Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э.Баумана)

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:

Метод оценки согласованности разметки данных об именованных сущностях для машинного обучения

Студент ИУ7-85 _____ Л. К. Кочкарова

Руководитель ВКР _____ Э. С. Клышинский

Консультант _____ _____

Консультант _____ Ю. М. Гаврилова_____

Нормоконтролер _____ Ю. В. Строганов _____

Москва, 2020

Реферат

Расчетно-пояснительная записка с. 66, рис. 63, табл. 6, 15 источников.

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ, ТЕОРИЯ ФРАКТАЛОВ, ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ, СЛОВАРНОЕ КОДИРОВАНИЕ, АЛГОРИТМЫ СЖАТИЯ.

Объект исследования: таблицы реляционных баз данных.

Цель работы - разработка метода фрактального сжатия таблиц для реляционных баз данных.

Поставленная задача достигается за счет применения теории фракталов.

Содержание

Введение

Информационные технологии в настоящее время имеют всё большее влияние в самых разных областях человеческой деятельности. С каждым годом объем накопленной информации стремительно увеличивается. Согласно предварительным оценкам, через пять лет суммарный объем информации составит порядка 10^{21} байт. При этом большая часть этой информации будет произведена не обычными пользователями, а корпорациями - на них придется более 60% от этого количества.

Корпоративные базы данных содержат огромное количество самых разнообразных сведений. В настоящее время крупнейшие базы данных работают с объемами информации в 10^{15} байт. Несмотря на возрастающий уровень технологий и постоянное удешевление оперативной памяти, хранение таких объемов данных всё ещё дорого. Современные базы данных нуждаются в применении различных методов сжатия для эффективного хранения информации.

Как правило, под базами данных в первую очередь понимают реляционные базы данных, представляющие собой наборы связанных между собой структурированных таблиц. Согласно статистике за 2019 год реляционные базы данных составляют 60.5% используемых баз данных.

Целью данной работы является рассмотрение реляционных баз данных с точки зрения теории фракталов и разработка метода сжатия данных.

1 Аналитический раздел

1.1 Цели и задачи

Целью данной работы является создание программного комплекса для сжатия реляционных баз данных. Для достижения данной цели необходимо решить следующие задачи:

а) пронализировать предметную область и существующие виды баз данных;

б) проанализироать способы сжатия баз данных;

в) разработать программный комплекс, реализующий предложенный метод;

г) провести исследование с использованием разработанного программного комплекса.

1.2 Предметная область

Основным понятием, используемым в данной работе, является фрактал. Термин фрактал происходит от латинского слова fractus, что в переводе означает сломленный, дробленый. [1]

Существует несколько равнозначимых определений фрактала:

— Фрактал - это геометрическая фигура, в которой один и тот же фрагмент повторяется при каждом уменьшении масштаба (Лаверье).

— Фрактальной называется структура, состоящая из частей, которые в каком-то смысле подобны целому (Мандельброт).

— Фракталы - это объекты, которые мы называем неправильными, шероховатыми, пористыми или раздробленными, причем указанными свойствами фракталы обладают в одинаковой степени в любом масштабе (Мандельброт).

Фрактал как явление и понятие имеет следующие признаки:

а) он обладает самоподобием – или приближенным самоподобием;
б) он обладает нетривиальной структурой;
в) он обладает дробной размерностью.

Фракталы были описаны в 1977 году французским математиком Бенуа Мандельбротом и с тех пор активно используются в самых разнообразных областях.

Помимо понятия фрактал определяют также понятие предфрактал. Предфрактал – самоподобный объект, каждый фрагмент которого повторяется в упрощенном виде при уменьшении масштаба. Большая часть объектов природы является именно предфракталами, а не фракталами, как принято считать.

Существует несколько классификаций фракталов. [2] Самой распространенной классификацией является следующая:

а) Геометрические фракталы – самая известная в силу своей наглядности группа фракталов. Могут строиться как на основе некоторой ломаной линии («двумерные» фракталы), так и на основе некоторой поверхности («трехмерные» фракталы). Основу фрактала называют генератором, и на каждой итерации часть генератора заменяется генератором в масштабе.

Примеры геометрических фракталов: снежинка Коха, T-квадрат, треугольник Серпинского, дерево Пифагора.

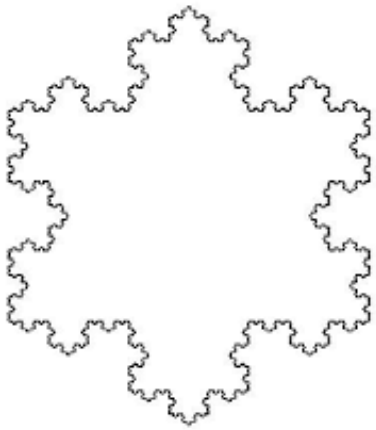

Рисунок 1.1 — Пример геометрического фрактала

б) Алгебраические фракталы – самая обширная группа фракталов. Алгебраические фракталы получают с помощью нелинейных процессов в n-мерных пространствах.
Пример алгебраического фрактала: множество Мандельброта.

