Автор: Казарян Михаил

Теоретические сведения

Краткое собрание теоретических сведений о алгоритмах генераторов (t,m,s)-сетей, и структур, понятий, необходимых для тестов генераторов.

Алгоритм семейства генераторов joe-kuo:

1. Начальная реализация

Алгоритм генерации последовательности Соболя объяснён в [2]. Здесь же мы дадим краткое описание деталям. Чтобы сгенерировать j-ю компоненту точки в последовательности Соболя, мы должны выбрать примитивный многочлен некой степени s_j в поле Z_2

$$x^{s_j} + a_{1.j}x^{s_j-1} + a_{2.j}x^{s_j-2} + \cdots + a_{s_j-1,j}x + 1, (1)$$

где наши коэффициенты $a_{1.j}, a_{2.j}, a_{3.j} \ldots a_{s_j-1,j}$ равны либо 0, либо 1.

Мы определяем последовательность положительных целых $\{m_{1,j}, m_{2,j} \ldots\}$ рекуррентным соотношением:

$$m_{k,j} \coloneqq 2a_{1.j}m_{k-1.j} \oplus 2^2a_{2.j}m_{k-2.j} \oplus \ldots \oplus 2^{s_j-1}a_{s_j-1.j}m_{k-s_j+1.j} \oplus 2^{s_j}m_{k-s_j.j} \oplus m_{k-s_j.j}, (2)$$

где $\oplus$ - оператор сложения по модулю 2 (исключающее «или»).

Начальные значения $m_{1,j}, m_{2,j}, \ldots, m_{s_j,j}$ могут быть любыми, но с условием, что $m_{k,j}$, $1 \le k \le s_j$, нечётное и меньше 2^k.

Направления $\{v_{1,j}, v_{2,j}, \ldots\}$ определяются по формуле:

$$v_{k,j} \coloneqq \frac{m_{k,j}}{2^k}$$

Тогда $x_{i,j}$,j-я компонента i −й точки в последовательности Соболева, задается формулой:

$$x_{i,j} \coloneqq i_1v_{1.j} \oplus i_2v_{2.j} \oplus \ldots, (3)$$

где i_k - это -я цифра справа, если i записана в двоичной форме $i = (\ldots i_3i_2i_1)_2$. В дальнейшем мы будем использовать $(\cdot)_2$ для обозначения двоичного представления номеров.

Например, при $s_j = 3$, $a_{1,j} = 0$ и $a_{2,j} = 1$ у нас имеется примитивный многочлен $x^3 + x + 1$. Начиная с $m_{1,j} = 1$, $m_{2,j} = 3$, и $m_{3,j} = 7$, мы

используем формулу (2) чтобы получить значения $m_{4,j} = 5$ $m_{5,j} = 7$, и т.д. Отсюда мы можем вычислить направления:

$v_{1,j} = (0.1)_2, v_{2,j} = (0.11)_2, v_{3,j} = (0.111)_2, v_{4,j} = (0.0101)_2, v_{5,j} = (0.00111)_2$,…..

Исходя из (3), мы получаем -е компоненты первых нескольких точек:

$0 = (0)_1$	$x_{0,j} = 0$
$1 = (1)_2$	$x_{1,j} = (0.1)_2 = 0.5$
$2 = (10)_2$	$x_{2,j} = (0.11)_2 = 0.75$
$3 = (11)_2$	$x_{3,j} = (0.1)_2 \oplus (0.11)_2 = (0.01)_2 = 0.25$
$4 = (100)_2$	$x_{4,j} = (0.111)_2 = 0.875$
$5 = (101)_2$	$x_{5,j} = (0.1)_2 \oplus (0.111)_2 = (0.011)_2 = 0.375$

2. Реализация кода Грэя

Формула (3) соответствует первоначальной реализации Соболя. Более эффективная реализация, которая была предложена Антоновым и Салеевым, получила название код Грэя.

