Очень часто современные программные продукты разрабатываются в сжатые сроки и при ограниченных бюджетах проектов. Программирование сегодня перешло из разряда искусства, став при этом ремеслом для многих миллионов специалистов. Но, к сожалению, в такой спешке разработчики зачастую игнорирует необходимость обеспечения информационной безопасности и защищённости своих продуктов, подвергая тем самым пользователей своих продуктов неоправданному риску.

Поэтому на сегодняшний день остро встает проблема исчерпывающего тестирования. Архитектурные особенности современных программных продуктов, многообразие физических дефектов и ошибок требует наличия эффективных и производительных методов и алгоритмов тестирования. Одним из таких методов является метод квазислучайного тестирования, использующие в своей основе квазислучайные тестовые последовательности.

**Понятия в тестировании.**

Большинство терминов и понятий в области тестирования и диагностики вычислительных систем имеют достаточно общий смысл и характеризуются отсутствием точности и однозначности. В основном это связано с разнообразием типов современных вычислительных систем, большим количеством различных методов их тестирования, различными уровнями абстракции их описания, а также формулировкой целей и задач тестирования. Поэтому для последующего обсуждения очень важно однозначно определить следующие термины: физический дефект (*physical defect*), ошибка проектирования (*mistake*), неисправность (*fault*), ошибка системы (*error*) и неисправное поведение системы (*failure*).

Обычно, на самом низком уровне абстракции используется термин «*неисправность*» как математическая модель, описывающая физические дефекты системы и ошибки при ее проектировании, изготовлении и использовании. Основными источниками неисправностей являются: ошибки спецификации(*specification mistakes*) из-за неправильных или неправильно интерпретированных проектных требований к системе и/или ее спецификации; ошибки проектирования (*implementation mistakes*) в основном из-за ошибок кодирования (*bags*); физические дефекты аппаратных компонентов системы (*component physical defects*); внешние факторы (*external factors*), такие как условия окружающей среды (температура, электромагнитные и радиационные излучения и др.) или человеческий фактор (ошибки оператора). Тогда неисправность как математическую модель неисправного состояния вычислительной системы определим следующим образом.

Неисправностью вычислительной системы является такая ее функциональность, которая может приводить к новым выходным значениям (наблюдаемому поведению) и/или к новому состоянию системы, не соответствующим спецификации системы и требованиям, предъявляемым к ней.

То есть неисправности вычислительной системы могут приводить к ее неверным наблюдаемым выходным значениям и ошибочным внутренним состояниям или только к ошибочным внутренним состояниям. Отметим, что не для всего множества входных значений наличие неисправности приводит к ошибочным состояниям системы и ошибочным выходным значениям.

Ошибкой вычислительной системы из-за наличия в ней неисправности является формирование для некоторого множества входных значений системы одного или более неверных результатов, отличающихся от ожидаемых значений.

Ошибки вычислительной системы, по сути, являются результатом активизации ее неисправностей. Они могут привести к неисправному поведению системы (системному сбою), которое, в свою очередь, может быть наблюдаемым признаком неисправного поведения (состояния) вычислительной системы.

Неисправным поведением вычислительной системы (сбоем вычислительной системы) называется формирование наблюдаемых выходных значений системы, отличных от ожидаемых, для некоторого множества входных значений.

Неисправное поведение вычислительной системы возникает как результат решения задачи транспортировки ошибочного значения (ошибки) системы на ее наблюдаемые выходы.

Приведенные определения в равной степени применимы как к программному, так и аппаратному обеспечению вычислительных систем. Обоснованность и применимость этих определений проиллюстрируем примером из программного обеспечения . Рассмотрим программу на языку Python, которая вычисляет остаток от деления на три квадрата (*data*\*\*2) входного значения *data* (рис. 1).

*begin*

*read* (*data*);

*data*: = 2\**data*; (← ***неисправность***)

*data*: = *data* mod 3;

*write* (*data*)

*end*.

Рис. 1. Пример программы, содержащей неисправность

В связи с наличием неисправности, из-за ошибки при кодировании, в третьей строке кода программы эта программа фактически вычисляет остаток от деления на три удвоенного значения *data*. При рассмотрении программы, представленной выше для исходного значения *data* = 3 выходной результат оператора *data*: = 2\**data*, содержащего неисправность, составляет 6, тогда как ожидаемое значение, при отсутствии неисправности, должно быть 9. Таким образом, возникает вычислительная ошибка. В то же время для *data* = 2 ошибочного значения не возникает, так как неисправность не активизируется. Кроме того, для обоих значений 3 и 2 входных данных *data* программа вычисляет ожидаемо правильные результаты, а именно 0 и 1. Это означает, что значения *data*, равные 3 и 2, не вызывают неисправного состояния (сбоя программы), так как значение *data* = 2 даже не вызывает ошибку, а значение *data* = 3 инициирует ошибочный промежуточный результат, равный 6, вместо 9. Однако в обоих случаях формируется правильное выходное значение программы, равное 0. В то же время для *data* = 4 программа, содержащая неисправность в третьем операторе, формирует ошибочное значение (вычислительную ошибку). Действительно, в этом случае выходное значение равняется 2 вместо ожидаемой величины 1, что свидетельствует о неисправном состоянии программы и приводит к ее сбою.

В следующем примере рассмотрим аппаратную составляющую вычислительной системы, представленную оперативным запоминающим устройством (ОЗУ) [7]. Предположим, что ОЗУ содержит четырехразрядные запоминающие ячейки, которые используются для хранения данных, состоящих из четырех бит и представляющих собой шестнадцатеричную цифру. В одной из ячеек ОЗУ возникла константная неисправность ≡0, которая приводит к тому, что, например, во втором разряде ячейки постоянно находится значение 0 [7]. Что касается остальных разрядов ячейки, то их содержимое может быть 0 или 1. Рассмотрим случай шестнадцатеричных данных равных 4, 3 и 2, которые должны быть записаны в ОЗУ и затем считаны. Для значения данных 4 наличие неисправности во втором бите ячейки ОЗУ не приводит к ошибке, несмотря на наличие неисправности ≡0, так как 4(16) = 0100(2). Оба значения данных 3 и 2 в двоичном представлении (0011, 0010) содержат 1 во втором бите, что является причиной возникновения ошибки выборки после выполнения операции чтения. Это, в свою очередь, может привести к неисправному состоянию вычислительной системы в целом.

