Московский государственный университет
имени М.В.Ломоносова

Механико-математический факультет

Кафедра Математической теории интеллектуальных систем

КУРСОВАЯ РАБОТА

Порождение равномерного распределения на множестве правильных семейств функций
Generation of uniform distribution on the set of proper families of functions

Выполнил студент 4 курса:
Р.А.Жигляев
Научный руководитель:
к.ф.-м.н., с.н.с. А.В.Галатенко

Москва, 2021

Аннотация

В работе рассматривается алгоритм порождения равномерного распределения на множестве правильных семейств заданного порядка, построение квазигрупп на основании порожденного семейства функций и проверка квазигруппы на полиномиальную полноту. Приводится оценка сложности алгоритма и программная реализация.

The paper considers an algorithm for generating a uniform distribution on the set of proper families of a given order, the construction of quasigroups based on of the generated family of functions and checking the quasigroup for polynomial completeness. We estimate the algorithm complexity and describe the software implementation.

1. Введение

Одним из важных направлений в создании криптографических алгоритмов [1] является изучение конечных квазигрупп. Известно, что почти все квазигруппы обладают свойством полиномиальной полноты [2], или, что то же самое, являются простыми и неафинными [3]. Это значит, что если взять случайную квазигруппу, то с большой вероятностью она окажется подходящим вариантом. Однако, для большей стойкости, необходимо получать квазигруппы достаточно больших порядков. В таком случае, работа с квазигруппами, заданными своими латинскими квадратами, может затрудниться, в связи с ограниченностью памяти компьютера. Поэтому, важно изучать более оптимальные способы задания. Например, в [4], а далее и в [5], [6], была развита идея задания квазигрупп при помощи правильных семейств функций. Обобщая результаты работ [7] и [8] можно получить МСМС-процедуру порождения равномерного распределения на множестве правильных семейств функций заданного порядка.

Задачами данной работы является описание указанного выше МСМС алгоритма, оценка сложности его шага и приведение программной реализации на языке C++. Проводится проверка равномерности генерации. Далее полученное семейство функций преобразуется в квазигруппу, а полученная квазигруппа проверяется на полиномиальную полноту. Все эксперименты проводились на компьютере с процессором Intel(R) Core(TM) i5-5200U 2.20GHz, 4 ГБ ОЗУ.

В результате будет представлено подробное описание алгоритма и программы и получена оценка сложности одного шага алгоритма.

Дальнейшая часть работы устроена следующим образом: в разделе 2 даны основные определения; раздел 3 посвящен описанию алгоритма генерации правильного семейства функций; в разделе 4 описывается программная реализация и оценка сложности алгоритма; раздел 5 посвящен экспериментам с программой; в разделе 6 подводятся итоги работы.

2. Основные определения

Определение 1. Конечное множество Q, на котором задана бинарная операция умножения f_Q, такая, что для любых элементов $a, b \in Q$ уравнения $ax = b$ и $ya = b$ однозначно разрешимы в Q, называется конечной квазигруппой.

Определение 2. Квазигруппа (Q, f_Q) называется аффинной, если на множестве Q можно ввести структуру абелевой группы $(Q, +)$, такую, что существуют автоморфизмы α, β группы $(Q, +)$ и элемент $c \in Q$, для которых выполнено тождество

$$f_Q(x, y) = \alpha(x) + \beta(y) + c.$$

Определение 3. Квазигруппа (Q, f_Q) называется простой, если операция f_Q не сохраняет никакое нетривиальное разбиение Q.

Определение 4. Семейство $(g_1, ..., g_n)$ n-местных функций k-значной логики будем называть правильным, если для любой пары различных наборов $s = (s_1, ..., s_n)$, $t = (t_1, ..., t_n)$ из E_k^n найдется номер $1 \leq i \leq n$, такой, что $s_i \neq t_i$, но $g_i(s) = g_i(t)$.

