МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра математического моделирования и анализа данных

ПОДОЛЬСКИЙ Владислав Олегович

РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ТЕСТИРОВАНИЯ СЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Магистерская диссертация

специальность 1-31 81 12 «Прикладной компьютерный анализ данных»

Научный руководитель Михаил Семенович Абрамович
кандидат физико-математических наук,
доцент

Допущена к защите
«___»_____ 2016 г.
Зав. кафедрой математического
моделирования и анализа данных
_____ Ю.С. Харин
доктор физико-математических наук,
профессор,
член-корреспондент НАН Беларуси

Минск, 2016

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ **4**

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ **5**

ВВЕДЕНИЕ **8**

1 ГЕНЕРАТОРЫ СЛУЧАЙНЫХ ЧИСЕЛ **10**

1.1 Генераторы псевдослучайных чисел 10

1.2 Физические генераторы случайных чисел 11

1.3 Основные требования к генераторам случайных чисел 14

2 ОБЗОР ФИЗИЧЕСКИХ УСТРОЙСТВ И ПРОЦЕССОВ, ИСПОЛЬЗУЕМЫХ ДЛЯ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ЧИСЕЛ **16**

2.1 Полупроводниковый шумовой диод 16

2.2 Генератор GRANG 17

2.3 Интернет-ресурс HotBits 18

2.4 Интернет-ресурс Random.org 19

2.5 Квантовые генераторы случайных последовательностей 20

2.6 Генератор LavaRand 21

3 АЛГОРИТМЫ ОЦЕНКИ КАЧЕСТВА ИСТОЧНИКОВ ЭНТРОПИИ **22**

3.1 Минимальная энтропия 22

3.2 Оценивание энтропии выходных последовательностей источника случайности 23

3.3 Проверка выходных последовательностей на независимость и одинаковую распределенность 23

3.4 Тесты выходной последовательности на независимость и одинаковую распределенность 24

3.4.1 Тесты перестановок для проверки независимости и одинаковой распределенности 24

3.4.2 Хи-квадрат тесты для проверки независимости и одинаковой распределенности 31
3.5 Оценка энтропии выходной последовательности 36
3.5.1 Оценка минимальной энтропии последовательности независимых и одинаково распределённых случайных величин 36
3.5.2 Оценка минимальной энтропии последовательности случайных величин, не являющихся независимыми и одинаково распределёнными 37

4 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТЕСТИРОВАНИЯ ВЫХОДНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ИСТОЧНИКОВ СЛУЧАЙНОСТИ И ЕГО ПРИМЕНЕНИЕ 46
4.1 Описание пакета программ для тестирования выходных последовательностей источников случайности 46
4.2 Тестирование выходных последовательностей источников случайности 47

ЗАКЛЮЧЕНИЕ 50

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 50

ПРИЛОЖЕНИЕ А 53

ПРИЛОЖЕНИЕ Б 73

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ГИСЧ	–	генератор истинно случайных чисел
ГПСЧ	–	генератор псевдослучайных чисел
ГСЧ	–	генератор случайных чисел
н.о.р.	–	независимые и одинаково распределённые
РРСП	–	равномерно распределенная случайная последовательность

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Ключевые слова: ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ, ЭНТРОПИЯ, ГЕНЕРАТОР ИСТИННО СЛУЧАЙНЫХ ЧИСЕЛ, ИСТОЧНИК СЛУЧАЙНОСТИ, ИСТОЧНИК ЭНТРОПИИ, ОЦЕНКА ЭНТРОПИИ, МИНИМАЛЬНАЯ ЭНТРОПИЯ.

Цель работы — разработка алгоритмов и программного обеспечения оценки энтропии выходных последовательностей источников случайности.

Актуальность работы заключается в том, что к источникам случайности предъявляются особые требования по количеству энтропии (например, в СТБ 34.101.27-2011), но отсутствовали методика и программные средства оценки энтропии.

Объект исследования — выходные последовательности источников случайности, предмет — количество неопределенности в последовательностях, их статистические свойства.

В результате работы описаны подходы и алгоритмы для оценки качества источников случайности, разработано универсальное программное обеспечение для этой оценки. Показана практическая применимость разработанного программного обеспечения.

Диссертация состоит из четырёх глав: в первой описаны общие сведения и понятия, во второй проведён обзор существующих источников случайности, в третьей описаны методы и алгоритмы оценки качества источников случайности, в четвёртой описано разработанное программное обеспечение и приведены результаты его применения. В диссертацию включены два приложения. Объём работы — 53 листа, объём приложений — 21 лист. Количество источников — 15.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Ключавыя словы: ГЕНЕРАТАР ВЫПАДКОВЫХ ЛІЧБАЎ, ЭНТРАПІЯ, ГЕНЕРАТАР САПРАЎДНЫХ ВЫПАДКОВЫХ ЛІЧБАЎ, КРЫНІЦА ВЫПАДКОВАСЦІ, КРЫНІЦА ЭНТРАПІІ, АЦЭНКА ЭНТРАПІІ, МІНІМАЛЬНАЯ ЭНТРАПІЯ.

Мэта працы — распрацоўка алгарытмаў і праграмнага забеспячэння ацэнкі энтрапіі выходных паслядоўнасцяў крыніц выпадковасці.

Актуальнасць працы складаецца ў тым, што да крыніц выпадковасці прад'яўляюцца асаблівыя патрабаванні па колькасці энтрапіі (напрыклад, у СТБ 34.101.27-2011), але адсутнічалі методыка і праграмныя сродкі ацэнкі энтрапіі.

Аб'ект даследавання — выходныя паслядоўнасці крыніц выпадковасці, прадмет — колькасць нявызначанасці ў паслядоўнасцях, іх статыстычныя ўласцівасці.

У выніку працы апісаны падыходы і алгарытмы для ацэнкі якасці крыніц выпадковасці, распрацавана ўніверсальнае праграмнае забеспячэнне для гэтай ацэнкі. Паказаная практычная дастасавальнасць распрацаванага праграмнага забеспячэння.

Дысертацыя складаецца з чатырох раздзелаў: у першым апісаны агульныя звесткі і паняцці, у другім праведзены агляд існуючых крыніц выпадковасці, у трэцім апісаны метады і алгарытмы ацэнкі якасці крыніц выпадковасці, у чацвёртым апісана распрацаванае праграмнае забеспячэнне і прыведзены вынікі яго прымянення. У дысертацыю ўключаны два прыкладання. Аб'ём працы — 53 лісты, аб'ём прыкладанняў — 21 ліст. Колькасць крыніц — 15.

ABSTRACT

Keywords: RANDOM NUMBER GENERATOR, ENTROPY, TRUE RANDOM NUMBER GENERATOR, RANDOMNESS SOURCE, ENTROPY SOURCE, ENTROPY ESTIMATION, MINIMAL ENTROPY.

The goal of the thesis is development of entropy estimation algorithms and software for output sequences of sources of randomness.

The relevance of the work lies in the fact that randomness sources have special requirements to the amount of entropy (e.g., STB 34.101.27-2011), but there was no methodology and software for entropy evaluation.

The objects of research are output sequences of randomness sources, subject is the amount of uncertainty in the sequences and their statistical properties.

As a result, the thesis describes the approaches and algorithms for evaluating the quality of randomness sources, and universal software for this assessment was developed. The practical applicability of the developed software was shown.

The thesis consists of four chapters: the first describes the basic facts and concepts; the second conducts a review of existing randomness sources; the third describes the methods and algorithms for evaluating the quality of randomness sources; developed software and the results of its application was described in the fourth chapter. The thesis includes two appendices. The volume of thesis is 53 sheets, the volume of appendices is 21 sheets. Number of sources is 15.

ВВЕДЕНИЕ

Случайные числа и их генераторы являются неотъемлемыми элементами современных криптосистем. Почти все криптографические протоколы требуют генерации и использования некоторых секретных значений, которые должны быть неизвестны атакующему. Например, генераторы случайных чисел нужны для генерации пар открытых и секретных ключей, использующихся в асимметричных (с открытым ключом) алгоритмах, включая RSA, DSA и алгоритм Диффи-Хеллмана. Сеансовые и другие ключи для симметричных (таких как DES, ГОСТ 28147–89, Blowfish) и гибридных криптосистем также формируются случайно. ГСЧ также используются для создания вызовов в системах вызов-ответ, одноразовых кодов и пр. Схема одноразовых блокнотов (шифр Вернама) – единственная система шифрования, для которой доказана абсолютная криптографическая стойкость – использует столько ключевого материала, сколько занимает шифротекст и требует, чтобы ключевой поток генерировался при помощи истинно случайного процесса. Приведем больше примеров использования случайных чисел в криптологии [2]:

- вектор инициализации для блочных криптосистем, работающих в режиме обратной связи;
- случайные значения параметров для многих систем ЭЦП, например DSA или белорусского стандарта СТБ 1176.2–99;
- случайные параметры в протоколах аутентификации, например в протоколе «Цербер» («Kerberos»);
- случайные параметры протоколов для обеспечения уникальности различных реализаций одного и того же протокола, например в протоколах SET и SSL.

Отметим, что для некоторых из этих криптографических применений необходимы огромные массивы случайных чисел, используемых конфиденциально. Например, в протоколе «Цербер» сетевой сервер генерирует тысячи сессионных ключей ежечасно.

К сожалению, компьютеры по своей конструкции являются детерминированными системами, поэтому на современных компьютерах генерация случайных чисел весьма затруднительна. Для получения случайных чисел используются детерминированные алгоритмы (ГПСЧ), которые принимают на

вход некоторое начальное значение и генерируют на его основе псевдослучайную последовательность. Когда существует необходимость генерировать совершенно непредсказуемые, абсолютно случайные числа, используют ГПСЧ с источником энтропии, также называемые просто ГСЧ. В то время, как существуют обширные исследования по вопросу генерации псевдослучайных битов, используя ГПСЧ и неизвестное начальное значение, создание такого неизвестного начального значения никогда не было должным образом описано. Единственная возможность для этого начального значения обеспечить реальную безопасность – это содержать достаточное количество случайности, например, из недетерминированного процесса – источника энтропии. В частности, в государственном стандарте Республики Беларусь СТБ 34.101.27-2011 [3] («Требования безопасности к программным средствам криптографической защиты информации») предъявляются следующие требования по генерации случайных чисел: для каждого генератора случайных чисел должна быть проведена оценка энтропии всех его источников случайности; способ обработки данных от источников случайности должен гарантировать, что собранные данные содержат достаточно неопределенности для надежной генерации критического объекта. Однако, методика тестирования генераторов и оценки энтропии не описана.

В магистерской диссертации описаны свойства, которыми источник энтропии должен обладать, чтобы быть пригодным для использования в криптографических генераторах случайных битов, проведен обзор существующих физических источников случайности. Также разработаны статистические тесты, используемые для определения качества источника энтропии по его выходным последовательностям. Разработано универсальное программное обеспечение для тестирования источников случайности. Разработанное ПО применено для оценки энтропии выходных последовательностей кольцевого осциллятора.

ГЛАВА 1
ГЕНЕРАТОРЫ СЛУЧАЙНЫХ ЧИСЕЛ

1.1 Генераторы псевдослучайных чисел

Большинство источников «случайных» чисел используют генератор псевдослучайных чисел (ГПСЧ). ГПСЧ, чтобы сгенерировать ряд выходов из начального состояния (зерна), используют детерминированные процессы. Из-за того, что выход является просто значением функции от данных начального состояния, фактическая энтропия выхода никогда не может превысить энтропии зерна. Однако, не всегда можно вычислительно отличить хороший ГПСЧ от идеального ГСЧ.

Например, ГПСЧ с 256 битами энтропии в зерне (с 256-битным начальным состоянием) не может произвести более 256 бит истинно случайных данных. Тот атакующий, который может подобрать верное начальное значение, может предсказать весь выход ГПСЧ. Подбор 256-битного начального значения вычислительно невыполним, таким образом, такой ГПСЧ можно использовать в криптографических приложениях. Например, в некоторые криптографические наборы инструментов включаются ГПСЧ, спроектированные специально для криптографических приложений: MD5Random и SHA1Random (которые, соответственно, имеют 128- и 160- битные состояния).

Несмотря на то, что должным образом реализованные, с правильно выбранным зерном ГПСЧ пригодны для использования в большинстве криптографических приложений, большое внимание должно быть уделено разработке, тестированию и выбору алгоритмов ГПСЧ. Очень важно, чтобы зерно для ГПСЧ было получено из надежного источника. Например, большинство ГПСЧ, включенных в стандартные программные библиотеки, используют предсказуемое начальное значение или дают выход, отличимый от случайных данных [6]. Известно в сфере исследователей генераторов случайных чисел высказывание Джона фон Неймана: *«Всякий, кто питает слабость к арифметическим методам получения случайных чисел, грешен вне всяких сомнений»*.

1.2 Физические генераторы случайных чисел

Генератор истинно случайных чисел (ГИСЧ) использует для получения случайности недетерминированный источник. Энтропия, надежность и производительность зависят от устройства ГИСЧ.

Например, аппаратные (физические) генераторы случайных чисел используют случайность, которая появляется в некоторых физических явлениях. Физический генератор – устройство, которое генерирует последовательность случайных чисел на основе измеряемых параметров протекающего физического процесса. Работа таких устройств часто основана на использовании надежных источников энтропии, таких как тепловой шум, фотоэлектрический эффект, квантовые явления и т.д. Чаще всего физические генераторы применяются в системах компьютерного моделирования для имитации стохастических процессов и в криптографии для выработки криптографических ключей.

Основная проблема физических генераторов заключается в том, что первичная последовательность генератора в большинстве случаев не обладает свойствами независимости и равномерной распределенности. Это вызвано особенностями физических процессов, используемых в генераторе. Данная проблема решается с помощью специальных алгоритмов, которые позволяют «улучшить» свойства первичной последовательности: например, выровнять долю нулей и единиц в битовой последовательности, увеличить уровень энтропии и пр. На этапе функционального преобразования первичная последовательность генератора преобразуется в выходную последовательность. Необходимо отметить, что применение специального функционального преобразования не является обязательным, если первичная последовательность уже обладает всеми необходимыми свойствами.

Примеры используемых физических явлений [7]:

- время, прошедшее между эмиссиями частиц во время радиоактивного распада;
- тепловой шум от полупроводникового диода или резистора;
- атмосферный шум;
- неустойчивость частоты осциллятора;
- величина на которую полупроводниковый конденсатор заряжается в течение фиксированного периода времени;
- турбулентность воздуха в герметичном дисководе, которая вызывает случайные флуктуации задержки чтения с диска;

- звук с микрофона или вход с камеры.

Генераторы, основанные на первых двух явлениях, вообще говоря, должны быть установлены вне устройства, использующего случайные биты. Следовательно, они могут стать целью злоумышленника. Генераторы, основанные на осцилляторах и конденсаторах, могут быть встроены в устройства; они могут быть помещены в защищенные от несанкционированного вмешательства аппаратные части, и следовательно защищены от активных атак. Более подробный обор существующих физических генераторов приведен далее в данной работе.

ГИСЧ сам по себе будет небезопасен, ему требуется ГПСЧ, которому он будет предоставлять начальные значения. Получение начальных значений требует источник истинной случайности, так как невозможно получить истинную случайность в детерминированной системе. Источник энтропии – ключевая компонента генератора, полностью обеспечивающая неопределенность выходной битовой последовательности. Выходные данные источника шума оцифровываются и представляют собой битовые строки.

Генераторы случайных чисел, использующие в качестве источника шума физические случайные процессы, позволяют получить случайные числа, но их производство относительно сложно и дорого. На компьютерах без аппаратного ГСЧ, программисты обычно пытаются получить энтропию для данных начального значения используя существующие периферийные устройства. Разработка программных генераторов более сложная задача. Они могут быть основаны на следующих процессах [6], [7]:

- системные часы;
- задержки между нажатиями кнопок клавиатуры или движениями мыши;
- содержимое буферов;
- пользовательский ввод;
- величины ОС, такие как загрузка системы или статистика сети;
- данные конфигурации и т.д.