Код Грэя (в двоичном представлении) от целого i определяется как

$$gray(i) \coloneqq i \oplus \left\lfloor \frac{i}{2} \right\rfloor = (\dots i_3 i_2 i_1)_2 \oplus (\dots i_4 i_3 i_2)_2$$

Он обладает таким свойством, что двоичные представления (i) и $(i-1)$ отличаются только на одну позицию, а именно, индекс первой цифры справа от 0 в двоичном представлении $(i-1)$

i	$gray(i)$
$0=(0000)_2$	$(0000)_2=0$
$1=(0001)_2$	$(0001)_2=1$
$2=(0010)_2$	$(0011)_2=3$
$3=(0011)_2$	$(0010)_2=2$
$4=(0100)_2$	$(0110)_2=6$
$5=(0101)_2$	$(0111)_2=7$
$6=(0110)_2$	$(0101)_2=5$
$7=(0111)_2$	$(0100)_2=4$
$8=(1000)_2$	$(1100)_2=12$

$9=(1001)_2$	$(1101)_2=13$
$10=(1010)_2$	$(1111)_2=15$
$11=(1011)_2$	$(1110)_2=14$
$12=(1100)_2$	$(1010)_2=10$
$13=(1101)_2$	$(1011)_2=11$
$14=(1110)_2$	$(1001)_2=9$
$15=(1111)_2$	$(1000)_2=8$

Исходя из таблицы, можно заметить, что код Грэя - это просто переупорядочение неотрицательных целых чисел в каждом блоке 2^m чисел для $m = 0,1, \dots$.

Вместо (3), мы генерируем точки Соболя, используя формулу

$$\overline{x}_{i,j} \coloneqq g_{i,1}v_{1,j} \oplus g_{i,2}v_{2,j} \oplus \dots, (4)$$

где $g_{i,k}$- это -я цифра справа от i в коде Грея в двоичном представлении, то есть, $gray(i) = (\dots g_{i,3}, g_{i,2}, g_{i,1})_2$. Аналогично, поскольку (i) и $(i-1)$ отличаются на одну позицию, мы можем сгенерировать точки рекурсивно, используя формулу

$$\overline{x}_{0,j} \coloneqq 0 \text{ и} \overline{x}_{i,j} \coloneqq \overline{x}_{i-1,j} \oplus v_{c_{i-1},j}, (5)$$

где c_i является индексом первой цифры 0 справа в двоичном представлении $i = (\dots i_3 i_2 i_1)_2$. Мы имеем $c_0 = 1$, $c_1 = 2$, $c_3 = 3$, $c_4 = 1$, $c_5 = 2$, и т.д.

С реализацией кода Грэя мы просто получаем точки в другом порядке, сохраняя при этом их свойства однородности. Это связано с тем, что каждый блок из 2^m точек для $m = 0,1, \dots$ аналогичен первоначальной реализации. Отметим, что (4) и (5) порождают одинаковую последовательность; (4) позволяет начать с любой позиции в последовательности, в то время как (5) является рекурсивной и более вычислительно эффективной формулой.

3. Примитивные многочлены и числа направленности

Следуя ограничениям, описанным в [2], мы определяем коэффициенты примитивного многочлена (1) с целыми числами

$$a_j \coloneqq (a_{1,j}a_{2,j} \dots a_{s_j-1,j})_2,$$

таким образом, что каждый примитивный многочлен однозначно задается степенью s_j вместе с числом a_j. Например, если $s_j = 7$ и $a_j = 28 = (011100)_2$ мы получаем многочлен $x^7 + x^5 + x^4 + x^3 + 1$.

Генератор joe-kuo, $D^{(6)}$ – генератор new-joe-kuo-6.21201:

Размерность, при которой каждое значение *t* первый раз появляется																			
		t																	
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
m=10	$D^{(6)}$	2	3	4	5	9	16	32	76	167	402	>21201								
	[1]	2	3	4	5	8	15	21	23	18	36	>1111								
m=12	$D^{(6)}$	2	3	4	6	10	16	34	40	109	233	559	1069	>21201						
	[1]	2	3	4	7	10	10	10	22	35	51	96	61	>1111						
m=14	$D^{(6)}$	2	3	4	6	8	12	22	48	85	164	383	720	1235	1861	>21201				
	[1]	2	3	4	5	9	10	25	17	40	55	67	67	131	61	>1111				
m=16	$D^{(6)}$	2	3	4	6	8	14	15	35	80	159	255	500	837	1553	2375	2721	>21201		
	[1]	2	3	4	6	9	8	15	13	32	58	69	74	102	95	447	167	>1111		
m=18	$D^{(6)}$	2	3	4	7	8	11	15	35	70	108	213	414	720	1177	1819	2616	3092	3677	>21201
	[1]	2	3	4	7	7	10	12	21	28	25	103	126	115	114	196	232	665	380	>1111