Рассмотрим квазигруппу с носителем Q, $|Q| = k^n$, где k, n натуральные числа, $k \geq 2$. Операцию умножения такой квазигруппы можно задать с помощью правильного семейства $(g_1, ..., g_n)$ n-местных функций k-значной логики по следующему правилу:

Пронумеруем все элементы квазигруппы числами от 0 до $k^n - 1$, а каждому номеру сопоставим его кодировку в k-ичной системе счисления. Возьмем кодировки двух элементов $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n)$. Тогда кодировку произведения $f(x, y) = (f_1, ..., f_n)$ этих элементов можно получить по формулам

$$f_i = x_i + y_i + g_i(\pi_1(x_1, y_1), ..., \pi_n(x_n, y_n)),\ 1 \leq i \leq n$$

Здесь сложение ведется по модулю k, а функции $\pi_1, ..., \pi_n$ — произвольные двухместные функции k-значной логики.

3. Описание алгоритма

Как было сказано выше, для порождения правильного семейства будем использовать результаты работ [7] и [8].

На старте работы алгоритма возьмем произвольное правильное семейство $(g_1, ..., g_n)$ n-местных функций k-значной логики. Сгодится, например, такое семейство, что $g_i \equiv \text{const}$, $1 \leq i \leq n$. Здесь можно взять любые случайные константы от 0 до $k-1$, причем, для каждой функции свою. Несложно видеть, что семейство таких функций является правильным.

Пусть на текущем шаге имеется некоторое правильное семейство $(g_1, ..., g_n)$. Получим из него новое правильное семейство по следующему алгоритму:

1. Сгенерируем случайное число $1 \leq i \leq n$.

2. Поменяем местами функции g_i с g_n и переменные x_i с x_n. Далее во всех записях будем подразумевать, что перестановка уже проведена.

3. Сформируем k семейств $G_0 = (g_1^0, ..., g_{n-1}^0), ..., G_{k-1} = (g_1^{k-1}, ..., g_{n-1}^{k-1})$. Здесь $g_j^m(x_1, ..., x_{n-1}) = g_j(x_1, ..., x_{n-1}, m)$. Иными словами, мы удаляем последнию переменную и последнюю функцию и каждое семейство G_m составляем из оставшихся функций, которым на место последней переменной поставили число m, $0 \leq m \leq k-1$.

4. Зададим первые $n-1$ функцию нового семейства по формуле:

$$g_j^{'}(x_1, ..., x_n) = \bigvee_{m=0}^{k-1}(I_{x_n}(m) \wedge g_j^m(x_1, ..., x_{n-1})),\ 1 \leq j \leq n-1$$

Здесь под $\vee$ будем подразумевать операцию взятия максимума, под $\wedge$ взятие минимума, а функции $I_{x_n}(m)$ имеют вид:

$$I_{x_n}(m) = \begin{cases} k-1, & x_n = m \\ 0, & \text{иначе} \end{cases}$$

5. Построим граф, в котором вершинами будут всевозможные наборы длины $n-1$ с элементами из E_k^{n-1}. Две различные вершины $s = (s_1, ..., s_{n-1})$ и $t = (t_1, ..., t_{n-1})$ соединим ребром, если можно найти такие номера $0 \leq j < m \leq k-1$, что $G_j(s)$ отличается от $G_m(t)$ во всех позициях, в которых s отличается от t.

6. Вычислим компоненты связности построенного графа.

7. На каждой из компонент связности будем задавать $g_n^{'}$ случайным значением от 0 до $k-1$ независимо друг от друга.

После выполнения всех пунктов мы получим новое правильное семейство функций $(g_1^{'}, ..., g_n^{'})$ и с новым семейством переходим к следующему шагу.

После выполнения необходимого числа шагов мы получим случайное правильное семейство функций. В качестве необходимого числа шагов в дальнейшем будем брать порядок квазигруппы, возведенный в квадрат. Как будет видно из экспериментов такого числа шагов будет достаточно.

Утверждение 1. Описанный алгоритм порождает равномерное распределение на множестве правильных семейств n-местных функций k-значной логики.

Доказательство вытекает из результатов работ [7] и [8].