Спецификация

системы

mistakes

Ошибки

реализации

Внешние

факторы

Физические

дефекты

дефекты

Неисправности

программного

обеспечения

Software

faults

Неисправности аппаратной

части системы

Неисправное поведение вычислительной

системы

Ошибки

вычислительной

системы

Рис. 2. Причинно-следственная связь между неисправностями, ошибками и неисправным поведением вычислительной системы

Из приведенного рис. 2, видно, что первопричиной неисправностей являются: ошибки спецификации, ошибки реализации, физические дефекты аппаратных компонент системы и внешние факторы [4–6]. Наличие неисправностей может вызывать ошибки как программных, так и аппаратных средств вычислительных систем. Ошибки возникают в результате активизации неисправностей вычислительной системы, как это было показано в приведенных выше примерах. И наконец, активизированная ошибка вычислительной системы может привести к наблюдаемому неисправному поведению вычислительной системы.

Существующие методологии построения тестовых последовательностей решают задачу нахождения такого подмножества входных тестовых воздействий (*failure patterns*), которые приводили бы к наблюдаемому неисправному поведению вычислительной системы в случае возникновения в ней неисправностей на всех стадиях жизненного цикла системы [4–6].

**Обобщенные входные тестовые воздействия**

Для того чтобы оценить эффективность различных методов тестирования вычислительных систем, рассмотрим множества их входных тестовых воздействий, которые приводят к наблюдаемому неисправному поведению систем. Очевидно, что данные множества являются уникальными для каждого из объектов тестирования и зависят от многих факторов, а также от его архитектуры и функциональности. Однако структура входных тестовых воздействий характеризуется рядом закономерностей, отмеченных в последних публикациях.

Большой объем эмпирических и экспериментальных исследований различного рода программного обеспечения показал обобщенность (структурированность) входных воздействий (наборов), которые инициируют неисправное поведение систем, т. у. являются тестовыми воздействиями . В основополагающей работе [10] была представлена классификация обобщенных входных тестовых воздействий на три категории: точечные тестовые наборы (*point patterns*), узкополосные тестовые наборы (*strip patterns*) и блочные тестовые наборы (*block patterns*) [10]. Для иллюстрации данной классификации пространство входных наборов рассматривается как двухмерное. Тогда три вида входных тестовых наборов имеют простую геометрическую интерпретацию (рис. 3) [8–11].

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

а) б) в)

Рис. 3. Типовые входные тестовые воздействия

На рис. 3 области, выделенные черным цветом, соответствуют входным воздействиям, приводящим к наблюдаемому неисправному поведению программного обеспечения, а фигуры квадратов соответствуют областям входных воздействий [8, 10, 11]. Соответственно, точечные тестовые наборы (рис. 3, а) соответствуют уникальным входным наборам либо небольшим их количествам, распределенным по всему пространству входных наборов, причем их распределение в большинстве случаев имеет регулярную структуру. Узкополосные тестовые наборы (рис. 3, б) характеризуются непрерывными множествами входных воздействий имеющих вид узких полос. Тестовые наборы блочного типа (рис. 3, в) представляют собой одну либо несколько непрерывных областей входных воздействий, приводящих к наблюдаемому неисправному поведению систем.

Как правило, эти типовые входные воздействия выбираются в качестве моделей входных тестовых воздействий, поскольку многочисленные эмпирические исследования подтверждают целесообразность их использования в качестве приближения к реальным входным тестовым воздействиям [8–11]. Несмотря на то что эти модели не являются реальными, их применение, а также применение их комбинаций оказывается весьма эффективным при построении тестовых последовательностей для реальных вычислительных систем [8–11].

Значительная часть аппаратной составляющей вычислительных систем, особенно систем на кристалле и встроенных систем, включает в себя встроенную память которая, в соответствии с прогнозами будет занимать доминирующую часть на кристаллах вычислительных систем. Поэтому эффективное тестирование памяти и совершенная методология анализа неисправностей запоминающих устройств будет способствовать повышению надежности и выхода годных встроенных систем и систем на кристалле, особенно при ускоренных процессах их разработки и применении передовых технологий изготовления [1, 7].

Причинно-следственный подход может быть использован для прогнозирования неисправного поведения ОЗУ, в случае возникновения физических дефектов. Неисправное поведение памяти представляется растровым изображением физического топологического представления результатов тестирования, показывающим расположение неисправных запоминающих ячеек. Чаще всего возникают одиночные неисправности запоминающих ячеек памяти; неисправности групп запоминающих ячеек; неисправности дешифратора адреса ячейки памяти; неисправности шин данных и другие неисправности запоминающих устройств [7, 12–14].

Согласно устоявшейся классификации неисправностей запоминающих устройств различают два их множества, а именно простые, в которых участвуют одна либо две ячейки памяти, и сложные, включающие в себя множество ячеек [7, 14]. К неисправностям, затрагивающим одну ячейку, относятся константные неисправности и переходные неисправности, а к неисправностям, в которых участвуют две ячейки, – неисправности взаимного влияния [7, 14]. В обоих случаях обобщающей моделью входных тестовых воздействий (адресов ячеек памяти) является точечная модель (рис. 4, а) либо, для случая многократных неисправностей данных типов, блоковая (рис. 4, б) [7, 13, 14]. Кодочувствительные неисправности, неисправности дешифратора адреса и неисправности типа «*разрушающая операция чтения*» и типа «*ошибочная запись*» покрываются моделью узкополосных входных воздействий (рис. 4, в) [7, 13, 14]. В случае комбинации физических дефектов памяти, а также их многократных разновидностей входные тестовые воздействия описываются комбинированной моделью (рис. 4, г), объединяющей блоковый и узкополосный тип [13].

