УДК 519.7, 681.3.05, 681.325.3

А.Э. Маевский, А.М. Пеленицын

РЕАЛИЗАЦИЯ ПРОГРАММНОГО АЛГЕБРО-ГЕОМЕТРИЧЕСКОГО КОДЕКА С ПРИМЕНЕНИЕМ АЛГОРИТМА САКАТЫ

Для защиты передаваемой информации по зашумленным каналам связи используются помехоустойчивые коды. При группировке ошибок в канале в пачки удобно иметь коды большой длины над полями сравнительно малой мощности. Указанным свойством обладают многие классы алгебро-геометрических кодов (АГ-коды) [1], но их экспериментальное исследование затруднено из-за высокой теоретической сложности. В [2,3] предложен класс сравнительно простых АГ-кодов, обобщающих коды Рида-Соломона и имеющих параметры, близкие к границе Синглтона.

Пусть **k** – конечное поле мощности q. Для некоторого целого числа s ($<q$) рассмотрим линейное векторное пространство $V_q(s)$ размерности $(s+1)(s+2)/2$, состоящее из всех однородных многочленов от трех переменных степени s над полем **k** вместе с нулевым многочленом. Пусть C_F – неособая кривая рода g в проективном пространстве $P^2(\mathbf{k})$, определяемая абсолютно неприводимым однородным многочленом $F \in \mathbf{k}[x,y,z]$ степени m. Через $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ обозначим множество всех ее проективных точек над полем **k**, представителей которых выберем стандартным образом: самая правая ненулевая координата представителя положительна. Вслед за [2] *АГ-код* длины $n = |\mathbf{P}|$, размерности $k(s) = mj-g+1$ ($s \in [m,n/m]$) или $k(s) = (s+1)(s+2)/2$ ($s \in [0,m-1]$), с конструктивным кодовым расстоянием $d(s) = n - ms$, определим как множество

$$AG_{\mathbf{P}}(s) = \{ (f(P_1), f(P_2), \ldots, f(P_n)) \mid f \in V_q(s), P_i \in \mathbf{P} \}.$$

В работе [2] приведен алгоритм декодирования кода $AG_{\mathbf{P}}(s)$, являющийся обобщением известного алгоритма Питерсона декодирования кодов Рида-Соломона и позволяющий исправлять до $d(s)/2-m^2/4$ ошибок.

Для изучения помехоустойчивости рассматриваемого класса АГ-кодов в работе [4] построена и программно реализована имитационная модель цифрового защищенного канала передачи данных, позволяющая закодировать с помощью АГ-кода произвольный файл, имитировать его передачу по зашумленному каналу, и затем декодировать. К параметрам модели относится вероятность ошибки на бит передаваемой информации и параметры АГ-кодека: поле k, кривая C_F, величина s. Канал передачи данных рассматривается как стационарный двоичный симметричный без памяти и без стираний с аддитивной помехой, моделируемой как равномерная периодическая случайная величина. Программная реализация модели использует модульный принцип построения, что позволяет сравнительно легко изменять различные составные части, например, вместо генератора помех, распределенных по равномерному закону, использовать другой генератор помех.

В данной работе построена программная реализация нового модуля декодирования АГ-кода, основанного на алгоритме декодирования из работы [3]. По сравнению с алгоритмом декодирования, опубликованным в [2], этот алгоритм позволяет исправить до $d(s)/2-m^2/8+m/4-9/8$ ошибок за счет применения вместо метода Гаусса решения системы однородных линейных уравнений алгоритм Сака-

ты построения аннулирующего многочлена двумерной рекуррентной последовательности над k. Таким образом, основным моментом при построении программной реализации модуля декодирования стала реализация алгоритма Сакаты. Необходимо отметить, что в работе Сакаты [5], посвященной обоснованию алгоритма, сам алгоритм в явном виде отсутствует: его заменяет последовательность рассуждений, лемм и теорем. Поэтому в данной работе на основе [5] построено формализованное, полное и обоснованное описание алгоритма Сакаты.

Так как алгоритм Сакаты может быть применен не только в задаче декодирования АГ-кодов, его реализация выполнена в виде отдельного модуля на языке программирования C++ в соответствии с действующим стандартом редакции 1998 года, что позволило, в свою очередь, подготовить варианты реализации для платформ Win32 и Linux. Сборки выполнялись с помощью имевшихся свободных средств разработки: Microsoft Visual Studio Express Edition 2005 для платформы Win32 и gcc 4.1.3 с использованием NetBeans IDE для Linux. При построении модуля были использованы следующие открытые библиотеки: NTL, предоставляющая функции работы с конечными полями [6], и Boost, облегчающая создание переносимых программ, ориентированных на современные концепции C++-дизайна [7]. Обе библиотеки используются сообществом на протяжении многих лет и хорошо зарекомендовали себя при решении соответствующих задач. Библиотека NTL обладает одной интересной особенностью в свете современных тенденций в программировании на C++. Она содержит разные классы для разных типов конечных полей (по мощности: простые, степени двойки, степени простого). Казалось бы, это должно создавать неудобства при написании кода, предназначенного для работы с конечными полями разных типов. Однако тот факт, что ее классы и сопутствующие свободные функции выполнены в соответствии с общими соглашениями или в терминах шаблонного программирования C++ удовлетворяют общим неявным интерфейсам, позволяет писать шаблонный код C++, который способен работать с каждым из классов NTL. Этот факт является тем более удивительным, что NTL создавалась в начале 90-х, когда вся мощь такого инструмента, как шаблоны C++, еще не была осознана сообществом.