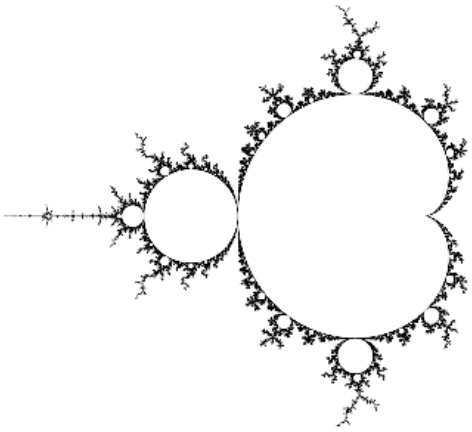

Рисунок 1.2 — Пример алгебраического фрактала

в) Стохастические фракталы. Данные фракталы получаются, если в итерационном процессе менять хаотично некоторые параметры фрактала. Объекты, полученные таким образом, похожи на объекты реального мира – например, на береговые линии.

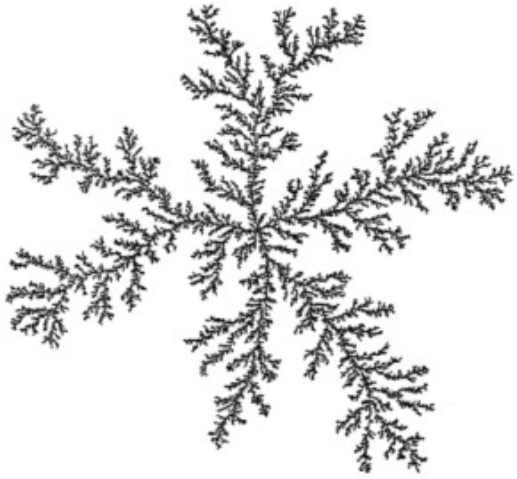

Рисунок 1.3 — Пример стохастического фрактала

По формальному признаку фракталы бывают:

а) связанные;

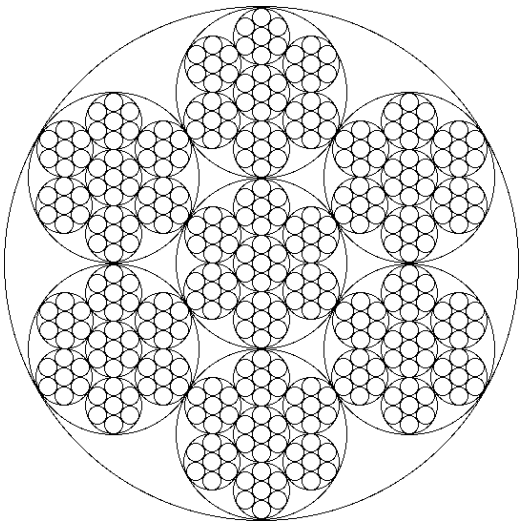

Рисунок 1.4 — Пример связанного фрактала

б) несвязанные.

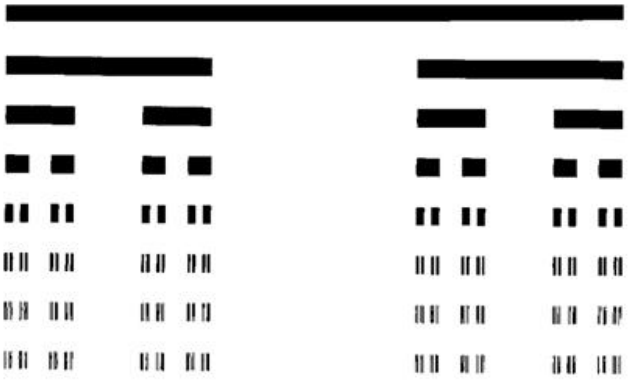

Рисунок 1.5 — Пример несвязанного фрактала

По размерности фракталы бывают:

а) с целой размерностью;

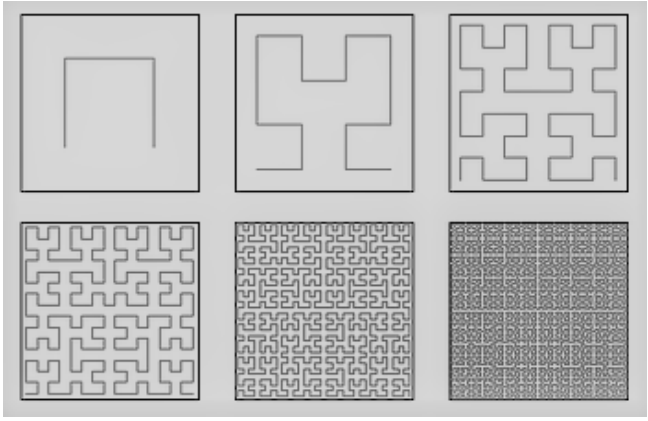

Рисунок 1.6 — Кривая Пеано

б) с дробной размерностью.