Таким образом, все множество моделей неисправного поведения запоминающих устройств описывается четырьмя обобщенными моделями входных тестовых воздействий, представленных на рис. 4 [13].

.

а) б) в) г)

Рис. 4. Типовые входные тестовые воздействия запоминающих устройств

В сравнении с входными тестовыми наборами для программного обеспечения тестовые воздействия запоминающих устройств имеют реальную двухмерную структуру в силу физического двухмерного представления матрицы запоминающих ячеек [14]. В данном случае входными тестовыми данными являются адреса запоминающих ячеек, состоящие из адресов по горизонтальной и вертикальной оси матрицы ячеек памяти.

Как видно из приведенного анализа, типовые входные тестовые воздействия для программного обеспечения и для большей части аппаратных средств вычислительных систем описываются аналогичными моделями, что, очевидно, позволяет использовать единый подход для тестирования вычислительных систем в целом. Следующим важным выводом анализа типовых входных тестовых воздействий является их разнородность по структуре (см. рис. 3 и 4), что затрудняет выбор наиболее эффективного метода их тестирования.

**Анализ методов тестирования вычислительных систем**

В общем случае тестирование вычислительных систем направлено на повышение их надежности и заключается в выявлении максимально возможного количества неисправностей систем за приемлемый (реальный) промежуток времени. Исчерпывающее тестирование (*exhaustive testing*) характеризуется максимально возможной эффективностью, так как при его реализации проверяется (тестируется) объект тестирования на всевозможных входных воздействиях для всех возможных состояниях объекта тестирования [15]. В случае двоичных входных воздействий из *N* бит необходимо сгенерировать 2*N* двоичных комбинаций. Всевозможные двоичные комбинации генерируются для каждого состояния, которое в случае цифровых устройств и систем определяется количеством *M* запоминающих элементов, которые используются для построения регистров, счетчиков и других последовательностных устройств систем. Таким образом, количество состояний цифрового устройства или системы определяется величиной 2*M* в силу того, что каждый запоминающий элемент может находиться в одном из двух состояний, а именно в состоянии 0 или состоянии 1. Сложность исчерпывающего теста (количество тестовых наборов) определяется астрономической величиной 2*N*+*M*. Экспоненциальный рост длины исчерпывающего теста существенно ограничивает области его применения только для случаев простейших вычислительных устройств с небольшими значениями *N* и *M* [15, 16]. Локально исчерпывающее (*locally exhaustive*) [16] или псевдоисчерпывающее (*pseudo exhaustive*) [17, 18] тестирование вычислительных систем позволяет избежать ограничений на число входов тестируемого устройства или системы. Эти подходы являются реальной альтернативой для исчерпывающего тестирования и позволяют существенно сократить число тестовых векторов, однако требуют большого объема исследований при построении подобных тестов [16, 18], а также детального описания вычислительных систем, что не всегда возможно и достижимо.

Для эффективной аппроксимации исчерпывающего и псевдоисчерпывающего тестирования широко используется случайное (вероятностное) тестирование (*random testing*) [19]. В данном случае под аппроксимацией понимается формирование заведомо не исчерпывающих и не псевдоисчерпывающих тестов, а тестов, которые являются некоторым их приближением. Данный метод тестирования широко применяется в рамках модели черного ящика (*black box*), когда вычислительная система описывается только на уровне выполняемых ею функций, а ее внутренняя структура и конкретная реализация как программной, так и аппаратной части не учитываются. Согласно определению случайного тестирования очередное значение тестового набора выбирается случайным образом, независимо от значений предыдущих наборов теста [19, 20]. Случайное тестирование является неэффективным, когда плотность оставшихся (необнаруживаемых) неисправностей оказывается низкой [20]. Случайное тестирование не использует информацию, которая доступна при реализации метода черного ящика, а именно информацию о предыдущих тестовых наборах, что увеличивает длину тестовой последовательности и уменьшает полноту покрытия неисправностей вычислительных систем [11, 18, 21]. Использование информации о предыдущих тестовых наборах явилось основой создания так называемого антислучайного тестирования (*antirandom testing*) [18, 22, 23].

Предпосылкой антислучайного тестирования является то, что для достижения более высокой полноты покрытия неисправностей вычислительных систем очередной тестовый набор выбирается из случайных тестовых наборов таким образом, чтобы он был максимально отличным от ранее сгенерированных наборов. Для этого используются различные метрики отличия, такие как расстояние Хемминга и декартово расстояние [22, 23]. При антислучайном тестировании очередной тестовый набор выбирается из множества наборов с максимальным расстоянием от ранее сгенерированных наборов, что позволяет достичь большей эффективности по сравнению со случайным тестированием [22, 23]. К сожалению, основным недостатком антислучайного тестирования является его вычислительная сложность, вызванная необходимостью вычисления метрик расстояния для каждого кандидата в тесты [22]. Даже для улучшенных вариантов антислучайного тестирования вычислительная сложность их реализации является существенным ограничением при практическом применении для реальных размерностей тестовых наборов [23]. В качестве более эффективной метрики для построения антислучайного теста используется максимальное минимальное расстояние Хэмминга (*maximal minimal hamming distance*), которое применяется для построения антислучайного теста с ограниченным числом наборов [18, 25, 26]. К сожалению, проблема вычисления расстояния является ключевой проблемой при генерировании антислучайных тестов, что ограничивает области их применения [11, 18, 22–26].