4. Описание программной реализации

Семейство функций будем хранить в памяти компьютера как двумерный массив. По одной размерности будем индексировать сами функции семейства, а по другой размерности будем хранить значения конкретной функции на всевозможных наборах из n чисел от 0 до $k-1$. При этом индексировать значения функции будем не самими наборами, а их десятичным представлением. В связи с этим, потребуется реализовать два метода — для кодирования чисел из

десятичной системы в k-ичную и обратно. Оба метода несложно реализовать за $O(n)$.

На перестановку i-й функции с последней и i-й переменной с последней уйдет $2n^2k^n + k^n + nk^n$ действий, а так же потребуется выделить память под nk^n элементов на время проведения перестановки. Память выделяется под новый вариант семейства, который формируется из n функций, имеющих по k^n значений. Каждое значение необходимо преобразовать в k-ичную систему и поменять местами i-й разряд с последним. Далее необходимо поменять k^n значений двух функций, а после этого, в память, выделенную под исходное семейство, поместить новое.

Набор семейств функций G_m, $0 \leq m \leq k-1$ представляет собой трехмерный массив из $k(n-1)k^{n-1}$ элементов. По одной размерности индексируются сами семейства, а две других — есть двумерный массив, хранящий семейство функций, аналогично исходному, но уже на наборах из $n-1$ числа.

Сформировать такой набор семейств можно не более чем за $k(n-1)k^{n-1}(3n-2)$ действий. Действительно: k штук семейств, каждое из которых состоит из $n-1$ функции, на каждой функции каждого семейства необходимо задать k^{n-1} значений. При этом, задавая значения функции, нам нужно получить из десятичного представления значения его k-ичную кодировку из $n-1$ символа, образовать на основе этой кодировке новую, состоящую уже из n элементов, поставив на последнюю позицию значение m, и уже новую k-ичную кодировку перевести обратно в десятичную. На это потребуется не более $3n-2$ действий.

На основании построенного набора G_m начинаем строить $(g_1^{'}, ..., g_{n-1}^{'})$. Это можно осуществить не более чем за $(n-1)k^n(3n-2+k)$ действий: для каждой из $n-1$ функций необходимо вычислить значение на k^n значениях. При этом, во время вычисления набора, в функцию $I_{x_n}(m)$ подставляется значение переменной x_n, для вычисления которой нужно получить k-ичную кодировку текущего значения, что делается не более чем за n шагов. Далее из этого набора нужно взять первые $n-1$ элемент и новый набор из $n-1$ элемента перевести в десятичное представление. Также, при вычислении каждого значения, потребуется k вычислений максимума из минимумов.

Далее, необходимо построить граф. Для этого нужно перебрать все пары наборов из E_k^{n-1} и для каждого из наборов вычислить k-ичные кодировки этой пары, а затем перебрать все пары семейств из набора G_m, $0 \leq m \leq k-1$ и на каждой паре сравнивать наборы из $n-1$ элемента. Итого, на это потребуется $\frac{k^{n-1}(k^{n-1}-1)}{2}(n-1)(\frac{k(k-1)}{2}+2)$ действий. Также, в процессе построения графа, будем задавать список смежностей, для дальнейшего применения алгоритма обхода графа в глубину и формирования компонент связности. Список смежностей будет содержать столько же элементов, сколько существует рёбер в построенном графе. Это не более чем $\frac{k^{n-1}(k^{n-1}-1)}{2}$ элементов.

Для вычисления компонент связности проинициализируем массив из k^{n-1} булевых переменных значением false. Каждое из значений будет обозначать посетили ли мы ту или иную вершину графа. После этого, обойдем граф и зададим значения функции $g_n{}'$ на каждой из компонент случайной константой. Учитывая, что сложность обхода графа в глубину линейно зависит от суммы

числа вершин и числа ребер графа, а инициализировать функцию придется на k^n наборах, приходим к выводу, что на всю процедуру уйдет не более $k^{n-1} + \frac{k^n(k^n-1)}{2} + k^n$ действий.