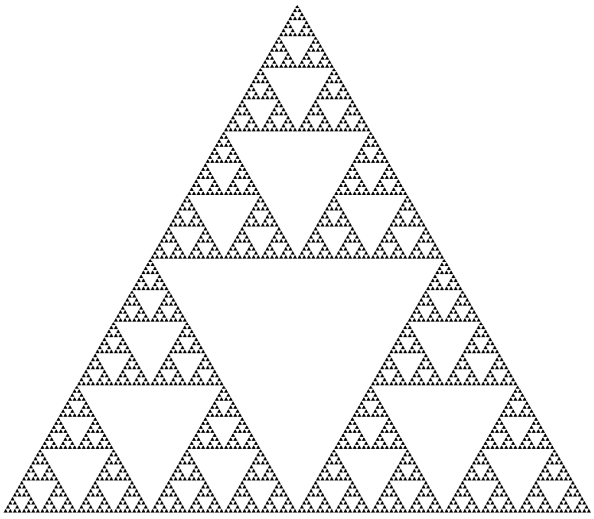

Рисунок 1.7 — Треугольник Серпинского

Помимо этого фракталы так же делятся на недетерминированные, к которым относятся геометрические и алгебраические фракталы, и детерминированные (стохастические и алеаторные), линейные, строящиеся по линейному алгоритму, и нелинейные.

В виду своей наглядности фракталы нашли свое применение в области компьютерной графики, но их также можно применять и для баз данных. В базах данных в качестве самоподобной части фрактала выступают домены.

1.3 Сжатие данных

Сжатие данных - процесс получения более компактного выходного потока из изначального некомпактного входного потока посредством преобразования первого. Все алгоритмы сжатия оперируют входным потоком, минимальной единицей которого является бит.

Сжатие данных в базах данных имеют определенные плюсы и минусы. Из плюсов можно выделить следующие:

— уменьшение физического размера самой базы данных;
— уменьшение объема журналирования за счет уменьшения объема исходной информации;
— увеличение скорости выполнения запросов;
— снижение требований к объему оперативной памяти;
— увеличение пропускных способностей каналов ввода-вывода устройств внешней памяти;
— увеличение фактической пропускной способности вычислительной сети;

Из минусов можно выделить следующие:

— увеличение расходов вычислительных ресурсов и дополнительные затраты оперативной памяти;
— усложение структуры базы данных, влекущее за собой усложнение обслуживающего базу данных программного обсепечения;
— возможное нарушение характеристик данных, таких как лескикорафическая упорядоченность;
— возрастающая вероятность неуспешного восстановления после ошибок системы.

Существуют два основных подхода к сжатию данных: сжатие с потерями и сжатие без потерь. Сжатие без потерь является обратимой операцией, то есть по представленному выходному потоку данных возможно в точности воспроизвести входной поток. При сжатии данных с потерями востановленные данные отличаются от исходных, но степень отличия не является существенной с точки зрения их дальнейшего использования. [3]

Основные технические характеристики методов сжатия:

— степень сжатия - отношение объема входного и выходого потоков; Compress Ratio $= \frac{UncompressedSize}{CompressedSize}$
— скорость сжатия - время, затраченное на сжатие входного потока;

— качество сжатия - коэффициент плотности сжатия входного потока, рассчитываемый на основании повторного сжатия.
Compress Ratio = $\frac{ompressedSize}{DoubleCompressedSize}$

В теории баз данных выделяют две схемы сжатия - легкое и тяжелое сжатие.

1.3.1 Легкое сжатие

Под схемой легкого сжатия понимают методы, основаные на словарном кодировании, преобразовании длинных значений в более короткие. В данной группе методов каждое новое уникальное значение добавляется в специальную таблицу и после кодируется в исходной таблице положением в специальной таблице. Развитие данных методов преследует цель большего сжатия последовательности словарных позиций.

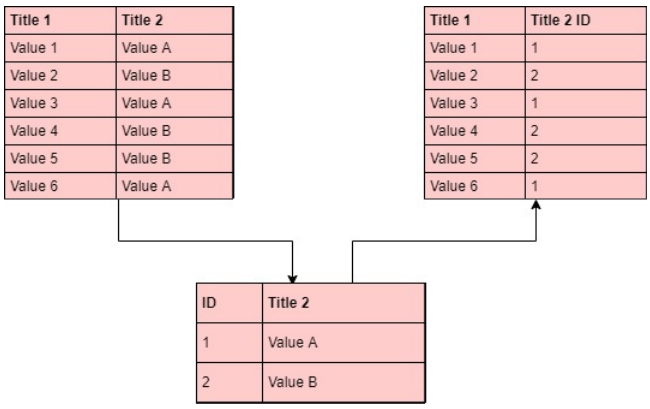

Рисунок 1.8 — Словарное сжатие

Основные методы легкого сжатия:

— подавление одинаковых значений;
— кодирование по длинам серий;
— кластерное кодирование;
— битовое сжатие;
— байтовое кодирование.

Метод подавления одинаковых значений позволяет избавиться от избыточности информации, хранящейся в таблице. На практике метод хорошо показывает себя в случаях, когда имеются столбцы с небольшим количеством ненулевых значений. При словарном кодировании повторяющиеся элементы заменяются индексом позиции в словаре, хранящим уникальные значения. Если в столбце рассредоточено большое количество уникальных значений, как правило испольщуется метод кодирования редких значений. В случае если ячейка не содержит значения,

записывается нуль, в противном случае - единица. Дополнительно хранится столбец с уникальными значениями, отсортированными в порядке появления в исходном столбце.