Все перечисленные методы тестирования в своей основе используют модель черного ящика и направлены на формирование входных тестовых наборов как подмножества входных воздействий формируемых в большинстве случаев с использованием случайных воздействий [15–26]. Случайное тестирование и все многообразие его модификаций можно описать в терминах численных *методов Монте-Карло*, основанных на получении большого числа реализаций стохастического (случайного) процесса. Основными недостатками методов случайного тестирования является их невысокая полнота покрытия неисправностей и большая длина тестовых последовательностей. Подобными недостатками характеризуется и метод Монте-Карло, для которых характерны значительная вычислительная погрешность и заметная временная сложность. Поэтому в качестве альтернативного решения для тестирования вычислительных систем в данной статье предлагается использование идеи квазислучайного тестирования, основанного на применении квазислучайных последовательностей в качестве тестовых последовательностей, эффективно покрывающих пространство входных воздействий систем [27–31]. Аналогично, как и в методе *квази*-Монте-Карло (КМК), квазислучайное тестирование, очевидно, позволит достичь большей полноты покрытия при меньших длинах тестовых последовательностей. Основой реализации квазислучайного тестирования являются квазислучайные последовательности. В следующем разделе статьи проводится анализ подобных последовательностей, обосновывается использование последовательностей Соболя и предлагаются авторские модификации для генерирования большого числа их реализаций.

**Равномерно распределенные последовательности точек.**

Прежде чем перейти к равномерно распределенным последовательностям точек разберемся с понятием равномерности. Если задаться вопросом, как равномерно разместить на отрезке 4 точки, то ответ не вызовет больших затруднений. Более того ответов может быть несколько, например как показано на рис 1.

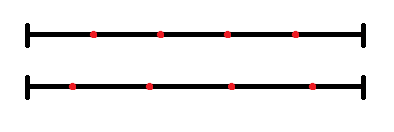


Рис. 1

Даже в одномерном случае не вполне понятно, что значит «равномерно». Вопрос еще более усложниться, если точки придется размещать в кубе или n-мерном пространстве, если количество точек , если надо увеличивать не нарушая равномерности.

Такие вопросы изучает теория равномерно распределенных последовательностей. Она исследует бесконечные последовательности точек обладающие тем свойством, что группа точек при каждом в каком-то смысле равномерно расположена в кубе. При увеличении плотность заполнения увеличивается, а равномерность сохраняется.

Обозначим буквой произвольный отрезок принадлежащий отрезку [0, 1], например , где . Для определенности будем в дальнейшем считать все отрезки замкнутыми слева и открытыми справа, за исключением случая, когда правый конец : в этом случае будем считат замкнутым также и справа. Длину обозначим через .

Двоичными отрезками назовем отрезки, которые могут быть получены путем деления отрезка [0, 1] на равных частей (). Для таких отрезков ниже будет использоваться обозначение , где очевидно . Конечно, в случае в соответствии с соглашением выше, считается замкнутым с обоих концов. Очевидно, что

Левую и правую половины обозначим через и ; это также двоичные отрезки, причем .

Наряду с двойной нумерацией будем также использовать простую нумерацию, полагая, что , где .

Дадим понятие равномерно распределенной последовательности. Рассмотрим произвольную последовательность точек принадлежащих отрезку [0, 1]. Пусть какой-нибудь отрезок . Выделим начальный участок последовательности и через обозначим количество точек этого участка, принадлежащих l.

Последовательность называется равномерно распределенной в отрезке[0, 1], если для любого l справедливо

. (1)

Геометрический смысл этого определения достаточно ясен: если равномерно распределена то при больших N количество точек , то есть пропорционально длине .

Свойство равномерной распределенности асимптотическое. Можно в заменить, добавить, исключить любое количество любых точек, и равномерность ее распределения от этого не измениться. В самом деле, если при все точки остались прежними, то при всех достаточно больших N значение может отличаться от старого значения не больше чем на . Назначении предела (1) это никак не скажется, ибо .

Следует отметить, что для того чтобы была равномерно распределен, необходимо и достаточно, чтобы соотношение (1) выполнялось для всех двоичных отрезков.

Отметим также, что в определении (1) можно заменить полуоткрытые отрезки открытыми или замкнутыми; можно ограничиться отрезками вида .

Связь определения (1) с задачами вычислительной математики в какой-то мере видна из теоремы Вейля, которая гласит, что для того чтобы была равномено распределенной, необходимо и достаточно, чтобы для любой интегрируемой по Риману функции выполнялось соотношение

(2)

Интеграл Римана – это обычный определенный интеграл, для существования которого необходимо, чтобы , была ограничена.

Чтобы показать связь между соотношениями (1) и (2) выберем произвольный отрезок рассмотрим функцию , называемую обычно индикатором этого отрезка: , если ; , если .

Так как , , то при соотношения (2) и (1) совпадают.

В формулировке теоремы Вейля выделим прямое и обратное утверждения: если равномерно распределена, то для любой интегрируемой, по Риману, функции выполняется (2). Обратное утверждение: если (2) справедливо для любой интегрируемой, по Риману, функции , то равномерно распределена.

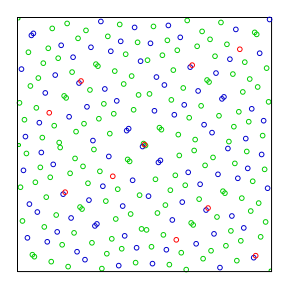
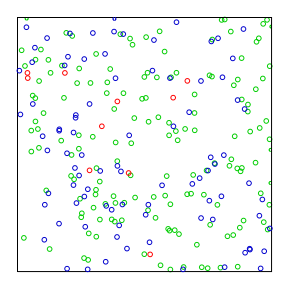
**Квазислучайные последовательности**

Квазислучайной последовательностью чисел, называется последовательность равномерно распределенных чисел, обладающая свойствами случайной последовательности. Квазислучайные последовательности получили широкое применение в приложениях математики и теории чисел. В частности, в задачах численных методов квазислучайные последовательности применяются в реализации алгоритма Моне-Карло, для вычисления несобственных интегралов. Такие алгоритмы получили название квази-Монте-Карло, их применение позволяет достичь меньших вычислительных погрешностей и более быстрой сходимости. В русскоязычной литературе такие последовательности получили название -последовательностей.

Дадим определение -последовательностей, через определение . называют конечную группу точек, если расстояние от произвольной точки до ближайшей из точек группы не превосходит . Таким образом, сетку, состоящую из точек, где v – целое, назовем , если каждому двоичному отрезку с длиной 1/N принадлежит одна точка сетки. Двоичным участком последовательности называется множество членов с номером i, удовлетворяющими неравенству вида , где , Тогда последовательность назовем -последовательностью, если любой ее двоичный участок представляет собой .