Самые распространенные техники включают наблюдения за действиями пользователя, но эти методы неудобные и медленные. Например, существует приложение, требующее от пользователя для генерации нового ключа потратить примерно 15 секунд нажимая на клавиатуре случайные клавиши или случайным образом передвигая мышь. Такие методы неудобны для пользователей, а также не могут быть использованы (или становятся небезопасными), когда контролируются автоматизированными скриптами. Некоторые

приложения используют длительность обращения к жесткому диску, но такие факторы, как технология диска, дисковые кэши и пределы разрешения системного таймера требуют осторожного рассмотрения.

В общем и целом, генераторы истинно случайных чисел, реализованные при помощи обычных аппаратных средств, как правило медленные, сложны в реализации, требуют участия пользователя и часто предоставляют неизвестное количество истинной энтропии. В этих методах также делаются предположения об аппаратных средствах, выполнение которых не гарантировано. Поведение вышеназванных процессов может значительно изменяться в зависимости от различных факторов, таких как компьютерная платформа. Также может быть трудно предотвратить наблюдение или манипуляции со стороны противника. Например, если противник имеет примерное представление о том, когда случайная последовательность была сгенерирована, он может угадать значение системных часов в то время с высокой степенью точности. А недостатком техник, полагающихся на участие пользователя является то, что они не могут быть надежными в системах, в которых нет постоянного присутствия пользователя, таких как серверы. Хорошо продуманный генератор должен использовать столько хороших источников случайности, сколько доступно. Использование нескольких источников предохраняет от возможного контроля за некоторыми из них со стороны противника или их отказа.

Несмотря на то, что у приложений есть возможность производить собственные безопасные случайные данные, не всегда это получается. Обзоры исследователей часто идентифицируют слабые места в генераторах случайных чисел. Чаще всего проблемным местом является выбор зерна. Известный в криптографических кругах Брюс Шнайер пишет [8]: *«Хорошие генераторы случайных чисел сложны в разработке, так как их безопасность часто зависит от используемых ими характеристик программного и аппаратного обеспечения. Большинство из исследованных мной продуктов используют плохие».*

Например, генератор широко известной компании Netscape использовал в качестве зерна производное только от трех значений: времени суток, ID процесса и ID родительского процесса. То есть, злоумышленник, который мог спрогнозировать три эти значения, мог применить широко известный алгоритм MD5 и вычислить точное значение зерна. Несмотря на то, что о большинстве недостатков ГСЧ не сообщается, проблемы проявляются достаточно часто. Частично из-за того, что в криптографических библиотеках поиск надежного источника зерна возложен на плечи программистов.

В большинстве случаев разработчики систем вынуждены выбирать между безопасностью и удобством. Например, чтобы избежать потребности при каждой загрузке программы собирать свежие данные для формирования зерна,

многие приложения хранят такие данные на жестком диске, где существует большой риск их компроментации.

1.3 Основные требования к генераторам случайных чисел

Т.к. протоколы безопасности полагаются на непредсказуемость используемых ключей, генераторы случайных чисел для криптографических приложений должны удовлетворять строгим требованиям. Самое важное – это то, что атакующий, даже зная конструкцию ГСЧ, не должен иметь возможности сделать какие-либо полезные предположения о выходных значениях ГСЧ. В частности, двоичная энтропия выхода ГСЧ должна быть как можно больше близка к его битовой длине.

По Шеннону [9], энтропия H любого сообщения вычисляется как:

$$H = -\sum_{i=1}^{n} p_i \log p_i, \tag{1.1}$$

где p_i – это вероятность состояния i из n возможных состояний. В случае, когда генератор случайных чисел обеспечивает k-битный результат, p_i - вероятность того, что выходное значение будет равняться i, где $0 \leq i < 2^k$. Таким образом, для идеального генератора случайных чисел $p_i = 2^{-k}$ и энтропия выхода равна k битам. Это означает, что все возможные исходы равновероятны, а информация, представленная в выходе, в среднем не может быть представлена последовательностью длиной короче, чем k бит. В сравнение, энтропия типичного текста на английском языке $1,5$ бита на символ [7].

ГСЧ для криптографического применения должен казаться вычислительно-ограниченным противникам как можно более неотличимым от идеального ГСЧ. Иначе говоря, проблема генерации случайной последовательности с произвольным законом распределения вероятностей сводится к проблеме генерации так называемой равномерно распределенной случайной последовательности (РРСП), или, как ее часто называют в криптографических приложениях, «чисто случайной» последовательности.

Генератор РРСП – устройство, позволяющее по запросу получить реализацию равномерно распределенной случайной последовательности $x_1, \ldots, x_n \in \mathfrak{A}$ длиной $n \in \mathbb{N}$; элементы $x_1, \ldots, x_n$ этой реализации принято называть случайными числами. Существует три типа генераторов РРСП [2]: табличный, физический и программный. Физический генератор случайных битов требует наличия источника случайности естественного происхождения. Разработка устройства или программы, использующих этот источник случайности для генерации несмещенной и некоррелированной битовой после-

довательности довольно сложная задача. К тому же, для большинства криптографических приложений генератор не должен наблюдаться противником или подвергаться манипуляциям с его стороны. Генераторы случайных битов, основанные на естественных источниках случайности подвержены влиянию внешних факторов, а также неполадкам. Поэтому они должны периодически проходить проверки.

ГЛАВА 2
ОБЗОР ФИЗИЧЕСКИХ УСТРОЙСТВ И ПРОЦЕССОВ, ИСПОЛЬЗУЕМЫХ ДЛЯ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ЧИСЕЛ

2.1 Полупроводниковый шумовой диод

Полупроводниковый шумовой диод – это прибор, являющийся источником шума использующий стабилитрон с заданной спектральной плотностью в определенном диапазоне частот. Правильно используемый шумовой диод является лучшим генератором шума на частотах, где еще не сказываются высокочастотные погрешности, вплоть до 300-400 мегагерц [4].

Начальная граница для шумового диода определяется фликкер-эффектом в нем и находится в районе 1000 герц для диода с торированным вольфрамовым катодом и еще ниже для диода с чисто вольфрамовым катодом.

Конструктивно шумовой диод мало отличается от обыкновенного диода. Однако получение максимального напряжения флуктуации в обыкновенном диоде затруднено наличием так называемого пространственного заряда, который образуется вблизи катода из облака электронов при избыточной эмиссии с катода. Облако электронов стабилизирует ток анода и тем самым уменьшает ток флуктуации.

При конструировании самих шумовых диодов стараются уменьшить межэлектродные емкости, выводы электродов делают минимально короткими и разносят их возможно дальше друг от друга. Подобные меры позволяют снизить значение шунтирующей емкости до нескольких десятых долей пикофарад.

Для получения напряжения помех в шумовых диодах используется явление единичного лавинного пробоя. При протекании тока в полупроводниках, носители заряда испытывают большое число столкновений с атомами и другими носителями внутри катода, в результате чего величина и направление скорости отдельного носителя при пролете через какое-либо сечение полупроводника могут быть различными. Это приводит к тому, что число носителей, прошедших через сечение за одинаковые малые промежутки времени, оказывается различным, вследствие чего ток испытывает случайные отклонения от своего среднего значения (флуктуации). Энергия шума дро-

бового эффекта равномерно распределена по частотному спектру, мощность шума пропорциональна полосе частот, в которой этот шум измеряется. Если диод работает в режиме насыщения, его располагаемая шумовая мощность известна и регулируема. Она ограничена допустимой мощностью рассеяния (попытки увеличить последнюю снижают верхний предел частоты).

Пространственный заряд уничтожается при переходе к режиму насыщения. Для этого анодное напряжение поддерживается достаточно высоким, а эмиссия с катода уменьшается путем понижения температуры катода. Все излученные электроны достигают анода, ток диода становится стабильным и между напряжением дробового эффекта и полным током диода устанавливается устойчивое соотношение.

Режим насыщения характеризуется постоянством тока анода при изменении анодного напряжения в определенных пределах. Начиная с некоего значения анодного напряжения (обычно это около 100 В) линия графика анодного тока идет почти параллельно линии координат анодного напряжения [4]. Следовательно, при проектировании схемы шумового генератора анодное напряжение следует выбрать с запасом порядка 25 В для исключения возможности выхода диода из режима насыщения в случае понижения питающего напряжения (например, при колебаниях сетевого напряжения).

2.2 Генератор GRANG

GRANG – Genuine Random Number Generator (генератор подлинных случайных чисел). Компания LE Tech выпускает несколько генераторов истинно случайных чисел в форме PCI плат и встраиваемых модулей. В эти генераторы встроены тесты проверки на случайность чисел, а также для них характерна высокая скорость генерации чисел. Данные модули разрабатывались на протяжении 5 лет и теперь ежедневно используются в военной сфере и приложениях для защиты облачных вычислительных технологий. Методы, которые они используют для генерации и тестирования случайных чисел, являются уникальными.

Существует много способов для генерации случайных последовательностей с использованием физических процессов в качестве источника. Тем не менее, нельзя достоверно утверждать, что такие последовательности являются действительно случайными, т.к. некоторые источники периодических сигналов, такие как мобильные телефоны, могут повлиять на сигналы, поступающие от процесса. Ранее не было никаких точных способов проверить, что поступающие сигналы не искажены. Но генератор GRANG решает данную проблему, непрерывно проверяя случайность последовательности, которая должна иметь Пуассоновское распределение. В случае если на генератор

воздействует шум окружающей среды или атаки злоумышленника, то происходит отклонение от распределения и такие числа отбрасываются.

Есть известные статистические тесты на случайность, такие как NIST SP800-22. Случайные последовательности, генерируемые GRANG, легко проходят этот тест. Даже если ради эксперимента отключить функцию самодиагностики и намеренно ввести периодические шумы, выходные данные все равно проходят тесты NIST, хотя, конечно, с включенной функцией данные будут отклонены, потому что не удовлетворяют ожидаемому распределению. Таким образом, внутренняя функция самотестирования позволяет получать более достоверные числа.

GRANG может генерировать числа с частотой от 50 до 550 Мб/с с помощью платы и от 2 до 7 Мб/с с помощью небольшого модуля [11]. Более высокая частота может быть легко получена.

В основе работы GRANG лежит использование теплового шума (белого шума) от работы электрического сопротивления, который генерирует случайную последовательность импульсов, затем измеряется интервалы между импульсами, которые преобразовываются в числовые значения с помощью счетчика. Далее полученные данные проверяются в режиме реального времени на соответствие экспоненциальному распределению.

Принцип работы [11]:
Источником является тепловой шум.
Шумовая волна подвержена внешнему воздействию, если она выходит за пределы диапазона амплитуд (за верхнюю красную линию).
Шум усиливается и конвертируется в импульсные интервалы (T1, T2, ...)
Шумовая волна преобразуется в импульсы с помощью порогового уровня.
Эти импульсы независимы друг от друга и поэтому должны удовлетворять Пуассоновскому распределению.
Данные проверяются на соответствие экспоненциальному распределению.

2.3 Интернет-ресурс HotBits

HotBits – интернет-ресурс, позволяющий сгенерировать настоящие случайные числа на основе данных радиоактивного распада.

Сервис разработан в 1996 году Джоном Уокером, основателем Autodesk, Inc. и соавтором AutoCAD. Для измерения данных радиоактивного распада используется коммерческий счетчик Гейгера-Мюллера американской фирмы Aware Electronics модели RM-80 и диск с Cs-137 в качестве источника радиации [12].

Присутствует поддержка протокола HTTPS для шифрования запросов пользователей и отправляемой случайной информации. Сервис также предо-

ставляет бесплатную программу для тестирования псевдослучайных последовательностей, randomX-пакет для Java с реализацией нескольких генераторов псевдослучайных последовательностей и возможностью генерации с помощью HotBits (удаленно) и драйвер с исходным кодом для генерации с помощью счетчика Гейгера-Мюллера фирмы Aware серии RM (локально).

Генерация основана на невозможности предсказания точного момента совершения следующего радиоактивного распада. Зная ожидаемый период полураспада нашего источника радиации и многие другие детали, такие, как, например, чувствительность детектора к данному типу распада, мы можем генерировать случайные биты сравнивая реальное и ожидаемое время между парой распадов. Но теоретический расчет ожидаемого времени очень трудоемок, т.к. зависит от очень большого числа различных параметров и неудобен на практике. Процесс можно упростить (и данный сервис использует именно этот вариант), используя тот факт, что из непредсказуемости точного времени совершения распада следует непредсказуемость точного интервала между распадами, благодаря чему мы можем генерировать случайные биты просто сравнивая время между двумя последовательными парами распадов.

Скорость генерации зависит от активности источника радиации. В качестве источника можно использовать в том числе и фоновую радиацию, но в таком случае из-за низкой активности скорость генерации может быть недостаточной. Для ускорения работы можно приобрести диск с радиоактивным изотопом. При удаленной генерации основным источником служит диск с Cs-137 и скорость составляет около 100 байт в секунду [12]. В силу того, что даже этого может быть недостаточно для удовлетворения всех запросов всех пользователей, сервер заполняет выдачу предварительно сгенерированными битами, но при этом однажды отправленные вам данные немедленно удаляются и никогда более не будут получены кем-либо другим.

2.4 Интернет-ресурс Random.org

Используемый физический источник случайности – атмосферный шум, который довольно легко уловить обычным радиоприемником. Радио настроено на частоту, на которой никто не вещает. Принятый атмосферный шум передается на рабочую станцию Sun SPARC через микрофонный порт, где он обрабатывается как восьмибитный моно-сигнал на частоте 8КГц. Верхние семь бит каждого отсчёта сразу отбрасываются, остальные превращаются в поток бит с высокой энтропией. Над битовым потоком применяется преобразование коррекции уклонения, чтобы обеспечить примерно равное распределение 0 и 1 [15].

2.5 Квантовые генераторы случайных последовательностей

Свет состоит из элементарных «частиц», называемых фотонами. В определенной ситуации фотоны демонстрируют случайное поведение. Одна из таких ситуаций, которая очень хорошо подходит для генерации бинарных случайных чисел, заключается в следующем. На полупрозрачное зеркало направляются фотоны, генерируемые источником одиночных фотонов. Фотон может отразиться, а может пройти через полупрозрачное зеркало с вероятностью 50%. Выбор, который «делает» фотон, абсолютно случаен. На выходе системы стоят два счетчика фотонов, регистрирующих прошедшие и отраженные фотоны и формирующих выходные электрические сигналы.

Кроме оптической части – «сердца» генератора случайных чисел – система включает в себя подсистему, которая управляет синхронизацией, а также сбором и обработкой данных, поступающих с детекторов, выполняет статистические и аппаратные проверки последовательности. Как отмечалось выше, физические процессы трудно точно сбалансировать. Таким образом, трудно гарантировать, что вероятность записи 0 и 1 в точности равна 50%. Например, в генераторе Quantis разница между этими двумя вероятностями меньше 10%, что эквивалентно вероятностям от 45% до 55% [14]. Поскольку это смещение не может быть приемлемым для некоторых приложений, то обязательно выполняется алгоритм пост-обработки для удаления смещения в последовательности случайных чисел. Как уже говорилось выше, одним из главных преимуществ квантовых генераторов случайных чисел на оптических фотонах является то, что они основаны на простом и принципиально случайном процессе, который легко моделировать и контролировать. Подобные квантовые генераторы имеют высокую скорость выходного потока – до 10-16 Мбит/с, – при которой не наблюдается никаких корреляций и выполняются все статистические тесты [14]. Еще одним примером является генератор случайных чисел с использованием полупроводникового лазера с короткими и резкими пиками интенсивности. Лазер пропускается через среду с обратной связью с задержкой, то есть интенсивность излучения на выходе определяется интенсивностью сигнала на входе и состоянием среды, которое зависит от интенсивности на входе. Известно, что изменение интенсивности – процесс квазипериодический, то есть с течением времени почти повторяется, поэтому напрямую использовать его в качестве генератора случайных чисел нельзя.

Для того чтобы избавиться от квазипериодичности, интенсивность излучения замеряется примерно 2.5 миллиарда раз в секунду.