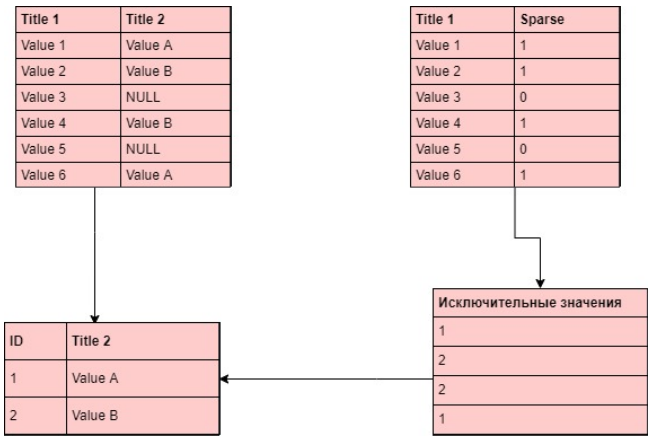

Рисунок 1.9 — Словарное исключительное сжатие

В случае если информация в столбце была заранее отсортирована, возможно использовать метод кодирования по длинам серий. Данная схема кодирования заменяет само значение на количество последовательных повторений данного значения. На практике, однако, значение заменяется на пару ячеек, где первая указывает на индекс последней строки, где встречается данное значение, а вторая - на позицию в словаре.

Рисунок 1.10 — Длины серии

Метод кластерного кодирования работает аналогично методу кодирования редких значений. Работа производится над блоками одинаковых размеров, содержащих некоторое количество различных значений. Дополнительно хранится информация о сжатии блока и расположении блока. Метод заключается в сжатии конкретного блока, например, в блоке с одним уникальным значением хранится только это значение в отдельной последовательности.

Методы битового и байтового кодирования идеологически схожи. Суть методов заключается в том, чтобы хранить данные не с помощью встроенных типов данных, а использовать строго необходимое количество информации. Значения разбиваются на части фиксированной длины, размер

определяется как наибольшая необходимая для хранения значения величина. Отличие байтового кодировани от битого заключается в использовании байта в качестве минимальной единицы хранения информации. Исходные данные разбиваются в блоки из семи битов, восьмым битом записывается нуль или единица в зависимости от того, является ли данный байт конечным байтом хранения информации. Современные ЭВМ имеют именно байтовую адресацию, что позволяет быстро и эффективно работать с байтами.

Title 1	Title 2
Value 1	Value A
Value 2	Value B
Value 3	Value A
Value 4	Value B
Value 5	Value B
Value 6	Value A

Title 1	Title 2 (bit)
Value 1	11011
Value 2	1100011
Value 3	11011
Value 4	1100011
Value 5	1100011
Value 6	11011

ID	Title 2	Bit value
1	Value A	0011011
2	NULL	0000000
3	Value B	1100011

Рисунок 1.11 — Битовое кодирование

Фрактальное сжатие данных также относится к схеме легкого сжатия, но в отличии от предыдущих методов работает не с одним столбцом, а с некоторым набором столбцов. В процессе фрактального сжатия базы данных создается словарь, хранящий множество кортежей. Выделение кортежей позволяет не только сжать данные, но и обнаружить нетривиальные корреляции. В отличие от метода подавлений одинаковых значений, фрактальному сжатию не требуется наличие большого количества нулевых значений, напротив, на этапе предобработки данных отбираются столбцы с низким коэффициентом уникальных значений. Также фрактальное сжатие не требует предварительной сортировки данных или разбиения данных на кластеры.

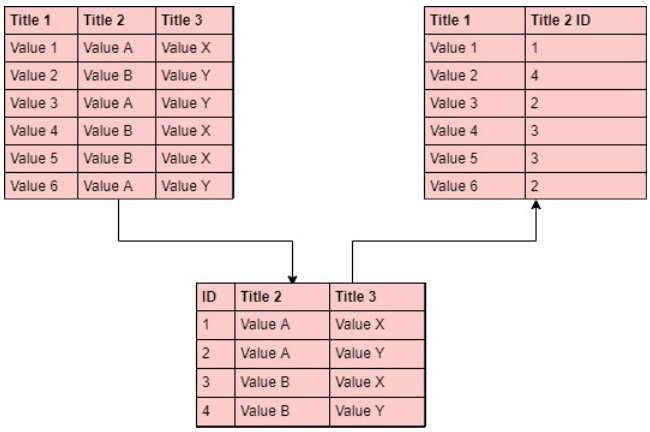

Рисунок 1.12 — Фрактальное сжатие

1.3.2 Тяжелое сжатие

В отличие от легкого сжатия, работающего с набором значений, методы тяжелого сжатия работают с набором байтов. Основная идея данной группы методов заключается в замене повторяющихся шаблонов ссылками на предыдущие упоминания.