Название , согласно Соболю, получилось в результате сокращения фразы «Любой двоичный участок есть ».

Для визуальной демонстрации большей равномерности квазислучайной последовательности точек по сравнению с псевдослучайной последовательностью обычно рассмотрим двухмерное пространство в виде квадрата. На рисунке ниже приведен пример заполнения пространства квадрата 256 точками псевдослучайной последовательности (слева) и квазислучайной последовательности(справа), в качестве 3–его измерения, используется цвет точки. Из рисунка видно, что точки квазислучайной последовательности заполняют пространство более равномерно.



***Последовательности Корпута***

Одним из простейших примеров квазислучайных последовательностей является последовательности Корпута. Алгоритм генерирования чисел Корпута представлен ниже.

Для целого числа значение элемента последовательности Корпута может быть получено для любого базиса *p* представления числа *n*, где *p* – простое целое число. Первоначально целое число *n* представляется в базисе *p* как

где α*i*(*n*)∈{0, 1, 2, …, *p*−1} является значением *i*-й цифры *p*-ичного представления числа *n*, а *I* – наименьшим целым значением *i*, для которого α*i*(*n*)≠0, а для всех *i*>*I* выполняется равенство α*i*(*n*)=0. Значение *I* вычисляется как

.

Для получения значения *xn* цифрыα*i*(*n*) числа *n* транспонируются относительно запятой, разделяющей целую и дробную части *p*-ичного числа, таким образом, что первой после запятой оказывается младшая α0(*n*) цифра *p*-ичного представления числа *n*. Следующей цифрой будет α1(*n*) и т. д.

Процедуру транспонирования, в результате которой получается значение *xn*, можно описать следующей формулой:

.

В качестве примера рассмотрим целое число *n* = 513, которое представим в базисе *р* = 5 как 513(10) = 4x53 + 0x52  +2x51+ 3x50 = 4023(5) с α0(513) = 3, α1(513) = 2, α2(513) = 0 и α3(513) = 4. Следует отметить, что *I* = ⎣log5513⎦ = 3 и соответственно α*i*(*n*)=0 для *i*>3. Выполнив процедуру транспонирования, получим значение элемента *x*513 последовательности Корпута *x*513 = 3/51 + 2/52 + 0/53 + 4/54 = 304/625, которое принадлежит интервалу [0, 1]. Квазислучайной последовательностью Корпута с основанием *p*=2 для *N* = 7 является последовательность *xn*, состоящая из следующих элементов: *x*1 = 1/2(10)=0,1(2); *x*2 = 1/4(10)=0,01(2); *x*3 = 3/4(10)=0,11(2); *x*4 = 1/8(10)=0,001(2); *x*5 = 5/8(10)=0,101(2); *x*6 = 3/8(10)=0,011(2) и *x*4 = 7/8(10)=0,111(2).

Важным свойством последовательности Корпута является то, что после генерирования каждого множества из 2*k*–1 элементов для приведенного примера *k* = 1, 2 и 3 последовательность максимально равномерно распределена, т. е. самый длинный интервал, который не содержит элементов из последовательности Корпута, является минимальным.

К преимуществам последовательности Корпута можно отнести относительно невысокую алгоритмическую сложность процедуры генерирования чисел.

К недостаткам чисел последовательности Корпута можно отнести их иррациональность, что предопределяется произвольным базисом p, что затрудняет использование данное множество чисел в современных приложениях информатики. Чаще всего используются последовательность Корпута, где p = 2.

***Последовательности Халтона***

В отличие от последовательности корпута, последовательность Халтона представляет собой многомерную квазислучайную последовательность. Данная последовательность, часто используется для построения других многомерных квазислучайных последовательностей. Для построения чисел Халтона используется последовательность Корпута. Координаты точек Халтона задаются последовательностями корпута с использованием простых чисел p как основания системы счисления. Например, первая координа задается из последовательности Корпута с основанием системы счисления p = 2, вторая – p = 3, третья – p =5 т.д.

В качестве примера рассмотрим последовательность точек Халтона в двухмерном пространстве (*s*=2) для *p*=2 и *p*=3. Для построения данной последовательности необходимо использовать две последовательности Корпута для *p* = 2 и 3. В десятичной системе счисления эти последовательности иррациональных чисел принимают следующий вид: 1/2, 1/4, 3/4, 1/8, 5/8, 3/8, 7/8, ..., и 1/3, 2/3, 1/9, 4/9, 7/9, 2/9, 5/9, ... . Тогда последовательность Халтона будет представляться последовательностью точек (1/2, 1/3), (1/4, 2/3), (3/4, 1/9), (1/8, 4/9), (5/8, 7/9), (3/8, 2/9), (7/8, 5/9), ... двухмерного пространства. Пример последовательности точек Халтона *xn* для трехмерного случая приведен в табл. 1.

|  |  |  |  |
| --- | --- | --- | --- |
| *xn* | *p*=2 | *p*=3 | *p*=5 |
| 1 | 1/2 | 1/3 | 1/5 |
| 2 | 1/4 | 2/3 | 2/5 |
| 3 | 3/4 | 1/9 | 3/5 |
| 4 | 1/8 | 4/9 | 4/5 |
| 5 | 5/8 | 7/9 | 1/25 |
| 6 | 3/8 | 2/9 | 6/25 |
| 7 | 7/8 | 5/9 | 11/25 |
| 8 | 1/16 | 8/9 | 16/25 |

Следует отметить, что последовательности Халтона характеризуются низким качеством для размерностей *s* больше чем 14. На практике в силу корреляционных зависимостей последовательности Халтона используются для значений *s* не большее чем 6.

Недостатки чисел последовательности Халтона, как и у последовательности Корпута заключаются сложности применения их на практике, поэтому как правило используется только одномерная последовательность, с основание счисления p =2. Также стоит отнести к недостаткам высокую алгоритмическую сложность процедуры генерирования последовательности для произвольного числа измерений, так как для этого необходимы алгоритмы генерирования простых чисел, которые в настоящее время имеют низкое быстродействие.