Результат каждого измерения записывается в строку длиной в 8 бит. Он вычитается из значения предыдущего измерения, а результат усекается. Та-

ким образом, удается избавиться от квазипериодичности и добиться генерации случайного потока нулей и единиц со скоростью примерно 12.5 Гигабит в секунду. Благодаря своей вероятностной природе квантовая физика является идеальным источником случайности.

2.6 Генератор LavaRand

Генератор случайных чисел LavaRand был создан в 1996 году в Silicon Graphics. Его работа основана на извлечении случайных данных из цифровых изображений лавовых ламп.

Лавовая лампа – это декоративный светильник, который представляет собой емкость с маслом и парафином, снизу которой расположена лампа накаливания. Когда лампа находится в охлажденном состоянии, то парафин, имея большую, чем у воды, плотность, «тонет» в ней. При включении лампы накаливания парафин, нагреваясь, становится легче и всплывает. В верхней части лампы он остывает, начинает тонуть, и процесс повторяется. Каждую секунду внешний вид содержимого лампы меняется непредсказуемым образом.

Как производится съем и обработка наблюдений в LavaRand [13]:

1. В помещении настраиваются лавовые лампы (как правило, 6 ламп).

2. Перед лампами устанавливается цифровая камера, которая через некоторый фиксированный интервал времени делает снимки ламп. Сама камера также вносит цифровой шум в изображения. И весьма вероятно, что две фотографии одного и того же объекта будут представлять собой различные изображения.

3. С помощью алгоритма SHA-1 изображение обрабатывается и сжимается. Полученное хеш-значение подается на выход.

ГЛАВА 3
АЛГОРИТМЫ ОЦЕНКИ КАЧЕСТВА ИСТОЧНИКОВ ЭНТРОПИИ

3.1 Минимальная энтропия

Для того, чтобы разработать источник энтропии, который обеспечивает достаточное количество энтропии на каждую выходную битовую строку необходимо точно оценить количество энтропии, которое может быть получено путем обработки наблюдений его оцифрованного источника случайности.

Если известно, что оцифрованный выход источника случайности является смещенным, то в конструкцию источника могут быть включены соответствующие обрабатывающие функции, чтобы уменьшить эту смещение до приемлемого уровня, прежде чем биты попадут на выход источника энтропии. А также, если установлено, что используемый источник энтропии обеспечивает энтропию в размере по крайней мере $1/2$ бита энтропии на бит оцифрованного наблюдения, то эта оценка, вероятно, будет отражена в количестве наблюдений, которые комбинируются с помощью обрабатывающих компонентов для производства битовых строк с таким количеством энтропии, которое отвечает требованиям к проектированию источников энтропии.

Итак, основная математическая концепция, лежащая в оценке генераторов случайных бит – энтропия. Энтропия определяется относительно чьего-либо знания X до непосредственного наблюдения и отражает неопределенность, связанную с прогнозированием значения X – чем больше энтропия, тем больше неопределенность в прогнозировании значения наблюдения. Есть много вариантов измерения энтропии; будем использовать традиционную меру – *минимальную энтропию*, которая определяет сложность угадывания наиболее вероятного выхода источника энтропии.

Минимальная энтропия часто используется в качестве показателя неопределенности, связанного с наблюдениями X в наихудшем случае: если X имеет минимальную энтропию m, то вероятность наблюдения какого-либо конкретного значения не больше, чем 2^{-m}. Пусть x_i – это оцифрованное наблюдение от источника случайности, которое представлено одним или более битами, пусть $x_1,\ x_2,\ \ldots,\ x_M$ – выходы источника случайности, и пусть $p(x_i)$ есть вероятность того, что x_i получен в любой момент получения образцов. Тогда

минимальная энтропия выходов определяется формулой [5]:

$$-\log_2(\max\ p(x_i)). \tag{3.1}$$

Это соответствует работе злоумышленника, пытающегося угадать выход источника шума, в наилучшем для него случае. Максимальным возможным значением минимальной энтропии случайной величины с k различными значениями является $\log_2 k$, которое достигается когда случайная величина имеет равномерное распределение вероятностей, т.е., $p_1 = p_2 \ldots = p_k = 1/k$.

3.2 Оценивание энтропии выходных последовательностей источника случайности

Оценка энтропии выходных последовательностей состоит в применении специальных тестов для оценки минимальной энтропии. Тесты описаны в 3.5.

Оценка энтропии осуществляется по-разному для выходных последовательностей, являющихся последовательностью н.о.р. случайных величин и не являющихся таковыми.

Если для некоторых тестов недостаточно данных и нет возможности получить для проверки выходную последовательность большей длины, то данные тесты пропускаются и их результаты не учитываются.

Оценка минимальной энтропии выходной последовательности равна наименьшей из всех оценок минимальной энтропии, полученных различными тестами. Оценка минимальной энтропии источника случайности равна наименьшей оценке минимальной энтропии среди всех исследуемых выходных последовательностей.

3.3 Проверка выходных последовательностей на независимость и одинаковую распределенность

Проверка выходных последовательностей на н.о.р. состоит в применении статистических тестов, описанных в 3.4, для проверки гипотезы о том, что выходная последовательность является последовательностью н.о.р. случайных величин.

Если для некоторых тестов недостаточно данных и нет возможности получить для проверки выходную последовательность большей длины, то данные тесты пропускаются и их результаты не учитываются.

Гипотеза о том, что выходная последовательность источника случайности является последовательностью н.о.р. случайных величин, принимается, если все исследуемые выходные последовательности проходят все тесты. Если

какая-либо выходная последовательность не проходит хотя бы один из тестов, то данная гипотеза отвергается.

При наличии различных режимов работы источника случайности гипотеза о н.о.р. проверяется отдельно для каждого режима работы. При этом проверяются только выходные последовательность, полученные в данном режиме работы.

Дальнейшее тестирование осуществляются согласно принятой по результатам проверки гипотезе.

3.4 Тесты выходной последовательности на независимость и одинаковую распределенность

В данном разделе описаны тесты, применяемые для проверки того, что анализируемая выборка является последовательностью независимых и одинаково распределенных (н.о.р.) наблюдений. Набор тестов включает в себя тесты перестановок 3.4.1, и хи-квадрат тесты 3.4.2. Если все тесты не отвергают утверждения о н.о.р. (т.е. выборка проходит тестирование), то оценка энтропии выполняется согласно 3.5.1 в предположении, что данные являются н.о.р. Если хотя бы один из тестов отвергает гипотезу о н.о.р., то оценка энтропии выполняется согласно 3.5.2 для данных, не являющихся н.о.р.

3.4.1 Тесты перестановок для проверки независимости и одинаковой распределенности

Если верна нулевая гипотеза о том, что наблюдения в выборке являются н.о.р., то после любой перестановки наблюдений в выборке они остаются независимыми и сохраняют свое распределение. Соответственно, новая выборка, полученная после перестановки наблюдений, должна иметь те же статистические свойства, что и исходная выборка. При этом статистики, вычисленные для исходной выборки и для новой выборки, будут иметь одинаковое распределение вероятностей. Однако, если нулевая гипотеза не выполняется (т.е. наблюдения не являются н.о.р.), то некоторые статистики могут значимо различаться для исходной и новой выборок.

Перед применением тестов перестановок выборка наблюдений разбивается на десять непересекающихся подвыборок одинакового размера. Например, для выборки длины N каждая подвыборка будет иметь длину $\left\lfloor \frac{N}{10} \right\rfloor$. Тесты перестановок применяются отдельно для каждой из подвыборок, проверяя н.о.р. ее наблюдений.

В основе тестов перестановок лежит вычисление ряда статистических оценок. Предполагается, что относительно малые или относительно большие значения оценок будут свидетельствовать об отклонении выборки от ожидаемой

модели н.о.р. случайных величин. В тестах перестановок предлагается использовать следующий набор оценок:

- оценка сжатия 3.4.1.1;
- оценки серий 3.4.1.2;
- оценка максимального кумулятивного отклонения 3.4.1.3;
- оценки направленных серий 3.4.1.4;
- оценка ковариации 3.4.1.5;
- оценки коллизий 3.4.1.6.

Схема тестов перестановок описывается следующим образом.

1. Для каждой из 10 подвыборок производятся следующие действия.
   (a) Вычисляются все оценки для исходной подвыборки, обозначим их $S_{0,j}^{(k)}$, $1 \le j \le J$ — номер оценки, J — общее число различных оценок, k – номер подвыборки.
   (b) Для i от 1 до 1000 повторяются следующие шаги:
      - осуществляется перестановка наблюдений подвыборки, используя алгоритм Фишера-Йейтса [1];
      - вычисляются все оценки $S_{i,j}^{(k)}$ для новой подвыборки, полученной после перестановки.
2. Для каждой подвыборки k и для каждого вида оценки j формируется массив $\Sigma_j^{(k)} = \left\{ S_{0,j}^{(k)},\ S_{1,j}^{(k)},\ \ldots,\ S_{1000,j}^{(k)} \right\}$ из 1001 значения оценок и определяется ранг $R_j^{(k)}$, $0 \le R_j^{(k)} \le 1000$, согласно номеру оценки $S_{0,j}^{(k)}$ в отсортированном массиве $\Sigma_j^{(k)}$.

   Например, если оценка $S_{0,j}^{(k)}$ будет самой низкой ($S_{0,j}^{(k)} < S_{i,j}^{(k)}$,$1 \le i \le 1000$), то ее ранг будет равен 0. Напротив, если она будет самой высокой ($S_{0,j}^{(k)} > S_{i,j}^{(k)}$,$1 \le i \le 1000$), то ее ранг будет равен 1000.

   В случае совпадения оценки $S_{0,j}^{(k)}$ с одной или несколькими оценками из множества $\Sigma_j^{(k)}$, то в качестве ранга $R_j^{(k)}$ выбирается значение, наиболее близкое к 500.
3. Всего будет получено $10 \times J$ значений рангов. Вероятность, что значение ранга меньше 50 или больше 950, составляет 10%. Для каждого вида оценки j, $1 \le j \le J$, во множестве рангов $\left\{ R_j^{(1)}, R_j^{(2)}, \ldots, R_j^{(10)} \right\}$ определяется число рангов, меньших 50 или больших 950.

Если для какой-то оценки j число таких рангов будет равно восьми или более, то выборка не проходит проверку и полагается, что ее наблюдения не являются н.о.р.

3.4.1.1 Оценка сжатия

Алгоритмы сжатия предназначены для устранения избыточности в строке символов, связанной с частым повторением некоторых последовательностей символов. Оценка сжатия выборки данных является мерой качества сжатия и определяется как длина сжатой последовательности. Если наблюдения исходной выборки не являются н.о.р., то в ней могут присутствовать часто повторяющиеся шаблоны (группы наблюдений), которые должны исчезнуть после перестановки наблюдений. Соответственно, оценка сжатия может отличаться для исходной выборки и выборки, полученной после перестановки наблюдений.

Оценка сжатия строится следующим образом.

1. Наблюдения из выборки данных кодируются в виде символьной строки, содержащей список значений, разделенных запятыми: например, «144,21,139,0,0,15».

2. Символьная строка обрабатывается алгоритмом сжатия BZ2 [10].

3. Оценка сжатия будет равна длине сжатой строки в байтах.

3.4.1.2 Оценки серий

Если наблюдения выборки н.о.р., то максимальная длина серии наблюдений, больших или меньших медианы, должна быть не слишком большой и не слишком малой. Аналогично, число серий наблюдений, больших или меньших медианы, также не должно быть слишком большим или слишком малым. Слишком большое или слишком малое значение максимальной длины серий или числа серий свидетельствует о наличии зависимости наблюдений выборки.

Оценки серий строятся следующим образом.

1. Вычисляется медиана выборки. Если выборка бинарная, то медиана равна 0,5.

2. Строится вспомогательная подвыборка на основе наблюдений исследуемой выборки по следующим правилам:

   (a) если наблюдение больше медианы, то во вспомогательную подвыборку добавляется значение «+1»;

(b) если наблюдение меньше медианы, то во вспомогательную подвыборку добавляется значение «–1»;

(c) если наблюдение совпадает с медианой, то во вспомогательную подвыборку ничего не добавляется, т.е. значения, равные медиане, игнорируются.

3. Определяется наиболее длинная серия, состоящая только из «+1» или только из «–1» во вспомогательной подвыборке. Длина этой серии является первой оценкой.

4. Второй оценкой будет число серий, состоящих только из «+1» или только из «–1».

Пример 3.1. Предположим, что исходная выборка состоит из семи наблюдений {5, 15, 12, 1, 13, 9, 4}. Медиана данной выборки равна 9.

Будет построена следующая вспомогательная подвыборка: {–1, +1, +1, –1, +1, –1}.

Во вспомогательной подвыборке можно выделить следующие серии: (–1), (+1, +1), (–1), (+1) и (–1).

Первая оценка, максимальная длина серии, равна 2. Вторая оценка, общее число серий, равна 5.

3.4.1.3 Оценка максимального кумулятивного отклонения

Оценка кумулятивного отклонения позволяет определить, образуются ли в выборке кластеры наблюдений с относительно большими или малыми значениями. Если наблюдения выборки н.о.р., то кумулятивное отклонение не должно быть слишком большим или слишком малым. Малое значение оценки кумулятивного отклонения показывает наличие некоторого процесса, предотвращающего даже случайное появление кластеров наблюдений с относительно большими или малыми значениями. Большое значение оценки кумулятивного отклонения показывает слишком частое появление таких кластеров.

Оценка максимального кумулятивного отклонения является мерой отклонения кумулятивной суммы значений наблюдений от ее математического ожидания. Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots s_N\}$ и среднее значение наблюдений равно μ, тогда кумулятивное отклонение в точке i будет равно $d_i = s_1 + s_2 + \ldots + s_i - i\mu$, $1 \leq i \leq N$. Оценкой является максимальное значение кумулятивного отклонения d_i.

Оценка максимального кумулятивного отклонения строится следующим образом.

1. Вычисляется выборочное среднее μ.

2. Для $i = 1, 2, \ldots, N$ вычисляются величины кумулятивных отклонений:

$$d_i = \left| \sum_{j=1}^{i} s_j - i\mu \right|.$$

3. Оценка максимального кумулятивного отклонения будет равна $d_{\max} = \max\limits_i d_i$.

Пример 3.2. Пусть выборка состоит из пяти наблюдений $\{2, 15, 4, 10, 9\}$ со средним значением $\mu = 8$. Кумулятивные отклонения имеют следующие значения:

$$\begin{aligned} d_1 &= |2 - 8| = 6; \\ d_2 &= |(2 + 15) - 2 \cdot 8| = 1; \\ d_3 &= |(2 + 15 + 4) - 3 \cdot 8| = 3; \\ d_4 &= |(2 + 15 + 4 + 10) - 4 \cdot 8| = 1; \\ d_5 &= |(2 + 15 + 4 + 10 + 9) - 5 \cdot 8| = 0. \end{aligned}$$

Таким образом, оценка максимального кумулятивного отклонения $d_{\max} = 6$.

3.4.1.4 Оценки направленных серий

Для построения оценок направленных серий исследуются возрастающие и убывающие серии наблюдений. Слишком большое или малое значение числа серий и максимальной длины серии возрастающих (убывающих) наблюдений свидетельствует о наличии зависимости наблюдений выборки.

Пусть выборка состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$. Если выборка является бинарной, то необходима предварительная обработка, в результате которой биты объединяются в байты и в качестве новых наблюдений выступают веса Хэмминга — число единиц в байте.

Оценки направленных серий строятся следующим образом.

1. По исходной выборке строится вспомогательная выборка по следующим правилам:
   (a) последовательно рассматриваются пары (s_1, s_2), (s_2, s_3), ... , (s_{N-1}, s_N);
   (b) если в паре (s_i, s_{i+1}) значение первого наблюдения больше второго, то к вспомогательной выборке добавляется элемент «–1»;
   (c) если в паре (s_i, s_{i+1}) значение первого наблюдения меньше второго, то к вспомогательной выборке добавляется элемент «+1»;

(d) если в паре (s_i, s_{i+1}) значения наблюдений равны, то к вспомогательной выборке добавляется элемент «0».