Классическим примером тяжелого сжатия является алгоритм LZO, являющийся модификацией словарного сжатия LZ77. Преобразованные данные хранятся в виде самообращающейся таблицы, динамически строящейся во время сжатия исходных данных. Каждая запись в новой таблице уникальна и представляет собой либо уникальное значение, либо ссылку на уже имеющеющуюся запись в сочетании с дополнительным символом - символом расхождения.

Принципиальное отличие LZ77 от LZO состоит в повышенной оптимизации последнего для декомпрессии данных.

Таблица 1.1 — Сравнение технологий

Свойство данных Свойство выборки	Небольшое количество данных	Большое количество данных
Повторение значений	RLE	RLE (серии должны существовать
Сортированные значения	RLE (серии должны существовать)	LZO
Несортированные значения	Словарные сжатия	LZO
Большое кол-во значений	Словарные сжатия	LZO
Небольшое кол-во значений	Словарные сжатия	Подавление значений

1.3.3 Реляционные базы данных

База данных - это взаимосвязанная информация об объектах, огранизованная специальным образом и хранящаяся на каком-то носителе. [4]

Существуют две логические модели данных: реляционная модель и нереляционная модель.

Под реляционной моделью данных имеется в виду логическая модель данных, основанная на теории множеств и логике первого порядка. В основе реализации лежат двумерные таблицы, состоящие из строк и столбцов.

Реляционная модель данных содержит следующие компоненты: [5]

а) структурный аспект: данные представлены наборами отношений;

б) целостностный аспект: отношения отвечают определенным условиям целостности, поддерживается целостность данных на уровне доменов (типы данных), на уровне отношений и на уровне базы данных в целом;

в) аспект обработки: обеспечивается поддержка манипулирования отношениями (реляционная алгебра, реляционное исчисление). Как правило, способом взаимодействия с базой данных являются запросы на языке Structured Query Language (SQL).

1.3.4 Нереляционные базы данных

В отличие от реляционных баз данных, в которых информация представлена в виде таблиц, нереляционные базы данных предоставляют большую гибкость. Структурный аспект нереляционных баз данных выбирается и оптимизируется под конкретную задачу, что позволяет добиться лучшей производительности. Нереляционные базы данных поддерживают целостность на уровне базы данных, но не поддерживают целостность на уровне доменов. Однако многие нереляционные базы данных поддерживают определенные типы данных. Для обработки данных нереляционные базы данных используют не SQL, а запросы на других языках программирования, хотя многие из них поддерживают запросы, совместимые с SQL. [6]

Выделяют следующие виды нереляционных баз данных:

а) документоориентированные базы данных;

б) базы данных "ключ-значение";

в) колоночные базы данных;

г) графовые базы данных;
д) базы данных временных рядов.

Документо-ориентированные базы данных работают с набором значений именованных строковых полей и данных объекта, называемых документом. Такой документ, как правило, содержит данные одной сущности, которые в реляционной модели данных распределяются по нескольким таблицам. Документо-ориентированные базы данных не накладывают ограничения на структуру документов - наличие одинаковой структуры для всех документов не требуется. Поддержка свободной формы обеспечивает большую гибкость.

Базы данных "ключ-значение"представлют собой хеш-таблицы. Каждое значение сопоставляется с уникальным ключом, который и используется для хранения данных. Как правило, позволяется хранить в качестве значения произвольные данные, однако некоторые базы данных накладывают ограничения. По существу, значения являются большими двоичными объектами. Такие базы данных обычно поддерживают только простейшие операции, такие как вставка и удаление, что не позволяет использовать их для сложных запросов и агрегирования.

Колоночные базы данных занимают промежуточное положение между реляционными и нереляционными базами данных. Ключевым отличием колоночных баз данных является хранение данных отдельных колонок последовательно в памяти. Табличные данные разделяются на определенные группы и семейства столбцов, каждое семейство извлекается и управляется как единое целое. Все столбцы одного семейства хранятся в едином файле. Операции чтения и записи для строк являются атомарными в пределах одного семейства, однако некоторые базы данных поддерживают атомарность для строки, охватывающей несколько семейств.

Графовые базы данных работают с двумя типами данных - узлами и ребрами. Узлы представляют собой сущности, а ребра - связи между нимии. Графовые базы данных позволяют эффективно обрабатывать данные с большим количеством связей.

Базы данных временных рядов работают со значениями, упорядоченными по времени. Данный тип баз данных оптимизирован для операций записи, что позволяет им агрегировать большой объем

информации в режиме реального времени. Как правило, такие данные не обновляются, а удаление происходит массово.

1.4 Фрактальное сжатие данных

Для решения задачи фрактального поиска воспользуемся теорией реляционных баз данных. Введём основные термины. [7]

Домен – некоторое конечное множество данных, обозначается как D_i , где i – номер домена. Отдельный элемент домена обозначется как d_i, где i – также номер домена.

Полное декартово произведение множеств – набор всевозможных сочетаний из n элементов каждое, где каждый элемент берётся из своего домена. Описание: $D_1 \times D_2 \times \cdots \times D_n$.

Отношение R – подмножество декартова произведения множеств D_1, $D_2 \ldots D_n$, необязательно различных. Описание: $R \subseteq D_1 \times D_2 \times \cdots \times D_n$. Число n называется степенью отношения.