**Последовательности Соболя**

Последовательности Соболя представляют собой многомерные квазислучайные последовательности. Особенностями последовательностей соболя является тот факт, что для построения чисел последовательности они используют двоичную систему счисления и таким образом находят широкое применение современных приложений. Последовательность точек Соболя в *s*-мерном пространстве строится на основании одномерной последовательности Корпута, когда первая координата точек формируется последовательностью Корпута для *p*=2, а остальные – путем процедуры перестановок. Перестановки зависят от так называемых направляющих чисел**, применяемых для всех измерений ** *s*-мерного пространства. На направляющие числа налагается ряд ограничений, аналитически их представить как **=, где **представляет собой целое нечетное число, удовлетворяющее неравенству 0<**<2*i*. Таким образом направляющие числа образуют последовательность дробных двоичных чисел с *w* двоичными битами после запятой. Для описания конкретной последовательности Соболя необходимо задать направляющие числа ** для всех измерений *s*-мерного пространства.

Значение *n*-го элемента (точки) *xn* последовательности Соболя определяется его координатами ** для всех измерений *s*, которые вычисляются согласно выражению

.

Где α*i*(*n*)∈{0, 1}, **, являются значениями цифр двоичного представления числа *n*, а символ ⊕ означает операцию поразрядного сложения по модулю два (XOR). Например, если ** = 1, ** = 3, ** = 5 (*w*=3) и соответственно ** =0,100, ** = 0,110 и **= 0,101, можно получить значение произвольного элемента последовательности Соболя для *n*∈{1, 2, … , 2*w*–1}={1, 2, … , 7}. Предположим, что *n*=7(10), которое в двоичном представлении записывается в виде 111(2), тогда значение *j*-й координаты ** седьмого элемента *x*7 последовательности Соболя вычисляется как

.

Все элементы одномерной последовательности Соболя длиной 2*w*–1=23–1=7 приведены в ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| *n*(10) | *n*(2)= α2(*n*)α1(*n*)α0(*n*) | *xn* | 2*wxn* |
| 1 | 0 0 1 | v1 | 1 0 0 |
| 2 | 0 1 0 | v2 | 1 1 0 |
| 3 | 0 1 1 | v1⊕v2 | 0 1 0 |
| 4 | 1 0 0 | v3 | 1 1 1 |
| 5 | 1 0 1 | v1⊕v3 | 0 1 1 |
| 6 | 1 1 0 | v2⊕v3 | 0 0 1 |
| 7 | 1 1 1 | v1⊕v2⊕v3 | 1 0 1 |

К недостаткам классической последовательности Соболя можно отнести высокую алгоритмическую сложность вычисления элементов последовательность сложность, зависящая от значения номера элемента (количества операций сложения по модулю два), а также ограниченное множество последовательностей, определяемое выбором направляющих чисел.

**5. Модифицированные последовательности Соболя**



В классической последовательности Соболя значение координаты *n*-го элемента *xn* вычисляется как поразрядная сумма по модулю два максимум *w* операндов, точное значение операндов зависимости от количества ненулевых компонентов двоичного представленияα*w*-1(*n*)α*w*-2(*n*)…α1(*n*)α0(*n*) величины *n*.

Существует способ сокращения операндов до одного. В основу этого способа положено применение двоичного кода грея. Известно, что двоичное число *n*+1, закодированное в коде Грея, отличается от числа *n* только в одном бите. Представление числа *n* в коде Грея может быть получено согласно известному соотношению *ng*=*gw*-1(*n*)*gw*-2(*n*)…*g*1(*n*)*g*0(*n*)=α*w*-1(*n*)α*w*-2(*n*)…α1(*n*)α0(*n*)⊕0α*w*-1(*n*)α*w*-2(*n*) … α1(*n*)*.* Здесь индекс *g* числа *ng* означает его представление в коде Грея.

Исходя из того что (*n*+1)*g* отличается от *ng* только в одном бите, значение **будет отличаться от величины **только значением одного направляющего числа **. Тогда значение *j*-й координаты **элемента *x*(*n*+1)*g* последовательности Соболя будет определяться как

**. (1)

Данный способ получил название экономисного способа вычисления ЛП0-последовательностей Антонова-Салеева, по фамилиям изобретателей.

Алгоритм формирования последовательности Соболя для одномерного случая в соответствии с (1) представлена в таблице ниже. Здесь рассмотрен случай последовательности сгенерированной для , в качестве направляющих чисел используются ** =0,100, ** = 0,110 и **= 0,111

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *ng* | vi | 2*wxng* |
| 0 0 1 | 001⊕000=001 | v1 | 100 |
| 0 1 0 | 010⊕001=011 | v2 | 010 |
| 0 1 1 | 011⊕001=010 | v1 | 110 |
| 1 0 0 | 100⊕010=110 | v3 | 001 |
| 1 0 1 | 101⊕010=111 | v1 | 101 |
| 1 1 0 | 110⊕011=101 | v2 | 011 |
| 1 1 1 | 111⊕011=100 | v1 | 111 |

Значение индекса направляющего числа **в выражении (1) зависит от так называемой последовательности переключений *Tq* отраженного кода Грея Для классического отраженного кода Грея переключательная последовательность задается рекурсивной процедурой следующим образом. Первоначально задается *T*1 =1 и, если *q*>1, *Tq* = *Tq*-1, *q*, *Tq*-1 и обозначает номер бита изменяемого при генерировании следующего числа кода грея на основании предыдущего. Например, для *q*=5 получим *T*5 = 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1

Применение соотношения (1) позволяет существенно снизить вычислительную сложность генерирования последовательностей Соболя, что и предопределило их широкое применение на практике.

Другим существенным недостатком классических последовательностей Соболя является ограниченность их количества, которое определяется наборами направляющих чисел. С целью расширения множества последовательностей Соболя введем формализацию представления модифицированных направляющих чисел в виде нижней треугольной матрицы (унитреугольной матрицы) с единичной главной диагональю.