2. Из вспомогательной выборки удаляются начальные нули.

   Например, вспомогательная выборка $\{0, 0, -1, 0, 0, 1\}$ превратится в $\{-1, 0, 0, 1\}$, а выборка $\{1, 0, 0, -1, 0\}$ по результатам данной операции не изменится.

3. Первой оценкой является общее число серий из элементов «+1» («–1»), при этом элемент «0» не прерывает серию.

   Например, серия (+1, 0, +1, +1, 0, 0, +1, ...) будет являться серией из элементов «+1».

4. Второй оценкой является максимальная длина серии из элементов «+1» («–1»), при этом элемент «0» не прерывает серию.

5. Подсчитывается число элементов «+1» и число элементов «–1». Третьей оценкой является максимальное из этих значений.

Пример 3.3. Пусть выборка состоит из наблюдений $\{2, 2, 2, 5, 7, 7, 9, 3, 1, 4, 4\}$. Вспомогательная выборка будет иметь вид $\{0, 0, +1, +1, 0, +1, -1, -1, +1, 0\}$. Можно выделить следующие серии, отбрасывая начальные нули из вспомогательной выборки: (+1, +1, 0, +1), (–1, –1) и (+1, 0).

Первая оценка, число серий, будет равна 3. Вторая оценка, максимальная длина серии, будет равна 4. Число элементов «+1» равно 4, число элементов «–1» равно 2. Третья оценка, максимум из этих значений, будет равна 4.

Пример 3.4. Рассмотрим бинарную выборку, которая преобразуется в следующую последовательность байтов, записанных в шестнадцатеричном виде: $\{A3, 57, 3F, 42, BD\}$. По весам Хэмминга $\{4, 5, 6, 2, 6\}$ строим вспомогательную выборку $\{+1, +1, -1, +1\}$, в которой можно выделить следующие серии: (+1, +1), (–1) и (+1).

Первая оценка, число серий, будет равна 3. Вторая оценка, максимальная длина серии, будет равна 2. Число элементов «+1» равно 3, число элементов «–1» равно 1. Третья оценка, максимум из этих значений, будет равна 3.

3.4.1.5 Оценка ковариации

Хи-квадрат тест независимости (из раздела 3.4.2.1) и оценка сжатия (из подраздела 3.4.1.1) являются эффективными при обнаружении повторяющихся шаблонов конкретных значений (например, строк значений наблюдений, которые появляются чаще, чем можно было бы ожидать, если бы наблюдения в выборке были бы н.о.р.), но не будут, в общем, обнаруживать

связи между числовыми значениями последовательных наблюдений (например, низкие значения наблюдений, как правило, следующие за высокими). Оценка ковариации будет выявлять такие отношения.

Оценка ковариации является мерой связи между соседними наблюдениями. При наличии линейной зависимости между соседними наблюдениями ковариация для исходной выборки должна быть по модулю больше, чем ковариация новой выборки, полученной перестановкой наблюдений.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$, и среднее значение равно μ. Оценка ковариации вычисляется по следующей формуле:

$$\frac{1}{N-1}\sum_{i=1}^{N-1}(s_i - \mu)(s_{i+1} - \mu).$$

Пример 3.5. Пусть выборка состоит из пяти наблюдений $\{15, 2, 6, 10, 12\}$ со средним значением $\mu = 9$.

Оценка ковариации равна

$$\frac{1}{4}\left[(15-9)(2-9) + (2-9)(6-9) + (6-9)(10-9) + (10-9)(12-9)\right] = \\ = \frac{1}{4}\left[-42 + 21 - 3 + 3\right] = -\frac{21}{4}.$$

3.4.1.6 Оценки коллизий

Оценки коллизий являются мерой числа наблюдений до появления повторного значения (коллизии). Предполагается, что если в выборке вероятности значений меняются со временем, то в исходной выборке потребуется меньше наблюдений до появления коллизии, чем в ее перестановках.

Пусть выборка состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$. Если выборка является бинарной, то необходима предварительная обработка, в результате которой биты объединяются в байты.

Оценки коллизий строятся следующим образом.

1. Для подсчета числа наблюдений до коллизии используется список C, который инициализируется пустым множеством.

2. Устанавливается текущая позиция $p = 1$.

3. Пока $p < N$, выполнять следующие действия:

   (a) найти минимальное i, что последовательность $s_p, s_{p+1}, \ldots, s_{p+i}$ содержит ровно одно повторяющееся наблюдение; если такого i не существует, то перейти к шагу 4;

   (b) добавить i в список C;

(c) сдвинуть текущую позицию $p \leftarrow p + i + 1$.

4. Оценками коллизий будут являться минимальное значение, среднее всех значений и максимальное значение в списке C.

Пример 3.6. Пусть выборка состоит из наблюдений $\{2, 1, 1, 2, 0, 1, 0, 1, 1, 2\}$. Если первое наблюдение «2» находится на позиции 0, то первая коллизия наблюдается при $i = 2$. Текущая позиция сдвигается на 3 наблюдения, после чего непросмотренная часть выборки имеет вид $\{2, 0, 1, 0, 1, 1, 2\}$. Следующая коллизия наблюдается при $i = 3$. Текущая позиция сдвигается на 4 наблюдения, непросмотренная часть выборки имеет вид $\{1, 1, 2\}$. Следующая коллизия наблюдается при $i = 1$. Текущая позиция сдвигается на 2 наблюдения, оставшаяся часть выборки имеет вид $\{2\}$, и больше невозможно обнаружить коллизии.

Построен список $C = \{2, 3, 1\}$. Первая оценка, минимальное значение в списке, будет равна 1. Вторая оценка, среднее всех значений в списке, будет равна 2. Третья оценка, максимальное значение в списке, будет равна 3.

3.4.2 Хи-квадрат тесты для проверки независимости и одинаковой распределенности

Если наблюдения выборки н.о.р., то ее можно рассматривать как выборку независимых величин с мультиномиальным распределением. Для проверки соответствия выборки наблюдений мультиномиальному распределению используется критерий хи-квадрат.

Если множество значений наблюдений слишком велико, то вероятность отдельных значений становится низкой, следовательно, для применения хи-квадрат теста требуются большие объемы данных. Если необходимое число данных собрать невозможно, то хи-квадрат тест не выполняется.

Используется два различных типа хи-квадрат критериев: критерий независимости и критерий согласия. Хи-квадрат критерий независимости предназначен для обнаружения зависимости в вероятностях между соседними наблюдениями и описан в 3.4.2.1 для небинарных данных и в 3.4.2.3 для бинарных данных. Критерий согласия хи-квадрат предназначен для проверки согласия распределения подвыборок, сформированных из исходной выборки и описан в 3.4.2.2 для небинарных данных и в 3.4.2.4 для бинарных данных.

3.4.2.1 Проверка независимости для небинарных данных

В данной проверке на первом шаге по выборке оцениваются вероятности всех возможных значений наблюдений. На втором шаге по выборке оцениваются вероятности соседних пар значений и проверяется согласие распределения вероятностей пар значений с теоретическим.

Пусть $p(x_i)$ — оценка вероятности того, что значение наблюдения равно x_i, $1 \leq i \leq n$, n — число различных значений наблюдений. Обозначим $E(x_i, x_j)$ ожидаемое число появлений пары (x_i, x_j) в анализируемой выборке. Пусть список *List1* содержит все пары (x_i, x_j), для которых $E(x_i, x_j) \geq 5$, и *List2* содержит все остальные пары (x_i, x_j). Обозначим E_0 ожидаемое число появления в выборке всех пар из списка *List2*.

Проверка на независимость для небинарных данных выполняется следующим образом.

1. По выборке оцениваются вероятности $p(x_i)$ всех возможных значений наблюдений x_i по следующей формуле:

$$p(x_i) = \frac{n(x_i)}{N},$$

   где $n(x_i)$ определяет, сколько раз в выборке встретилось наблюдение x_i, а N — длина выборки.

2. Определяется максимальная вероятность $p_{\max} = \max\limits_i p(x_i)$.

3. Подсчитывается число параметров $q = 1 + \sum\limits_i I(p(x_i)p_{\max} \geq 5/N)$, где $I(\cdot)$ — индикаторная функция.

   Если $q = 1$, то объем выборки слишком мал и алгоритм прекращает работу. Результат проверки в этом случае — «недостаточно данных».

4. Списки *List1* и *List2* инициализируются пустыми множествами, и устанавливается начальное значение частоты $E_0 = 0$.

5. Для каждой возможной пары значений (x_i, x_j), включая (x_i, x_i), выполняются следующие операции:

   (a) вычисляется значение $E(x_i, x_j) = p(x_i)p(x_j)(N-1)$;

   (b) если $E(x_i, x_j) \geq 5$, то пара (x_i, x_j) добавляется в список *List1*, иначе пара (x_i, x_j) добавляется в список *List2* с пересчетом значения $E_0 \leftarrow E_0 + E(x_i, x_j)$;

6. Проверяется наличие достаточного числа степеней свободы. Обозначим w число пар в списке *List1*. Если $w+1-q < 1$, то объем выборки слишком мал и алгоритм прекращает работу. Результат проверки в этом случае — «недостаточно данных».

7. Вычисляется хи-квадрат статистика χ^2. Обозначим $n(x_i, x_j)$ наблюдаемую частоту пары (x_i, x_j) в выборке. Выполняются следующие вычисления:

(a) инициализируется значение статистики $\chi^2 = 0$;

(b) для каждой пары (x_i, x_j) из списка *List1* пересчитывается значение статистики χ^2 следующим образом:

$$\chi^2 \leftarrow \chi^2 + \frac{(E(x_i, x_j) - n(x_i, x_j))^2}{E(x_i, x_j)};$$

(c) инициализируется наблюдаемая частота пар из списка *List2* $n_0 = 0$;

(d) для каждой пары (x_i, x_j) из списка *List2* пересчитывается наблюдаемая частота:

$$n_0 \leftarrow n_0 + n(x_i, x_j);$$

(e) пересчитывается значение статистики χ^2 следующим образом:

$$\chi^2 \leftarrow \chi^2 + \frac{(E_0 - n_0)^2}{E_0}.$$

8. Статистика χ^2 сравнивается с пороговым значением $\chi^2_{w+1-q}(\alpha)$ распределения хи-квадрат с $w + 1 - q$ степенями свободы и уровнем значимости $\alpha = 0,001$.

   Если $\chi^2 > \chi^2_{w+1-q}(\alpha)$, то выборка не проходит проверку на независимость наблюдений.

3.4.2.2 Проверка одинаковой распределенности для небинарных данных

В данной проверке анализируемая выборка разбивается на подвыборки, а затем проверяется гипотеза о согласии распределения каждой подвыборки с распределением исходной выборки. Если гипотеза о согласии отклоняется, то это свидетельствует о неоднородности выборки.

Проверка одинаковой распределенности для небинарных данных осуществляется следующим образом.

1. Исходная выборка длины N разбивается на 10 подвыборок длины $\lfloor N/10 \rfloor$.

2. Для каждого возможного значения наблюдения x_i, $1 \leq i \leq n$, где n — число различных значений наблюдений, по исходной выборке оценивается ожидаемая частота $E(x_i)$ его появления в подвыборке:

$$E(x_i) = \frac{n(x_i)}{N} \cdot \left\lfloor \frac{N}{10} \right\rfloor,$$

   где $n(x_i)$ определяет, сколько раз в исходной выборке встретилось наблюдение x_i.

3. Создается список *List3* всех возможных значений наблюдений x_i таких, что $E(x_i) \geq 5$.

4. Создается список *List4* всех возможных значений наблюдений x_i таких, что $E(x_i) < 5$.

5. Вычисляется ожидаемая суммарная частота E_0 встречаемости всех значений наблюдений x_i из списка *List4*, как сумма соответствующих значений частот $E(x_i)$.

6. Для каждой из десяти подвыборок ($1 \leq j \leq 10$) повторяются следующие операции:

   (a) устанавливается начальное значение статистики $\chi_j^2 = 0$;

   (b) для каждого значения x_i из списка *List3* пересчитывается значение статистики χ_j^2:

   $$\chi_j^2 \leftarrow \chi_j^2 + \frac{(E(x_i) - n_j(x_i))^2}{E(x_i)},$$

   где $n_j(x_i)$ определяет, сколько раз в j-ой подвыборке встретилось наблюдение x_i;

   (c) определяется величина $n_{0,j}$, сколько раз в j-ой подвыборке встретились наблюдения x_i из списка *List4*, как сумма соответствующих наблюдаемых значений $n_j(x_i)$;

   (d) пересчитывается значение статистики χ_j^2:

   $$\chi_j^2 \leftarrow \chi_j^2 + \frac{(E_0 - n_0)^2}{E_0}.$$

7. Статистики χ_j^2, $1 \leq j \leq 10$, сравниваются с пороговым значением $\chi_L^2(\alpha)$ распределения хи-квадрат с L степенями свободы, где L — размер списка $List3$, и уровнем значимости $\alpha = 0,001$.

   Если $\chi_j^2 > \chi_L^2(\alpha)$ для некоторой подвыборки j, $1 \leq j \leq 10$, это означает, что распределение данной подвыборки не согласуется с распределением всей выборки. В этом случае анализируемая выборка не проходит проверку на одинаковую распределенность наблюдений.

3.4.2.3 Проверка независимости для бинарных данных

Особенностью бинарных данных является то, что наблюдения принимают лишь два различных значения, кодируемых битом 0 или 1. Если наблюдения бинарной выборки н.о.р., то распределения любых двух битов выборки будет одинаковыми, следовательно, вероятность любой k-битной строки будет равна произведению вероятностей значений составляющих ее битов.

Данный тест проверяет, согласуется ли распределение k-битных строк с ожидаемым распределением. Если расположенные рядом биты не являются независимыми, то вероятности k-битных строк не будут равны произведению вероятностей составляющих их битов и гипотеза о согласии распределений будет отвергнута.

Проверка независимости для бинарных данных осуществляется следующим образом.

1. Определяются следующие величины: C_0 — число нулевых битов в анализируемой выборке, C_1 — число единичных битов, $C_x = \min(C_0, C_1)$, $p = C_1/N$ — частота единичных битов, N — общее число битов в выборке.

2. Для k от 2 до 11, пока выполняется условие $(C_x/N)^k > 5/N$, осуществляются следующие действия:

   (a) строится модифицированная выборка, объединяя последовательно каждые k битов в одно наблюдение со значением v, $0 \leq v \leq 2^k - 1$, при этом вероятность значения v составит

   $$p_v = \frac{N}{k} p^{W(v)} (1-p)^{k-W(v)},$$

   где вес Хэмминга $W(v)$ определяет число единичных битов в v;

   (b) вычисляется значение статистики χ^2:

   $$\chi^2 = \sum_{v=0}^{2^k-1} \frac{(n_v - p_v)^2}{p_v},$$

   где n_v — число наблюдений со значением v в модифицированной выборке.

3. Статистика χ^2 сравнивается с пороговым значением $\chi^2_{2^k-1}(\alpha)$ распределения хи-квадрат с $2^k - 1$ степенями свободы, и уровнем значимости $\alpha = 0,001$.

   Если $\chi^2 > \chi^2_{2^k-1}(\alpha)$, то выборка не проходит проверку на независимость.

3.4.2.4 Проверка одинаковой распределенности для бинарных данных

В данной проверке анализируемая выборка разбивается на подвыборки, а затем исследуется гипотеза о согласии распределений каждой подвыборки с распределением исходной выборки. Если гипотеза о согласии отклоняется, это свидетельствует о неоднородности выборки.

Проверка одинаковой распределенности для бинарных данных осуществляется следующим образом.

1. Подсчитывается число единичных битов в исходной выборке, равное n, и оценивается вероятность появления единичного бита $p = n/N$, где N — общее число битов в выборке.

2. Исходная выборка длины N разбивается на 10 подвыборок длины $N' = \lfloor N/10 \rfloor$.