Атрибутом называют домен, входящий в отношение. Степень отношения определяет количество атрибутов в отношении.

Схемой отношения S называется перечень имён атрибутов данного отношения с указанием домена, к которому они относятся. Описание: $S_R = (A_1, A_2, \ldots A_n.)$, $A_i \subseteq D_i$.

Каждому имени атрибута A_i ставится в соответствие множество D_i — множество значений атрибута A_i, $1 \leqslant i \leqslant n$, D — объединение D_i, $1 \leqslant i \leqslant n$.

Отношение R со схемой S — это конечное множество отображений t_1, t_2,..., t_n из S в D; причем каждое отображение t должно удовлетворять следующему ограничению: $t(A_i)$ принадлежит D_i, $1 \leqslant i \leqslant n$. Эти отображения называются кортежами.

Из последнего определения видно, что отношение состоит из нескольких частей, кортежей. Именно кортеж несёт в себе информацию о структуре отношения.

Алгоритм фрактального поиска рассматривает именно кортежи. Мы рассматриваем атрибуты кортежа как отдельные части, сохраняя при этом сведения о структуре в целом.

Так как каждый атрибут имеет конечное множество значений, множество значений набора атрибутов будет также конечно. В этом случае можно предположить, что домены в отношении могут иметь идентичную структуру и одинаковые значения.

Принимая это предположение, возможно выделить домены в отношении, определить взаимосвязь атрибутов и фрактально сжать таблицу, сохранив при этом её структуру. Фрактальное сжатие поможет не только более компактно хранить имеющиеся данные, но и обнаружить в данных нетривиальные корреляции, полезные для дальнейшего анализа данных.

Основная идея алгоритма фрактального поиска заключается в подборе такого множества доменов, которое не только полностью опишет структуру отношения, но и позволит представить данные в базе минимальным количеством записей.

1.5 Оптимизация фрактального сжатия

Пусть в нашей таблице имеется n атрибутов и в каждый домен входит k атрибутов, 1⩽k⩽n.

Количество различных доменов из k атрибутов: $C_n^k = \frac{(n!)}{(k!(n-k)!)}$

Количество всех возможных различных доменов: $\sum_{k=1}^{n} C_n^k = \sum_{k=1}^{n} \frac{(n!)}{(k!(n-k)!)}$

Как видно из графика, с ростом количества атрибутов резко возрастает количество всех возможных доменов. Как правило, при формировании отношения число входящих в него атрибутов стараются ограничивать 15, что дает нам количество возможных структур не более 32767, однако при проектировании схемы базы данных могут и не придерживаться этого правила. Сложность во многом будет зависеть именно от количества атрибутов в таблице.

Одной из важнейших характеристик домена является количество различных значений в нём. Чем меньше различных значений в домене, тем больше информации о структуре базы данных он отображает. Количество различных значений домена можно определить как произведение количества различных значений атрибутов, входяших в него, но на практике это число оказывается меньше. Поиск количества различных значений домена является самой затратной частью алгоритма фрактального поиска, поэтому имеет смысл ограничить размер доменов, чтобы сократить время выполнения данной части.

После того, как будет получено количество различных значений для каждого из множества доменов, необходимо получить оптимальное множество доменов. Оптимальное множество доменов полностью описывает таблицу и определяется как $D = D_1 + \cdots + D_m$.

Оптимальное множество доменов должно удовлетворять следующим требованиям:

а) все атрибуты должны быть в единственном экземпляре;

б) сумма количества различных значений доменов должна быть минимальна.

Алгоритм поиска оптимального множества является рекурсивной функцией: последовательно добавляются новые домены до тех пор, пока не будут просмотрены все атрибуты или пока количество различных значений

доменов не превышает текущего минимума. При первом запуске минимум определяется как произведение количества атрибутов на количество строк, т.е. каждая ячейка считается уникальным значением.

1.6 Стратегии выбора атрибутов для сжатия

Таккак с ростом количества атрибутов возрастает сложность алгоритма, необходимо проводить предобработку атрибутов. На этом этапе атрибуты разделяются на потенциально сжимаемые и несжимаемые. Последняя группа в дальнейшем алгоритме не используется, что позволяет снизить количество атрибутов.

Для каждого атрибута вводятся первичные параметры:

а) атрибут является первичным ключом; $PK = \begin{cases} 1, pkeyconstrain = true \\ 0, else \end{cases}$

б) атрибут является уникальным ключом; $K = \begin{cases} 1, keyconstrain = true \\ 0, else \end{cases}$

в) атрибут является непересекаемым ключом; $E = \begin{cases} 1, exclconstrain = true \\ 0, else \end{cases}$

г) атрибут является последовательностью; $S = \begin{cases} 1, seqconstrain = true \\ 0, else \end{cases}$

Сжимаемость атрибута определяется согласно следующей формуле: $Compressable = \neg(PK \vee K \vee E \vee S)$

Для каждого атрибута вводятся так же вторичные параметры:

а) фактор уникальности; $UniqueFactor = \frac{Unique\ Values}{Values}$

б) приоритет сжатия. $Priority = max(V)$

Приоритет сжатия может быть получен двумя способами:

а) последовательным проходом по каждому элементу с подсчётом его реального значения и сохранением максимума;

б) из ограничения размера, определяемого базой данных для каждого типа данных.