В силу налагаемых на направляющие числа ограничений, описанных выше, напрявляющие числа для всех измерений v*i*=0,β-1(*i*)β-2(*i*)…β-*w*(*i*), во всех случаях имеют определенные значения β-*i*(*i*) =1 и β-*j*(*i*)=0 для *j*>*i*, так же, как и произвольные значения β-*j*(*i*)∈{0,1} для *j*<*i*. Это означает, что для всех возможных последовательностей Соболя и всех их координат v1 = 0,100…0, а соответственно 2*w*v1=100…00 в силу того, что для *m*1 существует только одно безальтернативное значение *m*1=1<21. Так как *m*2 есть нечетное целое, меньшее чем 22, оно может принимать два значения: 1 или 3.Соответственно 2*w*v2=β-1(2)10…00, где β-1(2)=0 для *m*2=1 и β-1(2)=1 для *m*2=3. Для *m*3 имеем2*w*v3=β-1(3)β-2(3)10…00 и т. д.

Рассмотрим случай одномерных последовательностей Соболя и обозначим значения 2*w*v*i* новой переменной μ*i*, которую будем рассматривать как значения модифицированных направляющих чисел. Результаты, полученные для одномерного случая, могут быть экстраполированы на многомерные последовательности Соболя.

Числа μ*i* можно представить в виде нижней треугольной матрицы (унитреугольной матрицы) с единичной главной диагональю.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| μ*i* | β-1(*i*) | β-2(*i*) | β-3(*i*) | … | β*-w+*1(*i*) | β*-w*(*i*) |
| μ1 | 1 | 0 | 0 | … | 0 | 0 |
| μ2 | β-1(2) | 1 | 0 | … | 0 | 0 |
| μ3 | β-1(3) | β-2(3) | 1 | … | 0 | 0 |
| … | … | … | … | … | … | … |
| μ*w–*1 | β-1(*w*–1) | β-2(*w*–1) | β-3(*w*–1) | … | 1 | 0 |
| μ*w* | β-1(*w*) | β-2(*w*) | β-3(*w*) | … | β*-w*+1(*w*) | 1 |

Согласно экономичному алгоритму Антонова-Салеева генерирования *x*(*n+*1)*g* = *xng*⊕μ*i* последовательности Соболя (1) для получения очередного значения используется только одно направляющее число. Индекс *i* направляющего числа μ*i* определяется последовательностью переключений отраженного кода Грея, в которой на каждой второй позиции использует индекс 1, на каждой четвертой индекс 2, на каждой восьмой 3 и т. д. Это следует из определения переключательной последовательности. Таким образом, каждое второе значение последовательности Соболя получается с использованием μ1, каждое четвертое с использованием μ2 и т. д. Для μ1 значение β-1(1)=1, что обеспечивает максимальную частоту изменения старшего бита кода элемента *xng* последовательности Соболя. Максимальная частота изменения следующего бита кода *xng* в два раза меньше и т. д.

Одним из ключевых элементов при генерировании последовательности Соболя являются направляющие числа μ*i*, , которые должны принимать уникальные значения для каждой координаты точек в *s*-мерном пространстве. Наиболее значимым свойством направляющих чисел, вытекающим из ограничений на эти числа и подтверждающимся видом введенной авторами матрицы модифицированных порождающих чисел, является их линейная независимость.

Ниже представлено несколько способов генерирования порождающей матрицы.

Для получения полного множества модифицированных направляющих чисел μ*i*,  на основании *m*<*w* исходных, так же, как и в оригинальном методе Соболя, будем использовать примитивные порождающие полиномы. В общем виде примитивный полином имеет вид ϕ(*x*)=1⊕λ1*x*1⊕λ2*x*2⊕ … ⊕λ*m*-1*xm*-1⊕*xm*, где *m* = degϕ(*x*), а двоичные коэффициенты λ*i∈*{0, 1}, определяют конкретный вид полинома. Подобный подход, когда на основании *m* исходных направляющих чисел и порождающего полинома ϕ(*x*) генерируются остальные *w*−*m* чисела, представлен в оригинальном методе Соболя и его классических модификациях, в том числе и в экономичном способе Антонова и Салеева.

Рассмотрим алгоритм формирования модифицированных направляющих чисел, представленных в виде нижней треугольной матрицы с единичной главной диагональю.

При реализации каждой итерации значение предыдущих (ранее полученных) направляющих чисел будем модифицировать с целью формирования двоичных кодов, разрядность которых увеличивается от первоначальных *m* бит до *w* результирующих бит. Формально это достигается сдвигом направляющего числа на один разряд влево с одновременным приписыванием в младшем разряде двоичного нуля. Подобная модификация математически записывается как



Значение μ*j* в левой части равенства есть новое значение направляющего числа μ*j*, полученное на основании его предыдущего значения. Само рекуррентное соотношение примет вид

 (2)

Значение μ*i-m+*1/2*m* представляет собой сдвинутую копию вправо на *m* позиций двоичного кода μ*i-m+*1.

Например, для случая последовательности Соболя длиной 2*w*–1=25–1=31, где *w*=5, используем примитивный полином ϕ(*x*)=1⊕*x*1⊕*x*3 степени *m*=3, а для первых *m*=3 направляющих чисел возьмем целые нечетные числа *m*1 = 1, *m*2 = 3 и *m*3 = 5. Тогда v1 =0,100, v2 = 0,110, v3 = 0,101 и соответственно μ1 = v123 = 100, μ2 = v223 = 110 и μ3 = v323 = 101. Так как *w*=5, остальные (*w*–*m*) направляющие числа, в данном случае 5–3=2 (два) числа, генерируются с использованием рекуррентного соотношения (2) для заданных значений *m* и *w*



Для *i*=3 получим μ1=μ12=1000, μ2=μ22=1100 и μ3=μ32=1010, кроме того, μ1/23=0001. Тогда μ4=μ3⊕μ1⊕(μ1/23)=1010⊕1000⊕0001=0011, и далее для *i*=4 получим μ5=μ4⊕μ2⊕(μ2/23) = 00110 ⊕ 11000 ⊕ 00011 = 11101.