3. Вычисляется значение статистики χ^2:

$$\chi^2 = \sum_{i=1}^{10} \frac{(n_i - pN')^2}{pN'},$$

   где i определяет номер подвыборки, $1 \leq i \leq 10$, а n_i — число единиц в i-ой подвыборке.

4. Статистика χ^2 сравнивается с пороговым значением $\chi^2_9(\alpha)$ распределения хи-квадрат с 9 степенями свободы, и уровнем значимости $\alpha = 0,001$.

   Если $\chi^2 > \chi^2_9(\alpha) = 27.9$, то выборка не проходит проверку на одинаковую распределенность.

3.5 Оценка энтропии выходной последовательности

Оценка энтропии для последовательности н.о.р. наблюдений осуществляется согласно 3.5.1. Оценка энтропии для последовательности наблюдений, не являющихся н.о.р., производится согласно 3.5.2.

3.5.1 Оценка минимальной энтропии последовательности независимых и одинаково распределённых случайных величин

Оценка минимальной энтропии последовательности н.о.р. случайных величин производится на основе частоты наиболее встречаемого значения. Чтобы не переоценить значение энтропии, используется верхняя граница 99%-ого доверительного интервала в качестве оценки вероятности наиболее встречаемого значения.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$. Алгоритм вычисления минимальной энтропии состоит из следующих шагов.

1. Определяется наиболее встречаемое в выборке значение. Пусть оно встретилось n_{max} раз.

2. Строится верхняя граница 95%-ого доверительного интервала для вероятности наиболее встречаемого значения:

$$p_{max} = p + 2.3\sqrt{\frac{p(1-p)}{N}},$$

где $p = n_{max}/N$ — оценка вероятности наиболее встречаемого значения.

3. В качестве оценки энтропии полагается следующая величина:

$$H = -\log_2(p_{max}).$$

4. Пусть для представления каждого наблюдения выборки используется W бит. Тогда окончательная оценка минимальной энтропии определяется следующим образом:

$$H_{min} = \min(W, H).$$

Пример 3.7. Пусть выборка длины $N = 20$ состоит из наблюдений {0, 1, 1, 2, 0, 1, 2, 2, 0, 1, 0, 1, 1, 0, 2, 2, 1, 0, 2, 1}, при этом для представления каждого наблюдения используется $W = 3$ бит.

Определим частоты всех значений: $n_0 = 6$, $n_1 = 8$, $n_2 = 6$. Таким образом, самым встречаемым значением будет 1 и $n_{max} = 8$. Оценка его вероятности будет равна $p = 8/20 = 0.4$. Верхняя граница 95%-ого доверительного интервала равна

$$p_{max} = 0.4 + 2.3\sqrt{\frac{0.4 \cdot (1-0.4)}{20}} = 0.6520.$$

Оценка энтропии равна

$$H = -\log_2(p_max) = -\log_2(0.6520) = 0.6172.$$

Окончательная оценка минимальной энтропии равна

$$H_{min} = \min(W, H) = \min(3, 0.6172) = 0.6172.$$

3.5.2 Оценка минимальной энтропии последовательности случайных величин, не являющихся независимыми и одинаково распределёнными

Для оценки энтропии последовательности случайных величин, не являющихся н.о.р, используется пять статистических тестов 3.5.2.1 – 3.5.2.5. Во всех тестах используется уровень значимости $\alpha = 0.95$. В качестве оценки минимальной энтропии используется минимальная из всех оценок, полученных каждым тестом.

Если для каких-то из представленных здесь тестов недостаточно данных и нет возможности получить для проверки более длинную выходную последовательность, то данные тесты пропускаются и их результаты не учитываются при определении оценки минимальной энтропии.

3.5.2.1 Тест коллизий

В тесте коллизий измеряется среднее время до первого совпадения (коллизии) наблюдений. Данный тест даёт низкую оценку энтропии для источников случайности, которые имеют значительное смещение частоты нескольких значений от остальных, что приводит к более быстрому появлению коллизий.

Данный тест находит нижнюю границу энтропии с заданным уровнем доверия в том случае, когда наблюдения независимы. Наличие зависимостей между наблюдениями может привести к завышенной оценке энтропии по результатам данного теста.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$.

Алгоритм теста коллизий для оценки энтропии состоит из следующих шагов.

1. Инициализируются счётчик числа коллизий $\nu = 0$ и последовательность моментов коллизий $T = \{t_i\}$, $t_0 = 0$.

2. Производится поиск коллизии — такого минимального натурального j, что $s_j = s_i$ для некоторого i, $t_\nu < i < j$.

   Если коллизия найдена, то осуществляется переход на шаг 3, иначе — на шаг 5.

3. Увеличивается счётчик числа коллизий: $\nu \leftarrow \nu + 1$.

4. В последовательность T добавляется момент коллизии $t_\nu = j$. Переход на шаг 2.

5. Если $\nu < 1000$, то для теста коллизий недостаточно данных и алгоритм завершает свою работу.

6. Время до коллизии определяется разностью соседних значений $t_i - t_{i-1}$ моментов коллизий, $1 \leq i \leq \nu$.

   Вычисляется выборочное среднее μ и выборочное стандартное отклонение σ времени до коллизии:

$$\mu = \frac{1}{\nu} \sum_{i=1}^{\nu} (t_i - t_{i-1}),$$

$$\sigma = \sqrt{\frac{1}{\nu} \sum_{i=1}^{\nu} (t_i - t_{t-1})^2 - \mu^2}.$$

7. Вычисляется нижняя граница 95%-ого доверительного интервала для среднего:

$$\bar{\mu} = \mu - 1.96 \frac{\sigma}{\sqrt{\nu}}.$$

8. Определяется однопараметрическое семейство распределений с параметром p и функцией вероятности $P_p(i)$, заданной следующим образом:

$$P_p(i) = \begin{cases} p & i = 0, \\ \dfrac{1-p}{n-1}, & \text{иначе}, \end{cases}$$

где n — число различных возможных значений наблюдения.

9. Пусть случайная величина S определяется функцией вероятности $P_p(i)$. Используя метод бинарного поиска, вычисляется такое значение параметра p, чтобы математическое ожидание $E\{S\}$ случайной величины S было равно $\bar{\mu}$.

   Математическое ожидание $E\{S\}$ определяется следующим образом:

$$E\{S\} = pq^{-2}\left(1 + \frac{p^{-1} - q^{-1}}{n}\right) F(q) - \frac{pq^{-1}(p^{-1} - q^{-1})}{n},$$

   где $q = (1-p)/(n-1)$, $F(1/z) = \Gamma(n+1, z)z^{-n-1}e^{-z}$, $\Gamma(\cdot,\cdot)$ — неполная гамма-функция.

10. Оценка минимальной энтропии равна

$$H_{min} = -\log_2(p).$$

Замечание 3.1. При применении теста коллизий для оценки минимальной энтропии в исследуемой выборке должно наблюдаться как минимум 1000 коллизий. Число коллизий зависит от размера выборки и мощности множества выходных значений источника случайности. Если мощность множества выходных значений достаточно велика, это может накладывать невыполнимые на практике требования на длину выборки.

3.5.2.2 Тест частичных коллекций

В тесте частичных (неполных) коллекций оценка энтропии вычисляется на основе количества различных значений в выборке. Для источников случайности с небольшим числом различных выходных значений оценка энтропии будет низкой, для источников случайности с отсутствием или малым числом повторов выходных значений будет построена высокая оценка энтропии. Тест частичных коллекций в отличие от теста полных коллекций всегда заканчивается за конечное время.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$, и пусть n — мощность множества выходных значений источника случайности.

Алгоритм теста частичных коллекций для оценки энтропии состоит из следующих шагов.

1. Выборка $\{s_1, s_2, \ldots, s_N\}$ разбивается на $\nu = [N/n]$ непересекающихся блоков длины n. Если последний блок неполный, то он не используется в данном тесте.

   Если $\nu < 500$, то для теста частичных коллекций недостаточно данных и алгоритм прекращает работу.

2. В каждом блоке i определяется число различных значений наблюдений, которое обозначается t_i, $1 \leq i \leq \nu$.

3. Вычисляется выборочное среднее μ и выборочное стандартное отклонение σ для значений t_i, $1 \leq i \leq \nu$:

$$\mu = \frac{1}{\nu} \sum_{i=1}^{\nu} t_i,$$

$$\sigma = \sqrt{\frac{1}{\nu} \sum_{i=1}^{\nu} t_i^2 - \mu^2}.$$

4. Определяется нижняя граница 95%-ого доверительного интервала для среднего:

$$\bar{\mu} = \mu - 1.96 \frac{\sigma}{\sqrt{\nu}}.$$

5. Определяется однопараметрическое семейство распределений с параметром p и функцией вероятности $P_p(i)$, заданной следующим образом:

$$P_p(i) = \begin{cases} p & i = 0, \\ \dfrac{1-p}{n-1}, & \text{иначе.} \end{cases}$$

6. Пусть случайная величина S определяется функцией вероятности $P_p(i)$. Используя метод бинарного поиска, вычисляется такое значение параметра p, чтобы математическое ожидание $E\{S\}$ случайной величины S было равно $\bar{\mu}$.

   Математическое ожидание $E\{S\}$ определяется следующим образом:

$$E\{S\} = 1 - (1-p)^n + (n-1)(1 - (1-q)^n),$$

   где $q = (1-p)/(n-1)$.

7. Оценка минимальной энтропии равна:

$$H_{min} = -\log_2(p).$$

Замечание 3.2. С помощью теста частичных коллекций точная оценка энтропии может быть получена только в том случае, если точно известно множество выходных значений источника случайности.

Например, пусть в источнике случайности с четырёхбитным выходом только два бита находятся под влиянием источника случайности, тогда вместо $2^4 = 16$ возможных значений будет наблюдаться всего $2^2 = 4$ различных значений. Поиск всех 16 вариантов будет безрезультатным, и оценка энтропии в этом случае будет завышена.

Следовательно, разработчик должен предоставить полную информацию о множестве выходных значений источника случайности с указанием тех значений, которые никогда не появляются на выходе.

Замечание 3.3. Для применения теста частичных коллекций длина исследуемой выходной последовательности должна быть как минимум $500n$, где n — мощность множества выходных значений источника случайности.

Если мощность множества выходных значений достаточно велика, это может накладывать невыполнимые на практике требования на длину выборки.

3.5.2.3 Тест марковской зависимости

В тесте марковской зависимости строится оценка энтропии на основе зависимостей между соседними наблюдениями выходной последовательности. В качестве модели зависимости используется цепь Маркова первого порядка, в которой значение следующего наблюдения зависит только от значения текущего наблюдения.

По исследуемой выходной последовательности оценивается матрица вероятностей переходов и вектор вероятностей начальных состояний. При отсутствии данных для оценки каких-либо вероятностей, чтобы оценка энтропии не была завышенной, завышаются значения этих вероятностей.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$ и пусть n — размер множества выходных значений источника случайности. Таким образом, число различных состояний цепи Маркова будет равно n.

Алгоритм теста марковской зависимости для оценки энтропии состоит из следующих шагов.

1. Определяется уровень значимости $\alpha = \min(\alpha^{n^2}, \alpha^k)$, где $k = 128$ — предполагаемая длина цепи Маркова.

2. Оценивается вектор вероятностей начальных состояний $\pi = (\pi_1, \pi_2, \ldots, \pi_n)^T$:

$$\pi_i = \min\left\{1, \frac{n_i}{n} + \varepsilon\right\},\ 1 \le i \le n,$$

где n_i — число наблюдений со значением i в выборке, а величина ε определяется следующим образом:

$$\varepsilon = \sqrt{\frac{-\log_2(1-\alpha)}{2N}}.$$

3. Оценивается матрица вероятностей переходов $P = (p_{ij})_{n\times n}$:

$$p_{ij} = \begin{cases} 1, & n_i = 0, \\ \min\left\{1, \dfrac{n_{ij}}{n_i} + \varepsilon_i\right\}, & \text{иначе}, \end{cases}$$

где n_{ij} — наблюдаемое в выборке число переходов из состояния i в состояние j, $1 \le i, j \le n$, а величина ε_i определяется следующим образом:

$$\varepsilon_i = \sqrt{\frac{-\log_2(1-\alpha)}{2n_i}}.$$

4. Используя оценки вектора вероятностей начальных состояний π и матрицы вероятностей переходов P, определяется цепочка состояний длины $k = 128$ с максимальной вероятностью, значение которой обозначается $p_{\max}$:

$$p_{\max} = \max_{i_1 i_2 \ldots i_k} \left\{ \pi_{i_m} \prod_{j=2}^{k} p_{i_{j-1} i_j} \right\},$$

где $1 \le i_m \le n$, $1 \le m \le k$.

5. Оценка минимальной энтропии равна

$$H_{min} = -\log_2(p_{\max}).$$

.

Замечание 3.4. Для оценки матрицы вероятностей переходов необходимо как минимум $20n^2$ наблюдений, где n — мощность множества выходных значений источника случайности. При этом учитываются только достижимые на практике значения.

Если мощность множества выходных значений достаточно велика, это может накладывать невыполнимые на практике требования на длину выборки.

3.5.2.4 Тест сжатия

Тест сжатия основан на универсальном тесте Маурера и определяет оценку энтропии на основе меры того, насколько сильно может быть сжата выходная последовательность без потери информации.

Преимуществом теста Маурера является то, что для него не требуется независимость наблюдений. Для вычисления статистики Маурера требуется только один проход по выходной последовательности, что позволяет построить эффективный алгоритм.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$. Для выполнения теста сжатия данная выборка разбивается на две непересекающиеся части. Первая часть из $d = 1000$ наблюдений $\{s_1, s_2, \ldots, s_d\}$ используется в качестве словаря для обучения алгоритма сжатия. Вторая часть из $\nu = (N - d)$ наблюдений $\{s_{d+1}, s_{d+2}, \ldots, s_N\}$ используется для вычисления статистики Маурера.

Алгоритм теста сжатия для оценки энтропии состоит из следующих шагов.

1. На основании обучающей последовательности $\{s_1, s_2, \ldots, s_d\}$ строится словарь, состоящий из пар вида (s_i, i), где s_i определяет значение наблюдения, а i — его позицию. Для этого выполняется следующая последовательность действий:

   (a) словарь инициализируется пустым множеством;

   (b) инициализируется счётчик $i = 1$;

   (c) если значение s_i отсутствует в словаре, то в него добавляется пара (s_i, i);

   (d) если значение s_i присутствует в словаре, то обновляется только индекс, который устанавливается в значение i;

   (e) увеличивается значение счётчика $i \leftarrow i + 1$;

   (f) если $i \leq d$, то осуществляется переход к действию (c)), иначе — к шагу 2.

2. Для второй части последовательности $\{s_{d+1}, s_{d+2}, \ldots, s_N\}$ выполняются следующие действия:

   (a) устанавливается значение счётчика $i = d + 1$;

   (b) если значение s_i присутствует в словаре, то вычисляется значение A_i как разность индекса i и записанного в словаре индекса, после чего индекс обновляется значением i;

   (c) если значение s_i отсутствует в словаре, то в словарь добавляется пара (s_i, i);

   (d) увеличивается значение счётчика $i \leftarrow i + 1$;

   (e) если $i \leq N$, то осуществляется переход к действию (b)), иначе — к шагу 3.

3. Используя элементы последовательности A_i, $d+1 \leq i \leq N$, построенные на шаге 2, вычисляются выборочное среднее μ и среднеквадратическое отклонение σ для логарифмов:

$$\mu = \frac{1}{\nu} \sum_{i=d+1}^{N} \log_2 A_i,$$

$$\sigma = \sqrt{\frac{1}{\nu} \sum_{i=d+1}^{N} (log_2 A_i)^2 - \mu^2}.$$

4. Вычисляется нижняя граница 95%-ого доверительного интервала для среднего:

$$\bar{\mu} = \mu - 1.96 \frac{\sigma}{\sqrt{\nu}}.$$

5. Определяется однопараметрическое семейство распределений с параметром p и функцией вероятности $P_p(i)$, заданной следующим образом:

$$P_p(i) = \begin{cases} p & i = 0, \\ \dfrac{1-p}{n-1}, & \text{иначе}, \end{cases}$$

где n — мощность множества выходных значений источника случайности.