Был разработан ряд методов и классов, реализующих представление и арифметические операции для следующих математических объектов:

– многочлен двух переменных;

– множество последовательностей элементов конечного поля, индексированных неотрицательными целыми числами;

– множество многочленов двух переменных.

Кратко опишем основные операции. Для многочленов двух переменных реализованы стандартные операции сложения и умножения на скаляр, а также операции умножения на моном и умножения на последовательность. Поскольку при работе с многочленами нескольких переменных возникает вопрос выбора упорядочения мономов, были выбраны следующие два упорядочения, применимые также к последовательности элементов конечного поля, индексированной неотрицательными целыми числами: полное мономиальное, представляющее собой градуированное антилексикографическое упорядочение, и частичное, основанное на отношении делимости мономов. Следует отметить, что операция «<» языка C++ является перегружаемой, соответствует концепции Строго Слабого Упорядочения [8], и весьма удобна для использования. Для множества многочленов двух переменных реализована операция Берлекэмпа, работающая в соответствии с алгоритмом Сакаты.

Одними из самых трудоемким в реализации идеи Сакаты стали многочисленные манипуляции с подмножествами текущего «минимального множества» мно-

гочленов для заданного двумерного массива (массива синдромов в случае использования в декодере). Само минимальное множество достаточно легко реализуется на базе стандартного шаблона C++ std::set, учитывая, что для полиномов можно перегрузить операцию «<» в соответствии с упорядочением внутри минимального множества, которое является инвариантой на всех шагах алгоритма, в том числе при изменении этого множества (указанное упорядочение изначально продиктовано видом базиса нульмерного идеала многочленов, дающих рекуррентные соотношения для имеющегося двумерного массива). Эффективность переупорядочивания при изменении минимального множества, таким образом, достигается за счет реализации стандартной библиотеки. Сложнее дело обстоит с реализацией разбиения минимального множества на два подмножества: многочленов, которые «действительны» на данном шаге алгоритма (то есть являются частью минимального множества для рассматриваемого на данном шаге отрезка исходного массива), и многочлены, которые «недействительны» и подлежат замене. Это разбиение происходит на каждом шаге и потому должно быть максимально эффективным. Принципиальным техническим решением в этом вопросе стал «отказ от владения»: структура данных, реализующая подмножества, содержит только указатели (на самом деле, итераторы) на многочлены из основного (минимального) множества, что позволяет несколько сэкономить накладные расходы при ее заполнении.

С помощью программной реализации модели канала связи был проведен ряд экспериментов, в ходе которых подтвердилась теоретическая оценка количества исправляемых новым декодером ошибок, а также было изучено его поведение в случаях превышения количеством ошибок корректирующей способности. Проведенные эксперименты могут служить практическим критерием выбора тех или иных параметров АГ-кодов в зависимости от интенсивности ошибок в канале. В частности, благодаря экспериментам стало возможным установить, что каскадирование АГ-кодеков эффективно лишь в тех случаях, когда число исправляемых ошибок в следующем каскаде превышает длину АГ-кода в предыдущем каскаде.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Влэдуц С.Г., Ногин Д.Ю., Цфасман М.А.* Алгеброгеометрические коды. Основные понятия. – М.: МЦНМО, 2003.

2. *Justesen J., Larsen K.J., Havemose A., Jensen H.E., Hoholdt T.* Construction and decoding of a class of algebraic geometry codes // IEEE Transactions on Information Theory. – 1989, vol. 35, pp. 811-821.

3. *Justesen J., Larsen K.J., Jensen H.E., Hoholdt T.* Fast decoding of codes from algebraic plane curves // IEEE Transactions on Information Theory. – 1992, vol. 38, pp. 111-119.

4. *Маевский А.Э.* Некоторые алгебро-геометрические кодеки и их программная реализация // Труды участников международной школы-семинара по геометрии и анализу памяти Н.В.Ефимова. – Ростов-на-Дону: ООО «ЦВВР», 2004.

5. *Sakata S.* Finding a minimal set of linear recurring relations capable of generating a given finite two–dimensional array // Journal of Symbolic Computations. – 1988, vol. 5, pp. 321–337.

6. *Shoup V.* NTL: A library for doing number theory. 1996-2008. http://www.shoup.net/ntl

7. BOOST C++ Library. 1998-2008. http://www.boost.org

8. *Остерн М.Г.* Обобщенное программирование и STL: Использование и наращивание стандартной библиотеки шаблонов C++. – СПб.: Невский диалект, 2004.