**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 в поле

где наши коэффициенты равны либо 0, либо 1.

Мы определяем последовательность положительных целых рекуррентным соотношением:

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

Начальные значения могут быть любыми, но с условием, что , 1 ≤ k ≤ , нечётное и меньше .

Направления определяются по формуле:

Тогда ,-я компонента й точки в последовательности Соболева, задается формулой:

где - это -я цифра справа, если записана в двоичной форме . В дальнейшем мы будем использовать для обозначения двоичного представления номеров.

Например, при и у нас имеется примитивный многочлен . Начиная с , мы используем формулу (2) чтобы получить значения , и т.д. Отсюда мы можем вычислить направления:

,…..

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

|  |  |
| --- | --- |
| 0 = |  |
| 1 = |  |
| 2 = |  |
| 3 = |  |
| 4 = |  |
| 5 = |  |

1. 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 определяется как

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

|  |  |
| --- | --- |
|  |  |
| 0= | =0 |
| 1= | =1 |
| 2= | =3 |
| 3= | =2 |
| 4= | =6 |
| 5= | =7 |
| 6= | =5 |
| 7= | =4 |
| 8= | =12 |
| 9= | =13 |
| 10= | =15 |
| 11= | =14 |
| 12= | =10 |
| 13= | =11 |
| 14= | =9 |
| 15= | =8 |

Исходя из таблицы, можно заметить, что код Грэя - это просто переупорядочение неотрицательных целых чисел в каждом блоке чисел для .

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

где - это -я цифра справа от в коде Грея в двоичном представлении, то есть, = . Аналогично, поскольку и отличаются на одну позицию, мы можем сгенерировать точки рекурсивно, используя формулу

и

где является индексом первой цифры 0 справа в двоичном представлении . Мы имеем , , , , , 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 точек для 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.

1. Примитивные многочлены и числа направленности

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

таким образом, что каждый примитивный многочлен однозначно задается степенью вместе с числом . Например, если и мы получаем многочлен .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |  | 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 |  | 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 |  | 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 |  | 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 |  | 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 |

Генератор joe-kuo, – генератор new-joe-kuo-6.21201:

**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:



1. Проверка входных параметров для тестов
2. 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

1. **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.

1. **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.

1. **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, 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](http://doi.acm.org/10.1145/641876.641879).

[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](http://dx.doi.org/10.1137/070709359).

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

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