6. Пусть случайная величина S определяется функцией вероятности $P_p(i)$. Используя метод бинарного поиска, вычисляется такое значение параметра p, чтобы математическое ожидание $E\{S\}$ случайной величины S было равно $\bar{\mu}$.

Математическое ожидание $E\{S\}$ определяется следующим образом:

$$E\{S\} = G(p) + (n-1)G(q),$$

где

$$q = \frac{1-p}{n-1},$$
$$G(p_i) = \frac{1}{\nu} \sum_{t=d+1}^{N} \sum_{s=1}^{t} \log_2(s) P\left[A_t = s \cap X_t = i\right],$$
$$P\left[A_t = s \cap X_t = i\right] = \begin{cases} p_i^2 (1-p_i)^{s-1} & s < t, \\ p_i (1-p_i)^{t-1} & s = t. \end{cases}$$

7. Оценка минимальной энтропии равна:

$$H_{min} = -\log_2(p).$$

3.5.2.5 Частотный тест

В частотном тесте энтропия вычисляется на основе наиболее встречаемого значения выходной последовательности.

Пусть выборка длины N состоит из наблюдений $\{s_1, s_2, \ldots, s_N\}$.

Алгоритм частотного теста оценки энтропии состоит из следующих шагов.

1. Определяется наиболее встречаемое в выборке значение. Пусть оно встретилось $n_{\max}$ раз, тогда его вероятность можно оценить величиной $p = n_{\max}/N$.

2. Используя заданный уровень значимости $\alpha = 0.05$ вычисляется добавка

$$\varepsilon = \sqrt{\frac{-log_2(1-\alpha)}{2N}} = \frac{0.1924}{\sqrt{N}}.$$

3. Оценка энтропии равна

$$H_{min} = -\log_2(p_{\max} + \varepsilon).$$

ГЛАВА 4
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТЕСТИРОВАНИЯ ВЫХОДНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ИСТОЧНИКОВ СЛУЧАЙНОСТИ И ЕГО ПРИМЕНЕНИЕ

4.1 Описание пакета программ для тестирования выходных последовательностей источников случайности

Было разработано программное обеспечение тестирования выходных последовательностей источников случайностей, исходный код ключевых частей которого представлен в приложении. В нем реализованы тесты, описанные в главе 3. Пакет тестов реализован в виде консольного приложения. Формат вызова: `>EntropySourceTesting.exe FileName SampleSize`, где `FileName` – имя файла содержащего выходную последовательность в двоичном виде, `SampleSize` – количество битов представляющих одно наблюдение. Размер наблюдения технически ограничен 32 битами.

Пример 4.1. Вызов `>EntropySourceTesting.exe data.bin 8` означает, что последовательность будет считываться из файла `data.bin`, а размер одного наблюдения – 8 бит, т.е. наблюдения лежат в интервале $[0, 2^8 - 1]$.

Основные функции из состава ПО:

- Тесты перестановок для проверки независимости и одинаковой распределенности 3.4.1 представлены классами `ShufflingTest` и `StatisticalScores`, исходный код которых представлен в разделе А.1 приложения А. Непосредственно код теста представлен в А.1.1, а методы оценок представлены в подразделах А.1.2.1–А.1.2.6.

- Хи-квадрат тест для проверки независимости для небинарных данных 3.4.2.1 представлен в классе `ChiSquaredTestReport` методом `independenceTest(int[] dataset)`, исходный код которого представлен в приложении А в разделе А.2.1.

- Хи-квадрат тест для проверки одинаковой распределенности для небинарных данных 3.4.2.2 представлен в классе `ChiSquaredTestReport`

методом `goodnessOfFitTest(int[] dataset)`, исходный код которого представлен в приложении А в разделе А.2.2.

- Хи-квадрат тесты для проверки независимости 3.4.2.3 и одинаковой распределенности 3.4.2.4 для бинарных данных представлены в классе `ChiSquaredTestReport` методом `binaryTest(int[] dataset)`, исходный код которого представлен в приложении А в разделе А.2.3.

- Оценки минимальной энтропии из параграфа 3.5 представлены в классе `EntropyTests`:

  – Оценка минимальной энтропии последовательности случайных величин, не являющихся н.о.р. 3.5.1 представлена методом `iidTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.1.

  – Тест коллизий 3.5.2.1 представлен методом `collisionTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.2.

  – Тест частичных коллекций 3.5.2.2 представлен методом `partialCollectionTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.3.

  – Тест марковской зависимости 3.5.2.3 представлен методом `markovTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.4.

  – Тест сжатия 3.5.2.4 представлен методом `compressionTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.5.

  – Частотный тест 3.5.2.5 представлен методом `frequencyTest(int[] dataset, byte sampleSize)`, исходный код которого представлен в приложении А в разделе А.3.6.

4.2 Тестирование выходных последовательностей источников случайности

Будем подавать на вход разработанного ПО последовательности размером 1 Мб, полученные из кольцевого осциллятора (подробный вывод теста перестановок опустим в силу его объемности). Имена файлов с последовательностями `data1.bin` – `data5.bin`.

В кольцевом осцилляторе генерация случайных чисел происходит побитово, поэтому можем рассматривать последовательность как последовательность наблюдений размером 1 бит. Зададим соответствующий параметр на вход программы: `>EntropySourceTesting.exe data1.bin 1`. Пример подробного вывода результатов данной команды приведен в приложении Б. Более всего нас интересуют строки:

```
Н.о.р. тест ПРОЙДЕНО

Оценка энтропии для н.о.р. данных: 0,692203546584333
Минимальная оценка энтропии: 0,692203546584333
```

Это означает, что поданная на вход последовательность является н.о.р., а оценка минимальной энтропии равна 0,692, что в совокупности говорит нам о достаточно высоком качестве исследуемой последовательности.

Проведем подобное тестирование для всех доступных последовательностей. Все последовательности распознаются как н.о.р., оценки минимальной энтропии представим в таблице 4.1.

Таблица 4.1 — Оценки минимальной энтропии для различных последовательностей

Разм. набл.	data1.bin	data2.bin	data3.bin	data4.bin	data5.bin
1 бит	0,692203547	0,692249280	0,692167342	0,691975383	0,692205690
6 бит	4,130983400	4,128385326	4,125526329	4,120010266	4,119655437
8 бит	5,478302541	5,475286444	5,449223455	5,464916410	5,466981756
16 бит	9,718548948	9,718548948	9,718548948	9,642548930	9,718548948
32 бит	10,81791105	10,81791105	10,81791105	10,81791105	10,81791105

Дополнительно проведём тестирование с размерами наблюдений 6, 8, 16, 32 бита. Все последовательности были распознаны как н.о.р. Как видим, во всех случаях, кроме тестирования по 32-битным наблюдениям, оценка минимальной энтропии составляет более половины битовой длины наблюдения и относительно постоянна по всем последовательностям. Низкую оценку для размера наблюдений 32 бита можно объяснить недостаточной для точной оценки длиной последовательности (размерность выходного пространства значительно превышает длину последовательности).

Можно сделать вывод о что хорошем качестве исследуемого источника энтропии. Показана практическая применимость разработанного ПО.

Замечание 4.1. Стоит заметить, что процесс тестирования является достаточно ресурсоёмким (особенно тесты перестановок). Тестирование каждой из

приведённых последовательностей размером 1 Мб при разбиении на однобитовые наблюдения занимает около 60% ресурсов четырёхядерного процессора с тактовой частотой 3 ГГц и около 150 Мб оперативной памяти, при этом тестирование длится около часа.

ЗАКЛЮЧЕНИЕ

В работе получены следующие основные результаты:

1. Проведен аналитический обзор физических устройств и процессов, используемых для генерации случайных чисел.
2. Рассмотрены подходы к оценке качества выходных последовательностей источников случайности физических генераторов.
3. Разработаны алгоритмы и программное обеспечение для тестирования и оценки качества источников случайности.
4. С помощью разработанного ПО протестированы выходные последовательности реального источника.
5. Разработанное ПО может применяться для тестирования источников случайности на соответствие требованиям различных стандартов, в том числе СТБ 34.101.27-2011 «Требования безопасности к программным средствам криптографической защиты информации».

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования / Д. Кнут. – 3-е изд. Москва: Вильямс, 2011. — 832 с.

2. Математические и компьютерые основы криптологии: учеб. пособие / Ю. С. Харин [и др.]; под общ.ред. Ю. С. Харина. – Минск: Новое знание, 2003. – 382 с.

3. Информационные технологии и безопасность. Требования безопасности к программным средствам криптографической защиты информации: СТБ 34.101.27-2011 – Введ. 01.03.12. – Минск: Учреждение Белорусского государственного университета «Научно-исследовательский институт прикладных проблем математики и информатики», 2011. – 33 с.

4. Шумовые диоды // Музей электронных раритетов [Электронный ресурс]. –2013. – Режим доступа: http://www.155la3.ru/noise_diode.htm. – Дата доступа: 20.04.2016.

5. Barker, E. Recommendation for the entropy sources used for random bit generation: NIST special publication 800-90b / E. Barker, J. Kelsey. – Gaithersburg, MD: NIST, 2012. – 78 p.

6. Jun, B. The Intel® random number generator / B. Jun, P. Kocher. – San Francisco, CA: Cryptography Research, 1999. – 8 p.

7. Menezes, A. Handbook of applied cryptography / A. Menezes, P. van Oorschot, S. Vanstone. – 5th printing. – Boca Raton, FL: CRC Press, 2001. – 816 p.

8. Schneier, B. Security Pitfalls in Cryptography / B. Schneier // Schneier on Security [Electronic resource]. – 1998. – Mode of access: https://www.schneier.com/essays/archives/1998/01/security_pitfalls_in.html. – Date of access: 19.05.2016.

9. Shannon, C.E. A mathematical theory of communication / C. E. Shannon // The Bell System Technical Journal. – 1948. – Vol. 27. – P. 379-423

10. BZ2 compression algorithm [Electronic resource]. –Mode of access: http://www.bzip.org/. – Date of access: 17.12.2015.

11. Genuine Random Number Generator(GRANG) // LE Tech [Electronic resource]. – 2015. – Mode of access: http://www.letech.jpn.com/rng/about_rng_e.html. – Date of access: 08.05.2016.

12. HotBits: Genuine random numbers, generated by radioactive decay // Fourmilab [Electronic resource]. – 2006. – Mode of access: https://www.fourmilab.ch/hotbits/. – Date of access: 23.04.2016.

13. Lavarand // Wikipedia, the free encyclopedia [Electronic resource]. – 2015. – Mode of access: http://en.wikipedia.org/wiki/Lavarand. – Date of access: 19.05.2016.

14. True random number generation exploiting quantum physics // Random number generation from ID Quantique [Electronic resource]. – 2015. – Mode of access:http://www.idquantique.com/random-number-generation/. – Date of access: 15.05.2016.

15. RANDOM.ORG // RANDOM.ORG – True Random Number Service [Electronic resource]. – 2015. – Mode of access: https://www.random.org/faq/. – Date of access: 18.05.2016.

ПРИЛОЖЕНИЕ А

Исходный код основных частей разработанного приложения

A.1 Тесты перестановок

A.1.1 Основная логика теста

```
public struct ShufflingTestReport
{
    public string Name; //statistical score name
    public int[] P; //ranks
    public TestResult Result;
}

public static class ShufflingTest
{
    public const int subsetNumber = 10;
    public const int shuffleNumber = 1000;

    private struct RankRecord
    {
        public string Name;
        public int Subset;
        public int Rank;
    }

    private static Object shufLock = new Object();

    public static TestResult execute(int[] dataset, bool binary =
       false) {
        List<ShufflingTestReport> report;
        return execute(dataset, out report, binary);
    }

    public static TestResult execute(int[] dataset, out
       List<ShufflingTestReport> report, bool binary = false) {
        Random randGen = new Random();
        var ranks = new List<RankRecord>();
        int subsetLength = dataset.Length / subsetNumber;
        Console.Write("Shuffling test: Start... ");
        double percentage = .0;
        double stepPercent = 100.0 / subsetNumber;
        for(int i = 0; i < subsetNumber; ++i) {
            int[] subset = new int[subsetLength];
            //bytes are copying
            Buffer.BlockCopy(dataset, i * subsetLength << 2,
               subset, 0, subsetLength << 2);
            //var subset = dataset.Skip(subsetLength *
               i).Take(subsetLength).ToArray();
```

```
        var initScores = StatisticalScores.getScores(subset,
           binary);
        //Shuffling
        var shufScores = new List<Score>();
        Action[] shufTasks = new Action[shuffleNumber];
        for(int j = 0; j < shuffleNumber; ++j) {
            shufTasks[j] = new Action(() => {
                int[] subsetCopy = new int[subset.Length];
                lock(shufLock) {
                    Utilities.shuffle(subset, randGen);
                    Buffer.BlockCopy(subset, 0, subsetCopy, 0,
                       subset.Length << 2);
                }
                var scores =
                   StatisticalScores.getScores(subsetCopy,
                   binary);
                lock(shufScores) {
                    shufScores.AddRange(scores);
                }
            });
        }
        var parallelOptions = new ParallelOptions() {
            MaxDegreeOfParallelism = Utilities.CoreCount
        };
        Parallel.Invoke(parallelOptions, shufTasks);
        percentage += stepPercent;
        Console.Write("{0:0.}% ... ", percentage);
        //Shuffled subsets scores lists
        foreach(var initScore in initScores) {
            var scoreList = shufScores.Where(x => x.Name ==
               initScore.Name).ToList();
            int lowerRank = scoreList.Where(x => x.Value <
               initScore.Value).Count();
            int upperRank = scoreList.Where(x => x.Value <=
               initScore.Value).Count();
            int r = shuffleNumber >> 1;
            int rank = upperRank < r ? upperRank : (lowerRank
               > r ? lowerRank : r);
            ranks.Add(new RankRecord {
                Name = initScore.Name,
                Subset = i,
                Rank = rank
            });
        }
    }
    report = new List<ShufflingTestReport>();
    TestResult result = TestResult.Passed;
    foreach(var records in ranks.GroupBy(x => x.Name)) {
        int[] curRanks = (from r in records
                          orderby r.Subset
                          select r.Rank).ToArray();
        TestResult curResult = curRanks.Where(x => x < 50 || x
           > 950).Count() >= 8 ? TestResult.Failed :
```

```
                TestResult.Passed;
            if(curResult == TestResult.Failed) {
                result = TestResult.Failed;
            }
            report.Add(new ShufflingTestReport {
                Name = records.First().Name,
                P = curRanks,
                Result = curResult
            });
        }
        Console.WriteLine("\nShuffling test: Finished");
        return result;
    }
}
```

A.1.2 Вычисление статистических оценок

```
public struct Score
{
    public string Name;
    public decimal Value;
}

/// <summary>
/// introduces statistical scores used in IID tests
/// </summary>
public static class StatisticalScores
{
    public static List<Score> getScores(int[] sample, bool binary
       = false) {
        var scores = new List<Score>();
        scores.AddRange(compressionScores(sample, binary));
        scores.AddRange(runsScores(sample, binary));
        scores.AddRange(excursionScores(sample, binary));
        scores.AddRange(directionalRunsScores(sample, binary));
        scores.AddRange(covarianceScores(sample, binary));
        scores.AddRange(collisionScores(sample, binary));
        return scores;
    }
```

A.1.2.1 Оценка сжатия

```
    /// <summary>
    /// calculates compression score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>length of the compressed string in bytes</returns>
    public static Score[] compressionScores(int[] sample, bool
       binary = false) {
        byte[] input = Encoding.UTF8.GetBytes(string.Join(",",
           sample));
        //Console.WriteLine(str);
        byte[] output = BZip2.Compress(input, 4096);
        return new Score[]{
```

```
            new Score{
                Name = "compression score",
                Value = (output != null) ? output.Length : -1
            }
        };
    }
```