Атрибуты с ненулевой сжимаемостью и фактором униальности ниже порогового значения не вносятся в множество сжимаемых атрибутов и в дальнейшем не используются для сжатия.

1.7 Генетический алгоритм

Для решения задачи выбора наилучшего домена в условиях ограничения по времени можно воспользоваться генетическими алгоритмами. [8]

В качестве особи a_k^t, где k - номер особи, а t - номер поколения, представляется домен. Качественными признаками особи являются её кодировка s(x) и составляющие её компоненты $s_i(b_i)$, $i \in [1, n]$, n - количество атрибутов. В качестве гена примимается $s_i(b_i)$, i-тый ген принимает значение 0, если i-ый атрибут отсутствует в домене, и 1 иначе. Каждая особь характеризуется n генами, следующими друг за другом, кодировку s(x) можно интерпретировать как хромосому a_k^t и записывать как c_k^t. Генотип отображает, какие из атрибутов входят в домен, который представляет особь, фенотип отображает количество уникальных кортежей данного кортежа.

Целью эволюционного развития особей является определение такого генотипа, принадлежащего генофонду, который обеспечивает наибольшую приспособленность к внешней среде. Степень приспособленности особи a_k^t - $m(a_k^t)$ - определяется как функция от количества уникальных значений и количества атрибутов, входящих в домен. Чтобы не допустить вырождения фрактального сжатия в словарное кодирование, функция приспособленности для особей, содержащих одно ненулевое значение гена, приравнивается к нулю.

Так как кодировка особи является бинарной строкой, для определения близости особей можно использовать Хэммингово расстояние: $d(c_i, c_j) = \sum_{r=1}^{n} C_i \bigoplus C_j$

Обобщенно алгоритм может быть представлен следующим образом: [9]

а) Инициализация начальной популяции P^0

1) Генерация N различных случайных чисел в диапазоне $[2, 2^n]$, n - количество атрибутов, отобранных для сжатия, $N < 2^n$. Бинарное представление i-ого числа принимается за кодировку i-ой особи.

2) Оценка приспособленности каждой из особей.

б) Воспроизводство потомков с наследственными генетическими свойствами родителей

1) Выбрать случайным образом из текущей популяции P^t согласно схеме скрещивания кодировки двух родителей, образующих «брачную пару».

2) Оценка приспособленности каждой из особей.

в) Создание мутантов с генетическими свойствами, отличающимися от свойств родителей

1) Сгенерировать при помощи оператора мутации для выбранной кодировки с вероятностью p кодировку-мутанта, обеспечивающую изменчивость генетических свойств родителей.

2) Повторять пункт 3 до заданного числа мутантов

г) Оценка кодировки особей с помощи функции приспособленности Так как оценка функции приспособленности является затратной операцией в связи с необходимостью подсчитывать количество уникальных значений для каждого отдельно взятого случая, то оценку целесообразно проводить один раз для всех особей

д) Замена текущей популяции P^t новой популяцией $P^(1+t)$.

1) Выбрать стратегию формирования популяции $P^(1+t)$

2) Скопировать при помощи оператора селекции из репродукционного множества кодировки, реализующие стратегию формирования популяции

е) Проверка условия выхода из цикла

1.7.1 Системы скрещивания

Рассмотрим системы скрещивания: [15]

— Панмиксия – две любые особи имеют одинаковую возможность образовать пару.

— Инбридинг – две кодировки имеют возможность образовать пару только при условии, что они генетически похожи.

— Аутбридинг — две кодировки имеют возможность образовать пару только при условии, что они генетически не похожи.

— Ассоциативное скрещивание — скрещивание на основе функции приспособленности.

— Положительное ассоциативное скрещивание – скрещивание особей, имеющих близкое значение функции приспособленности.

— Отрицательное ассоциативное скрещивание – скрещивание особей, имеющих различающееся значение функции приспособленности.

— Селективное скрещивание – система скрещивания, в которой при образовании «брачной пары» осуществляется предварительное отстранение некоторых кодировок от участия в этом процессе.