Тестовая среда

Основная задача нашей тестовой среды заключается в том, чтобы облегчить работу тестировщикам в таких вещах как:

1) Создание новых типов тестов.
2) Итерирование по тестам и быстрое варьирование параметров.
   Составление xml файлов с понятной для человека структуры тестов:

```
<root>
    <testgroup name="joe-kuo-6(new)">
        <tmsnetgenerator name="joe-kuo" filename="new-joe-kuo-6.21201.txt"/>
        <!--<uniqueness  writeoutput="true" dimension="64" pointnum="65536"/>-->
        <integration writeoutput="true" dimension="32" pointnum="16384">
            <parameters function_key="subcube"/>
        </integration>
        <integration writeoutput="true" dimension="64" pointnum="32768">
            <parameters function_key="subcube"/>
        </integration>
        <projection writeoutput="true" x="50" y="100" pointnum="1024"/>
        <projection writeoutput="true" x="2" y="50" pointnum="4096"/>
        <projection writeoutput="true" x="50" y="8" pointnum="8192"/>
        <orthogonality writeoutput="true" dimension="32" b="2" pointnum="65536"/>
        <orthogonality writeoutput="true" dimension="32" b="32" pointnum="65536"/>
    </testgroup>
    <testgroup name="joe-kuo-5(new)">
        <tmsnetgenerator name="joe-kuo" filename="new-joe-kuo-5.21201.txt"/>
        <!--<uniqueness  writeoutput="true" dimension="64" pointnum="65536"/>-->
        <integration writeoutput="true" dimension="32" pointnum="16384">
            <parameters function_key="subcube"/>
        </integration>
        <integration writeoutput="true" dimension="64" pointnum="32768">
            <parameters function_key="subcube"/>
        </integration>
        <projection writeoutput="true" x="50" y="100" pointnum="1024"/>
        <projection writeoutput="true" x="2" y="50" pointnum="4096"/>
        <projection writeoutput="true" x="50" y="8" pointnum="8192"/>
        <orthogonality writeoutput="true" dimension="32" b="2" pointnum="65536"/>
        <orthogonality writeoutput="true" dimension="32" b="32" pointnum="65536"/>
    </testgroup>
    <testgroup name="joe-kuo-7(new)">
        <tmsnetgenerator name="joe-kuo" filename="new-joe-kuo-7.21201.txt"/>
        <!--<uniqueness  writeoutput="true" dimension="64" pointnum="65536"/>-->
        <integration writeoutput="true" dimension="32" pointnum="16384">
            <parameters function_key="subcube"/>
        </integration>
        <integration writeoutput="true" dimension="64" pointnum="32768">
            <parameters function_key="subcube"/>
        </integration>
        <projection writeoutput="true" x="50" y="100" pointnum="1024"/>
        <projection writeoutput="true" x="2" y="50" pointnum="4096"/>
        <projection writeoutput="true" x="50" y="8" pointnum="8192"/>
        <orthogonality writeoutput="true" dimension="32" b="2" pointnum="65536"/>
        <orthogonality writeoutput="true" dimension="32" b="32" pointnum="65536"/>
    </testgroup>
</root>
```

3) Проверка входных параметров для тестов
4) Подготовка к расширению семейства генераторов

Как работает тестовая среда:

В начале программа берет в качестве входных данных названия файла **xml** конфигураций (если нет параметров, то default.xml), далее происходит регистрация конфигурационных файлов с проверкой входных параметров(класс TestingSuite). В случае несоответствия параметров, тест не регистрируется. После регистрации тесты запускаются последовательно.