A.1.2.2 Оценки серий

```
    /// <summary>
    /// calculates Over/Under Runs Score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>longest run and total number of runs</returns>
    public static Score[] runsScores(int[] sample, bool binary =
       false) {
        int[] median = null;
        if(binary) {
            median = new int[] { 0, 1 };
        } else {
            median = StandardFunctions.integerMedian(sample);
        }
        //false: value is under the median
        //true: value is over the median
        //skipped: value is equal to the median
        List<bool> symbols = new List<bool>();
        for(int i = 0; i < sample.Length; ++i) {
            if(sample[i] < median[1]) {
                symbols.Add(false);
            } else if(sample[i] > median[0]) {
                symbols.Add(true);
            }
        }
        //first symbol produces first run of length 1
        int currentRunLength = 1;
        bool currentSymbol = symbols[0];
        int longestRunLength = 1;
        int runsNumber = 1;
        ////////////////////////////////////////
        for(int i = 1; i < symbols.Count; ++i) {
            if(symbols[i] != currentSymbol) {
                //the next run
                if(currentRunLength > longestRunLength) {
                    longestRunLength = currentRunLength;
                }
                currentRunLength = 1;
                ++runsNumber;
            } else {
                //the same run
                ++currentRunLength;
            }
        }
        if(currentRunLength > longestRunLength) {
            longestRunLength = currentRunLength;
```

```
        }
        return new Score[]{
            new Score{
                Name = "Longest run score",
                Value = longestRunLength
            },
            new Score{
                Name = "Runs number score",
                Value = runsNumber
            }
        };
    }
```

A.1.2.3 Оценка кумулятивного отклонения

```
    /// <summary>
    /// calculates excursion score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>maximal deviation from expected value</returns>
    public static Score[] excursionScores(int[] sample, bool
       binary = false) {
        if(sample.Length == 0) {
            return new Score[]{
            new Score{
                Name = "Excursion score",
                Value = 0
            }
        };
        }
        int initRemainder;
        int initAverage = StandardFunctions.integerAverage(sample,
           out initRemainder);
        //decimal average = initAverage + initRemainder * 100 /
           sample.Length;
        decimal average = initAverage + (decimal)initRemainder /
           sample.Length;
        decimal excursion = 0;
        decimal maxAbsValue = 0;
        for(int i = 0; i < sample.Length; ++i) {
            excursion += sample[i] - average;
            if(Math.Abs(excursion) > maxAbsValue) {
                maxAbsValue = Math.Abs(excursion);
            }
        }
        return new Score[]{
            new Score{
                Name = "Excursion score",
                Value = maxAbsValue
            }
        };
    }
```

A.1.2.4 Оценки направленных серий

```
/// <summary>
/// calculates directional runs score
/// </summary>
/// <param name="sample">sample to analyse</param>
/// <returns>longest run, total number of runs, total number
   of 1 or -1 (which is greater)</returns>
public static Score[] directionalRunsScores(int[] sample, bool
   binary = false) {
    if(binary){
        int count = sample.Length >> 3;
        int[] hammingWeights = new int[count];
        for(int j = 0; j < count; ++j) {
            for(int i = 0; i < 8; ++i) {
                hammingWeights[j] += sample[(j << 3) + i];
            }
        }
        return directionalRunsScores(hammingWeights, false);
    }
    //creation of derivatives array
    sbyte[] derivatives = new sbyte[sample.Length - 1];
    int numberOfOnes = 0;
    int numberOfMinusOnes = 0;
    //было for(int i = 0; i < derivatives.Length - 1; ++i) {
    for (int i = 0; i < derivatives.Length; ++i) {
        if(sample[i] < sample[i + 1]) {
            derivatives[i] = 1;
            ++numberOfOnes;
        } else if(sample[i] > sample[i + 1]) {
            derivatives[i] = -1;
            ++numberOfMinusOnes;
        } else {
            derivatives[i] = 0;
        }
    }
    //skipping leading 0
    int currentPosition = 0;
    while(derivatives[currentPosition] == 0 && currentPosition
       < derivatives.Length - 1) {
        ++currentPosition;
    }
    if(derivatives[currentPosition] == 0) {
        return new Score[]{
            new Score{
                Name = "Longest direct. run score",
                Value = 0
            },
            new Score{
                Name = "Direct. runs number score",
                Value = 0
            },
            new Score{
```

```
                Name = "Direct. runs -1/1 score",
                Value = 0
            }
        };
    }
    ////////////////////////////////////////////////
    int currentValue = derivatives[currentPosition];
    ++currentPosition;
    int runsNumber = 1;
    int currentRunLength = 1;
    int longestRunLength = 1;
    for(; currentPosition < derivatives.Length;
       ++currentPosition) {
        if(derivatives[currentPosition] == -currentValue) {
            //the next run (zeros do not break the run)
            ++runsNumber;
            if(currentRunLength > longestRunLength) {
                longestRunLength = currentRunLength;
            }
            currentRunLength = 1;
        } else {
            //the same run
            ++currentRunLength;
        }
    }
    if(currentRunLength > longestRunLength) {
        longestRunLength = currentRunLength;
    }
    return new Score[]{
        new Score{
            Name = "Longest direct. run score",
            Value = longestRunLength
        },
        new Score{
            Name = "Direct. runs number score",
            Value = runsNumber
        },
        new Score{
            Name = "Direct. runs -1/1 score",
            Value = Math.Max(numberOfOnes, numberOfMinusOnes)
        }
    };
}
```

A.1.2.5 Оценка ковариации

```
/// <summary>
/// calculates covariance score
/// </summary>
/// <param name="sample">sample to analyse</param>
/// <returns>covariance estimate</returns>
public static Score[] covarianceScores(int[] sample, bool
   binary = false) {
    int average = StandardFunctions.integerAverage(sample);
```

```
        int[] counts = new int[sample.Length - 1];
        for(int i = 0; i < counts.Length; ++i) {
            counts[i] = (sample[i] - average) * (sample[i + 1] -
               average);
        }
        return new Score[]{
            new Score{
                Name = "Covariance score",
                Value = StandardFunctions.integerAverage(counts)
            }
        };
    }
```

A.1.2.6 Оценки коллизий

```
    /// <summary>
    /// calculates collision score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>number of values to collision: minimal, maximal
       and average</returns>
    public static Score[] collisionScores(int[] sample, bool
       binary = false) {
        var collisions = StandardFunctions.allCollisions(sample);
        if(collisions.Count == 0) {
            return new Score[]{
                new Score{
                    Name = "Collision min score",
                    Value = -1
                },
                new Score{
                    Name = "Collision max score",
                    Value = -1
                },
                new Score{
                    Name = "Collision average score",
                    Value = -1
                }
            };
        } else {
            return new Score[]{
                new Score{
                    Name = "Collision min score",
                    Value = collisions.Min()
                },
                new Score{
                    Name = "Collision max score",
                    Value = collisions.Max()
                },
                new Score{
                    Name = "Collision average score",
                    Value =
                       StandardFunctions.integerAverage(collisions)
                }
```

```
            };
        }
    }
}
```

Поиск коллизий:

```
    public static int nextCollision<T>(T[] dataset, int
       startPosition) {
        var values = new HashSet<T>();
        int currentPosition = startPosition;
        while(currentPosition < dataset.Length) {
            if(values.Contains(dataset[currentPosition])){
                //the next collision is found
                return currentPosition;
            } else {
                values.Add(dataset[currentPosition]);
            }
            ++currentPosition;
        }
        //no collision is found
        return -1;
    }

    public static List<int> allCollisions<T>(T[] dataset) {
        var collisions = new List<int>();
        int currentPosition = 0;
        int nextPosition;
        while((nextPosition = nextCollision(dataset,
           currentPosition)) != -1) {
            collisions.Add(nextPosition - currentPosition + 1);
            currentPosition = nextPosition + 1;
        }
        return collisions;
    }
```

A.2 Хи-квадрат тесты

A.2.1 Хи-квадрат тест для проверки независимости для небинарных данных

```
public static ChiSquaredTestReport independenceTest(int[] dataset)
   {
        var report = new ChiSquaredTestReport{Name = "Chi-squared
           independence test"};
        var probs = dataset.GroupBy(x => x).ToDictionary(x =>
           x.Key, x => (double)x.Count() / dataset.Length);
        double maxProb = probs.Max(x => x.Value);
        double freqThreshold = 5.0 / dataset.Length;
        var freqProbs = from x in probs
                        where x.Value * maxProb >= freqThreshold
                        select x;
        int paramNumber = 1 + freqProbs.Count();
        if(paramNumber == 1) {
```

```
        report.Result = TestResult.Skipped;
        report.Comment = "Not enough data";
        return report;
    }
    var allPairs = from x in freqProbs
                   from y in freqProbs
                   select new KeyValuePair<Tuple<int, int>,
                      double>(Tuple.Create(x.Key, y.Key),
                      x.Value * y.Value * (dataset.Length -
                      1));
    var expFreqPairs = (from pair in allPairs
                        where pair.Value >= 5
                        select pair)
                        .ToDictionary(pair => pair.Key, pair
                           => pair.Value);
    int existRarePair = (expFreqPairs.LongCount() <
       (long)probs.Count * (long)probs.Count) ? 1 : 0;
    double expRarePairsFreq = dataset.Length - 1 -
       expFreqPairs.Aggregate(0.0, (freq, pair) => freq +
       pair.Value);
    int degreesOfFreedom = expFreqPairs.Count() + 1 -
       paramNumber;
    if(degreesOfFreedom < 1) {
        report.Result = TestResult.Skipped;
        report.Comment = "Not enough data";
        return report;
    }
    var neighbors = dataset.Skip(1).Zip(dataset, (next, prev)
       => Tuple.Create(prev, next));
    var observedPairs = neighbors.GroupBy(pair =>
       pair).ToDictionary(x => x.Key, x => x.Count());
    var obsFreqPairs = (from pair in observedPairs
                        where expFreqPairs.ContainsKey(pair.Key)
                        select pair)
                        .ToDictionary(pair => pair.Key, pair =>
                           pair.Value);
    if(existRarePair > 0) {
        int obsRarePairsFreq = (from pair in observedPairs
        where !expFreqPairs.ContainsKey(pair.Key)
                                select pair)
                                .Aggregate(0, (freq, pair) =>
                                   freq + pair.Value);
        report.Stat = (expRarePairsFreq - obsRarePairsFreq) *
           (expRarePairsFreq - obsRarePairsFreq);
        report.Stat /= expRarePairsFreq;
    }
    foreach(var pair in expFreqPairs) {
        double obsFreq = obsFreqPairs.ContainsKey(pair.Key) ?
           obsFreqPairs[pair.Key] : 0.0;
        obsFreq -= pair.Value;
        report.Stat += obsFreq * obsFreq / pair.Value;
    }
    report.Df = expFreqPairs.Count - 1 + existRarePair;
```

```
        report.P = SpecMath.chisqc(report.Df, report.Stat);
        report.Result = report.P > confidenceLevel ?
           TestResult.Passed : TestResult.Failed;
        return report;
    }
```

A.2.2 Хи-квадрат тест для проверки одинаковой распределенности для небинарных данных

```
public static List<ChiSquaredTestReport> goodnessOfFitTest(int[]
   dataset) {
        var reportList = new
           List<ChiSquaredTestReport>(subsetNumber);
        var report = new ChiSquaredTestReport{
            Name = "Chi-squared goodness-of-fit test"
        };
        int subsetLength = dataset.Length / subsetNumber;
        double multiplier = (double) subsetLength / dataset.Length;
        var probs = dataset.GroupBy(item =>
           item).ToDictionary(item => item.Key, item => multiplier
           * item.Count());
        double probThreshold = 5.0;
        var freqValues = (from item in probs
                          where item.Value >= probThreshold
                          select item).
                          ToDictionary(item => item.Key, item =>
                             item.Value);
        if(freqValues.Count == 0) {
            report.Result = TestResult.Skipped;
            report.Comment = "Not enough data";
            reportList.Add(report);
            return reportList;
        }
        int existRareValue = (freqValues.Count < probs.Count) ? 1
           : 0;
        double rareValuesProb = dataset.Length -
           freqValues.Aggregate(0.0, (prob, item) => prob +
           item.Value);
        report.Df = freqValues.Count - 1 + existRareValue;
        for(int i = 0; i < subsetNumber; ++i) {
            report.Comment = string.Format("Subset {0}", i);
            var subset = dataset.Skip(i *
               subsetLength).Take(subsetLength).GroupBy(item =>
               item).ToDictionary(item => item.Key, item =>
               item.Count());
            var subsetFreqValues = (from item in subset
            where freqValues.ContainsKey(item.Key)
                                    select item)
                                    .ToDictionary(item =>
                                       item.Key, item =>
                                       item.Value);
            report.Stat = .0;
            if(existRareValue > 0) {
```

```
                int subsetRareValuesProb = (from item in subset
                where !freqValues.ContainsKey(item.Key)
                                        select item)
                                        .Aggregate(0, (prob,
                                           item) => prob +
                                           item.Value);
                report.Stat = (rareValuesProb -
                   subsetRareValuesProb) * (rareValuesProb -
                   subsetRareValuesProb);
                report.Stat *= report.Stat / rareValuesProb;
            }
            foreach(var item in freqValues) {
                double obsProb =
                   subsetFreqValues.ContainsKey(item.Key) ?
                   subsetFreqValues[item.Key] : 0;
                obsProb -= item.Value;
                report.Stat += obsProb * obsProb / item.Value;
            }
            report.P = SpecMath.chisqc(report.Df, report.Stat);
            report.Result = report.P > confidenceLevel ?
               TestResult.Passed : TestResult.Failed;
            reportList.Add(report);
            if(report.Result == TestResult.Failed) {
                return reportList;
            } else {
                report = new ChiSquaredTestReport{
                    Name = report.Name,
                    Df = report.Df
                };
            }
        }
        return reportList;
    }
```

A.2.3 Хи-квадрат тесты для проверки независимости и одинаковой распределенности для бинарных данных

```
public static List<ChiSquaredTestReport> binaryTest(int[] dataset)
   {
        var reportList = new List<ChiSquaredTestReport>(10 +
           subsetNumber);
        int onesNumber = dataset.Count(value => value == 1);
        int zerosNumber = dataset.Length - onesNumber;
        double onesProb = (double) onesNumber / dataset.Length;
        double zerosProb = 1.0 - onesProb;
        double minProb = Math.Min(onesProb, zerosProb);
        double freq = minProb * minProb * dataset.Length;
        int subsetLength;
        for(int k = 2; (k <= 11) && (freq > 5); ++k, freq *=
           minProb) {
            var report = new ChiSquaredTestReport{
                Name = "Chi-squared independence test for binary
                   data",
```

```
                Comment = string.Format("{0}-bit chunks", k)
            };
            subsetLength = dataset.Length / k;
            var subset = new ushort[subsetLength];
            for(int i = 0; i < subsetLength; ++i) {
                for(int j = 0; j < k; ++j) {
                    subset[i] += (ushort)(dataset[i * k + j] << (k
                        - j - 1));
                }
            }
            var subsetFreq = subset.GroupBy(value =>
                value).ToDictionary(item => (ushort)item.Key, item
                => item.Count());
            report.Stat = .0;
            for(ushort i = 0; i < (1 << k); ++i) {
                byte hw = Utilities.hammingWeight(i);
                double expFreq = Math.Pow(onesProb, hw) *
                    Math.Pow(zerosProb, k - hw) * subsetLength;
                double obsFreq = subsetFreq.ContainsKey(i) ?
                    subsetFreq[i] : 0;
                report.Stat += (expFreq - obsFreq) * (expFreq -
                    obsFreq) / expFreq;
            }
            report.Df = (1 << k) - 1;
            report.P = SpecMath.chisqc(report.Df, report.Stat);
            report.Result = report.P > confidenceLevel ?
                TestResult.Passed : TestResult.Failed;
            reportList.Add(report);
            if(report.Result == TestResult.Failed) {
                break;
            }
        }
        var report2 = new ChiSquaredTestReport{
            Name = "Chi-squared goodness-of-fit test for binary
                data"
        };
        subsetLength = dataset.Length / subsetNumber;
        double expOnesNum = onesProb * subsetLength;
        for(int i = 0; i < subsetNumber; ++i) {
            int obsOnesNum = dataset.Skip(i *
                subsetLength).Take(subsetLength).Count(value =>
                value == 1);
            report2.Stat += (expOnesNum - obsOnesNum) *
                (expOnesNum - obsOnesNum) / expOnesNum;
        }
        report2.Df = subsetNumber - 1;
        report2.P = SpecMath.chisqc(report2.Df, report2.Stat);
        report2.Result = report2.P > confidenceLevel ?
            TestResult.Passed : TestResult.Failed;
        reportList.Add(report2);
        return reportList;
    }
}
```