Итого, весь шаг завершит работу не более чем за

$$(8n^2 - 9n + 4)k^n + (n-1)k^{n+1} + 2k^n + k^{n-1} + \frac{k^{n-1}(k^{n-1}-1)}{2}(1 + (n-1)(\frac{k(k-1)}{2} + 2))$$

действий. Отсюда получаем, что временная сложность алгоритма $O(nk^{2n})$. Суммарный объем дополнительно выделенной памяти за весь шаг не превышает $nk^n + k(n-1)k^{n-1} + \frac{k^{n-1}(k^{n-1}-1)}{2} + 2k^{n-1}$ элементов. Следовательно, сложность по памяти составляет $O(k^{2n})$. Обозначив порядок квазигруппы за $p = k^n$, получаем следующее:

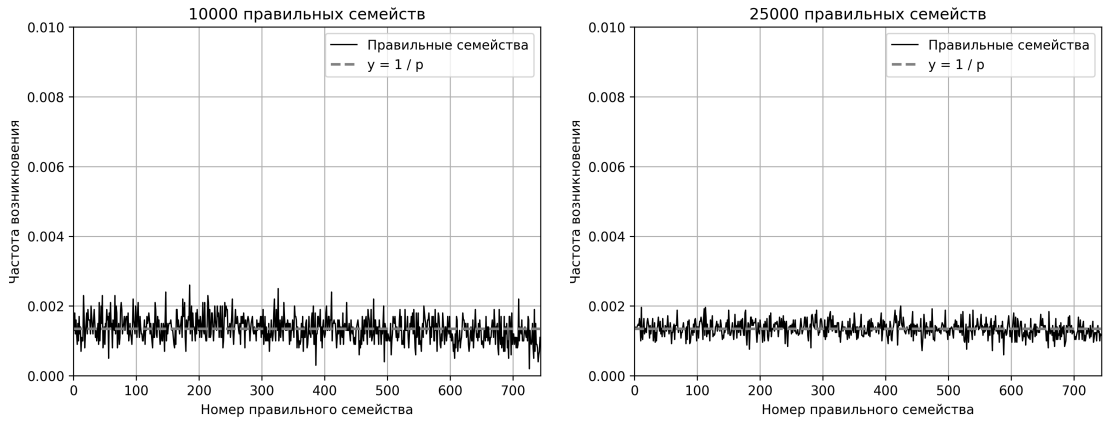
Утверждение 2. Один шаг алгоритма имеет временную сложность $O(p^2 \log_k p)$, общая временная сложность высчитывается как произведение числа шагов и сложности одного шага. Сложность по памяти как одного шага алгоритма, так и всего алгоритма в целом, составляет $O(p^2)$.

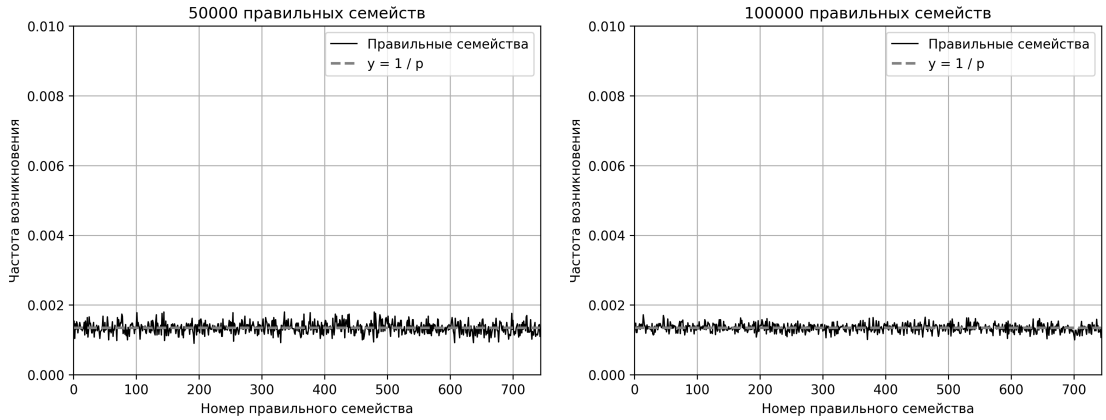
5. Эксперименты

Над программой было поставлено два эксперемента. В первом, были взяты достаточно маленькие значения для k и n и генерировалось большое число правильных семейств с целью убедиться в равномерности генерации на множестве всех правильных семейств. Второй эксперимент посвящен проверке квазигрупп, соответствующих порожденным правильным семействам, на полиномиальную полноту.