Что происходит при регистрации теста:
В первую очередь при регистрации автоматически обрабатывается (парсится) файл конфигурации. Для успешной регистрации тестовой группы необходимо, чтобы:

1) Был корректно обозначен как минимум один генератор
2) Должны быть корректно обозначены тесты, при успешном парсинге теста, создается в куче объект этого теста и указатель на него сохраняется в векторе tests. При этом создается имя теста, которое используется для генерации файлов

Имя теста содержит в себе:
1. Название группы тестов, к которой принадлежит
2а. генератор над которым проходит тест
2б. имя файла направляющих чисел, если требуется для инициализации генератора
3. входные параметры
В данный момент каждый тип теста парсится по-своему, поэтому **для каждого нового типа тестов нужно реализовывать свой парсер**

Все тесты прогоняются по методу абстрактного родительского класса RunTest().

Какие тесты есть сейчас:
1. Покомпонентная уникальность
2. Интегрирование функций
3. Утилита по созданию проекций точек сети на плоскость
4. Попарная ортогональность точек сети

1. Покомпонентная уникальность

Uniqueness(writeoutput, dimension, pointnum) , где
writeoutput – флаг записи результатов в файл,
dimension – размерность пространства, где строится сеть,
pointnum – количество элементов сети.

Описание:
В этом тесте идет проверка на уникальность вхождения значения компоненты в множество.

Реализация:
Создается структура данных set, берем последовательно компоненту (номер оси координат), идет прохождение по каждой точке, а затем значение ее компоненты заносится в set. После этого идет проверка на количество элементов в set и в генераторе, и если хотя бы в одной компоненте есть несоответствие - значит значение из компоненты входило два раза

2. Интегрирование функций

Integration(writeoutput, dimension, pointnum, function) , где
writeoutput – флаг записи результатов в файл,
dimension – размерность пространства, где строится сеть,
pointnum – количество элементов сети,
function – наименование функции, для соответствующего вызовы подтеста.

Описание:
Данный тест(утилита) берет в себя название функции, из чего при прогонке теста по названию функции выбирает нужную реализацию функции и при фиксированном числе точек и переменной размерности пространства считает модуль разности аналитического значения интеграла от функции и численного. В данный момент тест нужен для оценки корректности реализации генератора, условий на остановку у теста нет(утилита).
В планах сделать дополнительный параметр нормировки максимума ошибки, для того, чтобы при прогонке теста валидировать метод, а не просто выписывать значения ошибки.

3. Утилита по созданию проекций точек сети на плоскость

Projection(writeoutput, x, y, pointnum), где
writeoutput – флаг записи результатов в файл,
x – первая ось, лежащая в плоскости,
y – вторая ось, лежащая в плоскости,
pointnum – количество элементов сети.

Описание:
Этот тест нужен для наглядной проверки и сравнения результатов распределения проекций точек сети у разных генераторов. С помощью скрипта в tools можно создать график распределения точек на плоскости.
В планах создать обработку-вычисление на discrepancy в плоскости для того, чтобы при прогонке теста валидировать метод, а не просто выписывать точки проекции.

4. Попарная ортогональность точек сети

Orthogonality(writeoutput, dimension, b, pointnum), где
writeoutput – флаг записи результатов в файл,
dimension – размерность пространства, где строится сеть,
b – основание, по которому точки разделяются на семейство упорядоченных пар,
pointnum – количество элементов сети.

Описание:
В тесте идет проверка на соответсвие того, что заданная сеть обладает свойством ортогональности для любых пар координат, при разделении точек на семейства-пары, где пара – координата квадрата с стороной длиной $\frac{1}{b}$, в котором находится точка.
Этот тест является одним из необходимых тестов на проверку, что сеть является ортогональным массивом «силы» r. Подробнее об ортогональном анализе и ортогональных массивах смотрите в [3] и [4](в будущем надо будет выписать в краткие теоретические сведения).

Ссылки по теории:

[1]. S. Joe and F. Y. Kuo, *Remark on Algorithm 659: Implementing Sobol's quasirandom sequence generator*, ACM Trans. Math. Softw. **29**, 49-57 (2003). Link to paper.

[2]. S. Joe and F. Y. Kuo, *Constructing Sobol sequences with better two-dimensional projections*, SIAM J. Sci. Comput. **30**, 2635-2654 (2008). Link to paper.

[3]. https://en.wikipedia.org/wiki/Orthogonal_array

[4]. https://statistics.stanford.edu/sites/default/files/EFS%20NSF%20464.pdf