A.3 Оценки минимальной энтропии

A.3.1 Оценка минимальной энтропии последовательности случайных величин, являющихся н.о.р.

```
/// <summary>
/// estimates minimal entropy for IID sources
/// </summary>
/// <param name="dataset"></param>
/// <param name="sampleSize">number of bits in each
   sample</param>
/// <returns>entropy estimate</returns>
public static double iidTest(int[] dataset, byte sampleSize)
{
    double maxProb = dataset.GroupBy(x => x).Max(x =>
       x.Count()) / (double)dataset.Length;
    double pmax = maxProb + 2.3 * Math.Sqrt(maxProb * (1 -
       maxProb) / dataset.Length);
    double entropy = -Math.Log(pmax);
    return Math.Min(entropy, sampleSize);
}
```

A.3.2 Тест коллизий

```
        private static double solveBinarySearch(Func<double,
           double, double, double> f, double n, double m) {
                double eps = 0.00001;
                double high = 1 - eps;
                double low = eps;
                double highF = f (high, n, m);
                double lowF = f (low, n, m);
                if (highF * lowF > 0)
                        throw new Exception("Method not
                           applicable");
                double diff;
                double diffF;
                do {
                        diff = high - low;
                        diffF = Math.Max(Math.Abs(high),
                           Math.Abs(low));
                        double middle = (high + low) / 2;
                        double middleF = f(middle,n,m);
                        if(middleF * highF > 0) {
                                high = middle;
                                highF = middleF;
                        } else {
                                low = middle;
                                lowF = middleF;
                        }
                } while ((diff>eps) || (diffF<eps));
                if (Math.Abs (high) < Math.Abs (low))
                        return high;
                return low;
```

```
            }

            private static double expectFunction(double p, double
               n, double m) {
                    double q = (1 - p) / (n - 1);
                    double f = SpecMath.igamma(n+1,1/q) * Math.Pow
                       (q, n + 1) * Math.Exp (-1/q);
                    double x = (1 / p - 1 / q) / n;
     double z = p / q * (1 / p - 1 / q) / n;
                    double y = (1 + x) * f / q;
                    return (y - x) * p / q - z - m;
            }

/// <summary>
/// estimates minimal entropy for non-IID sources using
   collision test
/// </summary>
/// <param name="dataset"></param>
/// <param name="sampleSize">number of bits in each
   sample</param>
/// <returns>entropy estimate</returns>
public static double collisionTest(int[] dataset, byte
   sampleSize) {
    //throw new Exception("not implemented");
    //you need to observe at least 1000 collisions
    //see birthday problem to calculate necessary length of
       the dataset
    //depending on the number of bits in a sample [e.g.
       Ramanujan asymptotic]
    var collisions = StandardFunctions.allCollisions(dataset);
    if(collisions.Count < 1000) {
        throw new NotEnoughDataException("Collision test:
           there are not enough collisions in the dataset");
    }
    double lowerBound =
       StandardFunctions.confidenceInterval(collisions)[0];
                    double p = solveBinarySearch (expectFunction,
                       sampleSize, lowerBound);
                    return -Math.Log (p, 2);
}

            private static double epFunction(double p, double n,
               double m) {
                    double q = (1 - p) / (n - 1);
                    double f = 1 - Math.Pow (1 - p, n);
                    double g = (n-1)*(1 - Math.Pow (1 - q, n));
                    return f+ g - m;
            }
```

A.3.3 Тест частичных коллекций

```
            private static double epFunction(double p, double n,
               double m) {
                    double q = (1 - p) / (n - 1);
```

```
                    double f = 1 - Math.Pow (1 - p, n);
                    double g = (n-1)*(1 - Math.Pow (1 - q, n));
                    return f+ g - m;
                }

        /// <summary>
        /// estimates minimal entropy for non-IID sources using
           partial collection test
        /// </summary>
        /// <param name="dataset"></param>
        /// <param name="sampleSize">number of bits in each
           sample</param>
        /// <returns>entropy estimate</returns>
        public static double partialCollectionTest(int[] dataset, byte
           sampleSize) {
            //the output space or indication of values that never
               appear in the output SHALL be
            //provided by the developer for validation testing

            //you need to observe minimum 500 events, so dataset
               length need to be at least 500*2^b,
            //where b is the number of bits in a sample
            int outputSpaceSize = 1 << sampleSize;
            int[] distinctValuesNumbers = new int[dataset.Length /
               outputSpaceSize];
            var valuesSet = new HashSet<int>();
            for(int subset = 0; subset < distinctValuesNumbers.Length;
               ++subset) {
                for(int position = 0; position < outputSpaceSize;
                   ++position) {
                    valuesSet.Add(dataset[subset * outputSpaceSize +
                       position]);
                }
                distinctValuesNumbers[subset] = valuesSet.Count;
                valuesSet.Clear();
            }
                    int m = distinctValuesNumbers.Max();
                    if (outputSpaceSize - m > 500) {
                            throw new
                               NotEnoughDataException("Partial
                               collection test: not enough data");
                    }
            double lowerBound =
StandardFunctions.confidenceInterval(distinctValuesNumbers)[0];
                    double p = solveBinarySearch (epFunction,
                       sampleSize, lowerBound);
                    return -Math.Log (p, 2);
        }
```

A.3.4 Тест марковской зависимости

```
        public static double markovTest(int[] dataset, byte
           sampleSize) {
```

```
                    //the largest sample size accommodated by this
                       test is 6 bits
                    int N = 1 << sampleSize;
                    int power = Math.Max (N * N, 128);
            double a = - power * Math.Log(confidenceLevel, 2);
                    double e = Math.Sqrt (0.5 * a /
                       dataset.Length);
                    if (sampleSize > 6)
                            throw new
                               NotEnoughDataException("Markov
                               test: too big output space for the
                               test");
                    SparseArray<int, double> prior = new
                       SparseArray<int, double>(0.0);
                    SparseArray<int, double> priorE = new
                       SparseArray<int, double>(0.0);
                    SparseArray<int, double> minP = new
                       SparseArray<int, double>(100000.0);
                    SparseArray<int, int> minWay = new
                       SparseArray<int, int>(-1);
                    Sparse2DMatrix<int, int, double> trans = new
                       Sparse2DMatrix<int, int, double>(1000000);
                    Sparse2DMatrix<int, int, double> vertex = new
                       Sparse2DMatrix<int, int, double>(-1.0);
                    prior [dataset [0]]++;
                    for (int i = 1; i < dataset.Length; i++) {
                            prior [dataset [i]]++;
                            trans [dataset [i - 1], dataset [i]]++;
                    }
                    foreach(KeyValuePair<int, double> pair in
                       prior)
                    {
        priorE[pair.Key] = Math.Sqrt (0.5 * a / pair.Value);
                            //Console.WriteLine("pair vector[{0}]
                               = {1}", pair.Key, pair.Value);
                    }
                    foreach (KeyValuePair<ComparableTuple2<int,
                       int>, double> pair in trans)
                    {
                            int key0 = 0;
                            int key1 = 0;
                            trans.SeparateCombinedKeys(pair.Key,
                               ref key0, ref key1);
                            trans [key0,key1] = -
                               Math.Log(Math.Min(1, pair.Value /
                               prior[key0] + priorE[key0]),2);
                            //Console.WriteLine("pair matrix[{0},
                               {1}] = {2}", key0, key1,
                               pair.Value);
                    }
                    foreach(KeyValuePair<int, double> pair in
                       prior)
                    {
```

```
                prior[pair.Key] = -
                    Math.Log(Math.Min(1, pair.Value /
                    dataset.Length + e),2);
                //Console.WriteLine("pair vector[{0}]
                    = {1}", pair.Key, pair.Value);
            }
            foreach (KeyValuePair<ComparableTuple2<int,
                int>, double> pair in trans)
            {
                int key0 = 0;
                int key1 = 0;
                trans.SeparateCombinedKeys(pair.Key,
                    ref key0, ref key1);
                if (pair.Value < minP [key0]) {
                        minP [key0] = pair.Value;
                        minWay [key0] = key1;
                }
                //Console.WriteLine("pair matrix[{0},
                    {1}] = {2}", key0, key1,
                    pair.Value);
            }
            foreach(KeyValuePair<int, double> pair in
                prior)
            {
                vertex[0,pair.Key] = pair.Value;
                //Console.WriteLine("pair vector[{0}]
                    = {1}", pair.Key, pair.Value);
            }
            for (int k = 1; k < 128; k++) {
                foreach(KeyValuePair<int, double> end
                    in prior) {
                        vertex [k, end.Key] =
                            double.MaxValue;
                        foreach(KeyValuePair<int,
                            double> begin in prior) {
                                if (vertex [k,
                                    end.Key] > vertex
                                    [k-1, begin.Key] +
                                    trans[begin.Key,
                                    end.Key]) {
                                        vertex [k,
                                            end.Key] =
                                            vertex [k -
                                            1,
                                            begin.Key]
                                            + trans
                                            [begin.Key,
                                            end.Key];
                                }
                        }
                }
            }
            double min = double.MaxValue;
```

```
                    foreach(KeyValuePair<int, double> pair in
                        prior)
                    {
                            if (min > vertex [127, pair.Key])
                                    min = vertex [127, pair.Key];
                            //Console.WriteLine("pair vector[{0}]
                                = {1}", pair.Key, pair.Value);
                    }
                    return min;
    }
```

A.3.5 Тест сжатия

```
            private static double functionG(double p, double n) {
                    double firstSum = 0;
                    for (int t = dictionaryLearningSetLength+1; t
                        <= n; t++) {
                            double secondSum = 0;
                            for (int s = 1; s < t; s++)
                                    secondSum += Math.Log (s, 2) *
                                        p * p * Math.Pow (1 - p, s
                                        - 1);
                            secondSum += Math.Log (t, 2) * p *
                                Math.Pow (1 - p, t - 1);
                            firstSum += secondSum;
                    }
                    return firstSum / (n -
                        dictionaryLearningSetLength);
            }

            private static double functionE(double p, double
                n,double m) {
                    double q = (1 - p) / (n - 1);
                    double Ep = functionG (p, n) + (n - 1) *
                        functionG (q, n);
                    return Ep - m;
            }

    private static readonly int dictionaryLearningSetLength = 1000;

    /// <summary>
    /// estimates minimal entropy for non-IID sources using
        compression test
    /// </summary>
    /// <param name="dataset"></param>
    /// <param name="sampleSize">number of bits in each
        sample</param>
    /// <returns>entropy estimate</returns>
    public static double compressionTest(int[] dataset, byte
        sampleSize) {
         var dictionary = new Dictionary<int, int>();
         if(dataset.Length <= dictionaryLearningSetLength){
            throw new NotEnoughDataException("Compression test:
                not enough data");
```

```
        }
        //first part of the dataset is used to create dictionary
        int position;
        for(position = 0; position < dictionaryLearningSetLength;
           ++position) {
            if(dictionary.ContainsKey(dataset[position])) {
                dictionary[dataset[position]] = position;
            } else {
                dictionary.Add(dataset[position], position);
            }
        }
        //second part of the dataset is used for the test
        var repetitions = new List<int>(dataset.Length - position);
        for(; position < dataset.Length; ++position) {
            if(dictionary.ContainsKey(dataset[position])) {
                repetitions.Add(position -
                   dictionary[dataset[position]]);
                dictionary[dataset[position]] = position;
            } else {
                repetitions.Add(position);
                dictionary.Add(dataset[position], position);
            }
        }
        double lowerBound =
           StandardFunctions.confidenceInterval(repetitions)[0];
                    double p = solveBinarySearch (functionE,
                       sampleSize, lowerBound);
                    return -Math.Log (p, 2);
    }
```

A.3.6 Частотный тест

```
    /// <summary>
    /// estimates minimal entropy for non-IID sources using
       frequency test
    /// </summary>
    /// <param name="dataset"></param>
    /// <param name="sampleSize">number of bits in each
       sample</param>
    /// <returns>entropy estimate</returns>
    public static double frequencyTest(int[] dataset, byte
       sampleSize) {
        double maxProb = dataset.GroupBy(x => x).Max(x =>
           x.Count()) / (double)dataset.Length;
        return -Math.Log(maxProb + Math.Sqrt(0.5 * Math.Log(1 / (1
           - confidenceLevel), 2) / dataset.Length), 2);
    }
```

ПРИЛОЖЕНИЕ Б

Пример вывода результатов применения разработанного ПО

```
>EntropySourceTesting.exe data1.bin 1
Имя файла: data1.bin
Размер наблюдения в битах: 1

Хи-квадрат тест независимости для бинарных данных
2-битные строки
Статистика: 1,05940167178556
Степени свободы: 3
P-значение: 0,786882595535632
 ПРОЙДЕНО
3-битные строки
Статистика: 8,59778029152712
Степени свободы: 7
P-значение: 0,282837419348987
 ПРОЙДЕНО
4-битные строки
Статистика: 8,5445215312206
Степени свободы: 15
P-значение: 0,900104722757102
 ПРОЙДЕНО
5-битные строки
Статистика: 17,3931705350896
Степени свободы: 31
P-значение: 0,976553796184551
 ПРОЙДЕНО
6-битные строки
Статистика: 78,4500673374023
Степени свободы: 63
P-значение: 0,0907788166889782
 ПРОЙДЕНО
7-битные строки
Статистика: 123,073552236177
Степени свободы: 127
P-значение: 0,581994389766079
 ПРОЙДЕНО
8-битные строки
Статистика: 225,724166390702
Степени свободы: 255
P-значение: 0,906542467568111
 ПРОЙДЕНО
9-битные строки
Статистика: 533,320631367063
Степени свободы: 511
P-значение: 0,239141067880565
 ПРОЙДЕНО
10-битные строки
Статистика: 1026,14996542014
Степени свободы: 1023
P-значение: 0,466407672790989
```

```
 ПРОЙДЕНО
11-битные строки
Статистика: 2038,33725348191
Степени свободы: 2047
P-значение: 0,549800289599266
 ПРОЙДЕНО

Хи-квадрат тест одинаковой распределенности для бинарных данных
Статистика: 3,06259909290993
Степени свободы: 9
P-значение: 0,961761274722441
 ПРОЙДЕНО
Хи-квадрат тест ПРОЙДЕНО
Тест сжатия: Старт... 100% ...
Тест перестановок: Окончен
Оценка сжатия
313 289 144 289 197 840 787 925 557 817
 ПРОЙДЕНО
Оценка самой длинной серии
670 434 911 500 826 436 659 500 467 43
 ПРОЙДЕНО
Оценка количества серий
500 500 500 526 499 495 481 475 500 516
 ПРОЙДЕНО
Оценка кумулятивного отклонения
434 673 900 288 125 220 659 497 327 400
 ПРОЙДЕНО
Оценка самой длинной направленной серии
500 500 778 928 406 500 440 500 786 399
 ПРОЙДЕНО
Оценка количества направленных серий
61 502 348 131 734 688 337 678 634 388
 ПРОЙДЕНО
Оценка количества -1/1 в направленных сериях
652 271 130 193 546 470 703 466 862 697
 ПРОЙДЕНО
Оценка ковариации
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка минимального количества коллизий
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка максимального количества коллизий
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка среднего количества коллизий
500 514 502 492 494 481 500 504 480 504
 ПРОЙДЕНО
Тест перестановок ПРОЙДЕНО
Н.о.р. тест ПРОЙДЕНО

Оценка энтропии для н.о.р. данных: 0,692203546584333
Минимальная оценка энтропии: 0,692203546584333
```