вычисления значений искажений. Данный результат получен совместно с А.А. Давыдовым. Пусть при вычислении многочлена локаторов искажений у нас сохранились все частичные многочлены $\Lambda^{(i)}(x)$.

Для $i = g, g-1, \ldots, 1$

1. Вычислим D_i по формуле 3.
2. $S_j^{(i-1)} = S_j^{(i)} - L_i^j D_i,\ j = 1, \ldots, i-1$
3. Повторить для следующего i.

Определим требуемое количество операций. Для вычисления значения числителя в формуле 3 требуется $i-1$ операций умножения и $i-1$ операций сложения, для знаменателя – ещё столько же. Вся дробь требует $2i-1$ операций умножения, $2i-2$ операций сложения и одна операция взятия обратного элемента. Всего требуется g^2 операций умножения, g^2-g операций сложения и g операций взятия обратного элемента. Для вычисления частичных синдромов требуется ещё по $(g^2-g)/2$ операций сложения и умножения.

Для $g=28$ потребуется 1134 операций умножения, 1176 операций сложений и 28 операции нахождения обратного элемента. Может показаться, что это больше, чем в алгоритме Форни, но для работы итеративного алгоритма не требуется вычислять коэффициенты многочлена значений искажений.

Ещё одним методом вычисления значений искажений является нахождение преобразования Фурье многочлена значений искажений. При использовании алгоритмов Кули-Тьюки и Ванга-Жу не удаётся достичь выигрыша в количестве операций, но можно избавиться от операции взятия обратного элемента. Кроме того, сложность алгоритма Кули-Тьюки является аффинной функцией от g с положительным свободным членом.

Значения многочлена на всех символах $GF(q)$ можно так же найти с помощью алгоритма описанного в [3, Раздел 3.1]. Оценка числа операций для этого случая не производилась.

6. Практическая реализация описанных алгоритмов

С целью проверки вышеуказанных расчётов была написана библиотека на языке Python. Были реализованы алгоритмы быстрого преобразования Фурье:

- модификация алгоритма Кули-Тьюки, описанная выше,
- алгоритм Ванга-Жу, также видоизменённый для получения частичного преобразования Фурье,
- алгоритм, описанный в [5].

Затем был реализован алгоритм Берлекэмпа-Месси и алгоритм Форни. Также была реализована итеративная модификация алгоритма Форни.

Подсчёт количества операций сложения, умножения и деления производился автоматически, за счёт гибкости объектной модели языка Python. Для всех модулей библиотеки были написаны подробные тесты. Оптимизация кода не проводилась.

Количество операций, необходимых для вычисления преобразования Фурье, подтвердило теоретические оценки. Использование итеративного алгоритма Форни позволило уменьшить число операций умножения с 3130 до 2877 на всю процедуру декодирования. Количество операций сложения и возведения в степень изменилось незначительно. Вычисление коэффициентов многочлена значений искажений и применение алгоритма Форни потребовало $406+1120=1526$ умножения. Для итеративного алгоритма Форни потребовалось 1218 умножений. Данный результат можно улучшить, положив $j_0=0$

и повторно использовав некоторые результаты вычислений.

7. Заключение

В статье описан алгоритм декодирования РС-кодов, исправляющий ошибки и стирания. На каждом этапе декодирования представлено несколько возможных реализаций. Для РС кода [255,225,31] приведена оценка количества операций сложений и умножений в поле $GF(q)$, а также умножения элементов $GF(p) \subset GF(q)$ на элемент $GF(q)$ и взятия обратного элемента, необходимых для выполнения каждого из представленных методов. Описана новая итеративная модификация алгоритма Форни, позволяющая уменьшить число операций. Была написана программная реализация всех описанных алгоритмов. С её помощью проведена проверка теоретических оценок на число операций.

Список литературы

[1] Ф.Д. Мак-Вильямс and Н.Д.А. Слоэн. Теория кодов, исправляющих ошибки. *М.: Связь*, 1979.

[2] Р. Блейхут. *Теория и практика кодов, контролирующих ошибки.* Мир, 1986.

[3] С.В. Федоренко. *Методы быстрого декодирования линейных блоковых кодов.* ГУАП СПб., 2008.

[4] N. Chen and Z. Yan. Reduced-complexity Reed–Solomon decoders based on cyclotomic FFTs. *Signal Processing Letters, IEEE*, 16(4):279–282, 2009.

[5] S. Gao and T. Mateer. Additive fast Fourier transforms over finite fields. *Information Theory, IEEE Transactions on*, 56(12):6265–6272, 2010.

[6] G. Schmidt. *Algebraic Decoding Beyond Half the Minimum Distance Based on Shift Register Synthesis.* VDI-Verl., 2007.

[7] Y. Wang and X. Zhu. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. *Selected Areas in Communications, IEEE Journal on*, 6(3):572–577, 1988.