5.1 Первый эксперимент

Параметрами были взяты $k = 2$, $n = 3$. Всего было 4 различных прогона тестов, на 10000, 25000, 50000, 100000 генераций. Во всех прогонах были получены все 744 семейства порядка 3. Далее замерялись отношения числа возникновения каждого конкретного семейства к их общему числу. На графиках ниже показаны частоты возникновения семейств и проведена прямая $y = 1/p$, где p - количество всех правильных семейств порядка 3.





Как видно из графиков, с ростом числа сгенерированных семейств, частоты возникновения каждого семейства сходятся к $1/p$. Этот факт наглядно показывает равномерность генерации.

Далее, на выборке, полученной во время эксперимента на 100000 генераций, проверялась гипотеза о том, что выборка взята из равномерного дискретного распределения. Проверка гипотезы осуществлялась с помощью критерия хи-квадрат, реализованного в библиотеке SciPy языка программирования Python. В результате было получено P-значение равное 0.584431, что позволяет, например, принять гипотезу о равномернности распределения с уровнем значимости 0.05.

5.2 Второй эксперимент

Параметрами были взяты $k = 4$, $n = 3$. После генерации правильное семейство преобразовывалось в квазигруппу. При этом, функции π_i генерировались случайным образом. Всего было сгенерировано 2500 квазигрупп порядка 64. Каждая полученная квазигруппа проверялась на простоту и афинность. Как и ожидалось, все полученные квазигруппы были неафинными, однако, лишь 20 из них оказались простыми. Остальные не обладают свойством простоты, а следовательно и не являются полиномиально полными.

6. Заключение

Как видно из результатов работы, предложенная MCMC процедура генерации правильных семейств действительно задает равномерное порождение на множестве всех правильных семейств функций. Однако, латинские квадраты, получаемые из сгенерированных правильных семейств, приемущественно оказывается непростыми, а следовательно, не являются полиномиально полными.

В дальнейшем планируется провести оптимизацию шага алгоритма, точно определить количество шагов, необходимых для равномерной генерации, а также провести поиск классов семейств, преимущественно задающих полиномиально полные квазигруппы.

Список литературы

[1] Глухов М. М. О применениях квазигрупп в криптографии // Прикладная дискретная математика. 2008. № 2. С. 28–32.

[2] P. J. Cameron Almost all quasigroups have rank 2. Discrete Mathematics. 1992. Vol. 106–107. P. 111-115.

[3] J. Hagemann, C. Herrmann Arithmetical locally equational classes and representation of partial functions. Universal Algebra, Colloquia Mathematica Societatis János Bolyai. 1982. Vol 29. P. 345–360.

[4] Носов В. А. Построение классов латинских квадратов в булевой базе данных // Интеллектуальные системы. 1999. Том 4, № 3–4. С. 307–320.

[5] Nosov V. A., Pankratiev A. E. Latin squares over Abelian groups // Journal of Mathematical Sciences. 2008. Vol. 149, № 3. P. 1230–1234.

[6] Galatenko A.V., Nosov V. A., Pankratiev A. E. Latin squares over quasigroups // Lobachevskii Journal of Mathematics. 2020. Vol. 41. P. 194–203.

[7] Галатенко А. В., Носов В. А., Панкратьев А. Е. Об одном алгоритме построения правильных семейств функций // Материалы XVIII Международной конференции «Алгебра, теория чисел и дискретная геометрия: современные проблемы, приложения и проблемы истории», Тула, 2020, С. 142–146.

[8] Schurr I. A. Unique sink orientations of cubes. — Doctoral Thesis, ETH Zurich, 2004.