Последовательность Соболя, соответствующая полученным направляющим числам μ1=10000, μ2=11000, μ3=10100, μ4=00110 и μ5=11101

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *n* | *Tq* | *xng* | *n* | *Tq* | *xng* | *n* | *Tq* | *xng* | *n* | *Tq* | *xng* |
| 1 | 1 | 10000 | 9 | 1 | 00010 | 17 | 1 | 01011 | 25 | 1 | 11001 |
| 2 | 2 | 01000 | 10 | 2 | 11010 | 18 | 2 | 10011 | 26 | 2 | 00001 |
| 3 | 1 | 11000 | 11 | 1 | 01010 | 19 | 1 | 00011 | 27 | 1 | 10001 |
| 4 | 3 | 01100 | 12 | 3 | 11110 | 20 | 3 | 10111 | 28 | 3 | 00101 |
| 5 | 1 | 11100 | 13 | 1 | 01110 | 21 | 1 | 00111 | 29 | 1 | 10101 |
| 6 | 2 | 00100 | 14 | 2 | 10110 | 22 | 2 | 11111 | 30 | 2 | 01101 |
| 7 | 1 | 10100 | 15 | 1 | 00110 | 23 | 1 | 01111 | 31 | 1 | 11101 |
| 8 | 4 | 10010 | 16 | 5 | 11011 | 24 | 4 | 01001 |  |  |  |

Второй модификацией последовательностей Соболя является использование в качестве направляющих чисел *w* последовательных значений *М*-последовательности, формируемых в соответствии с примитивным порождающим полиномом степени *w*. В отличие от классического метода формирования направляющих чисел, в данном случае генерируется все множество *w* чисел, а не только *w*–*m*, на основании *m* исходных чисел.

Для соблюдения всех свойств последовательностей Соболя необходимо, чтобы μ1 всегда равнялось *w*-разрядному коду 100…0. Для получения всего множества модифицированных направляющих чисел двоичный код 100…0 используем в качестве начального состояния генератора *M*-последовательности и сформируем *w*–1 последующих состояний генератора, которые будем использовать как μ2, μ3, …, μ*w*.

Например, для полинома *ϕ*(*x*)=1⊕*x*1⊕*x*4 степени *m*=4 направляющие числа принимают значения: μ1 =1000,μ2 =1100, μ3 =1110 и μ4 =1111. Изменив порождающий полином на ϕ(*x*)=1⊕*x*3⊕*x*4, получим новое множество модифицированных направляющих чисел, а именно μ1 =1000;μ2 =0100; μ3 =0010 и μ4 =1001. Для двух приведенных примеров соответствующие треугольные матрицы *V*1 и *V*2 с единичной главной диагональю, построенные на основании модифицированных направляющих чисел, имеют вид



Для каждого множества направляющих чисел, описываемых матрицами *V*1 и *V*2, может быть сформирована уникальная последовательность Соболя.

Очевидно, что количество множеств модифицированных порождающих чисел определяется количеством примитивных порождающих полиномов степени *w*. Для общего случая это количество вычисляется как Ф(2*w*–1)/*w*, где Ф есть функция Эйлера от целого числа 2*w*–1.

Следующей модификацией последовательностей Соболя является использование для формирования произвольных значений β-*j*(*i*)∈{0,1} для *j*<*i* последовательных (*w*2–*w*)/2 бит произвольной псевдослучайной последовательности, в том числе и *М*-последовательности. Возможность формирования подобным образом модифицированных направляющих чисел *μi* следует из того факта, что равенство единице младшего бита двоичного представления целого числа свидетельствует о его нечетности. Очевидно, что старшие биты нечетного целого числа могут принимать произвольные значения.

В случае модифицированных направляющих чисел, как отмечалось ранее, значение *μ*1 всегда равняется *w*-разрядному коду 100…00. Соответственно μ2=β-1(2)10…00, где β-1(2)∈{0, 1},μ3=β-1(3)β-2(3)10…00, где β-1(3) иβ-2(3)∈{0, 1}, и т. д. Значение μ*w*=β-1(*w*)β-2(*w*)β-3(*w*) …β-*w*+1(*w*)1, где β-*j*(*w*)∈{0,1} для *j*<*w*. Таким образом, β-*j*(*i*)∈{0,1} при *j*<*i* для модифицированных направляющих чисел μ*i* могут принимать произвольные двоичные значения. Количество возможных вариантов значенийβ-*j*(*i*)∈{0,1} при *j*<*i* определяется их числом (*w*2–*w*)/2 и вычисляется как . Это подтверждается анализом табл. 4 и матриц *V*1 и *V*2.

В качестве примера рассмотрим случай, когда *w*=4. Тогда число значенийβ-*j*(*i*)∈{0,1} при *j*<*i*, где , равняется (*w*2–*w*)/2=(42–4)/2=6. Это означает, что шесть элементов матрицы модифицированных направляющих чисел, находящихся под главной диагональю, могут принимать произвольные двоичные значения. Количество подобных матриц и соответственно множеств модифицированных направляющих чисел равняется  Две из указанных матриц *V*1 и *V*2 приводились ранее.

Максимально возможное количество уникальных множеств из *w* модифицированных направляющих чисел μ*i* равняется, а их элементы могут формироваться с использованием различных алгоритмов генерирования равновероятных двоичных цифр. В случае использования для этих целей *M*-последовательностей единственным ограничением на степень *m* порождающего полинома ϕ(*x*) является равенство degϕ(*x*)=(*w*2–*w*)/2. Выполнение данного равенства обеспечивает формирование различных вариантов значенийβ-*j*(*i*)∈{0,1} при *j*<*i* и соответственно всевозможных последовательностей Соболя для заданного значения *w*. Для случая, когда *w*=4, для порождающего полинома ϕ(*x*) необходимо, чтобы degϕ(*x*)=6, что позволит получить все возможные матрицы вида *V*1 и *V*2, которые будут отличаться значениями элементов, находящихся ниже главной диагонали.