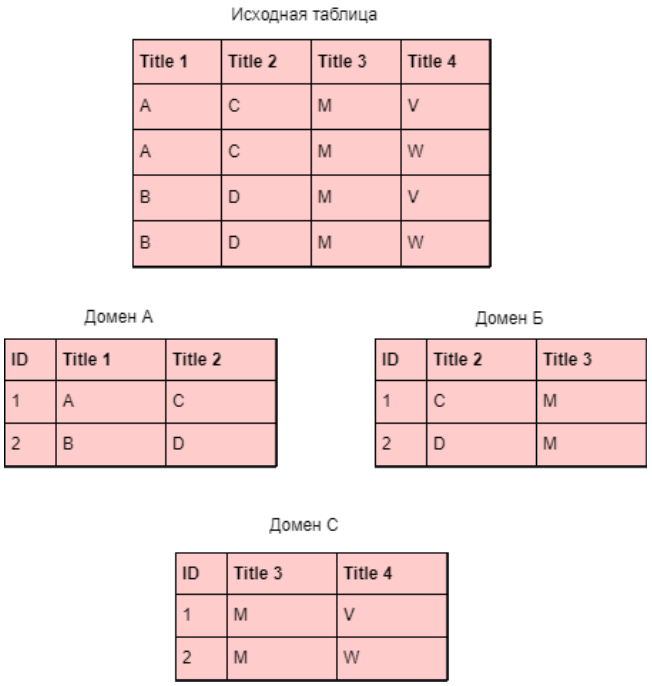

Рисунок 1.13 — Возможные домены размера 2

Системы скрещивания, основанные на функции приспособленности, не подходят для данной системы, так как домены, имеющие разный генотип, могут иметь одинаковую или схожую оценку приспособленности, однако их потомки будут иметь худшую оценку приспособленности.

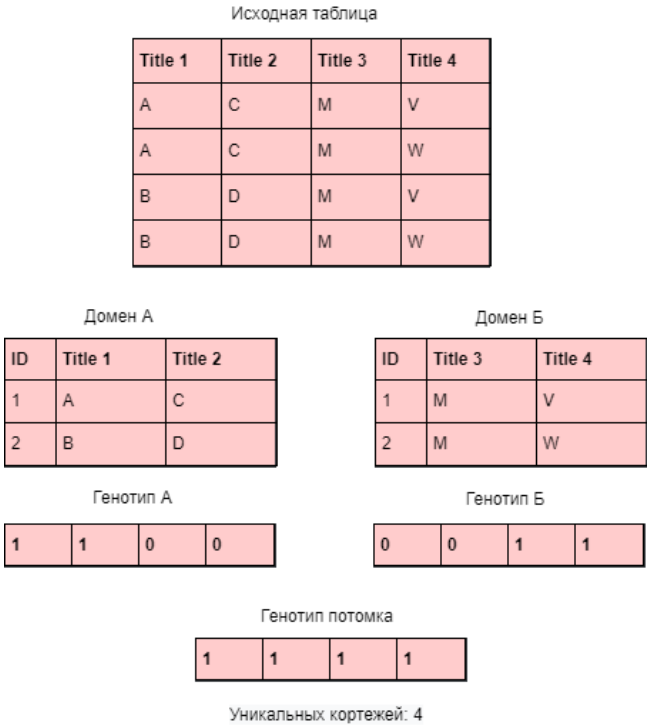

Рисунок 1.14 — Аутбридинг

Аналогично, для данной системы не целесообразно использовать аутбридинг.

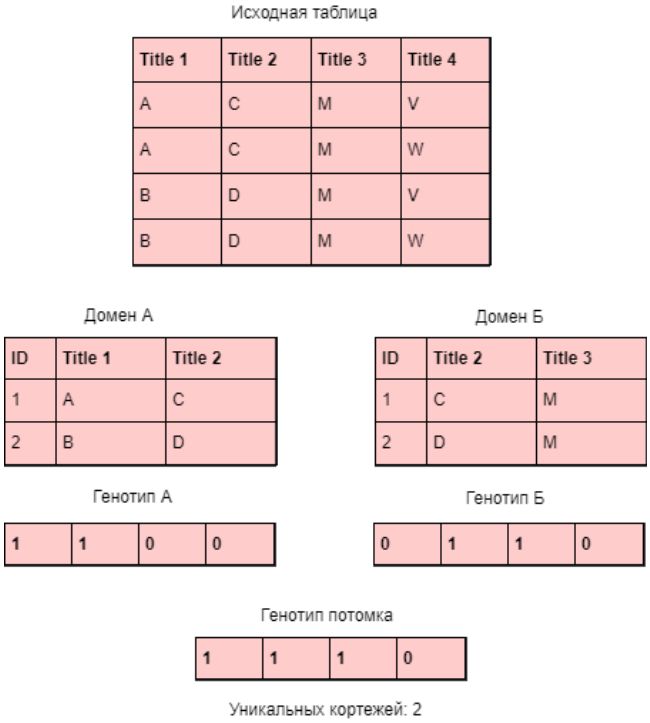

Рисунок 1.15 — Инбридинг

Для данного кейса стоит использовать селективное скрещивание и отстранять от участия в дальнейшем размножении особей, у которых доля уникальных значений от общего числа значений составляет больше половины, так как в этом случае не достигается достаточное сжатие. [13]

1.7.2 Скрещивание особей

Скрещивание особей в генетическом алгоритме происходит через оператор скрещивания.

Обобщенно алгоритм скрещивания может быть представлен следующим образом:

— генерация случайного числа или чисел;
— копирование родительских кодировок;
— разбиение родительских кодировок на куски;
— составление кодировок потомков из кусков кодировок родителей.

Классические операторы скрещивания: [10]

— универсальный кроссовер - каждый бит родительской кодировки имеет равные шансы быть выбранным и скопированным в кодировку потомка;

— одноточечный кроссовер - кодировки родителей разрываются в одной и той же точке и из полученных кусков формируются кодировки потомков;

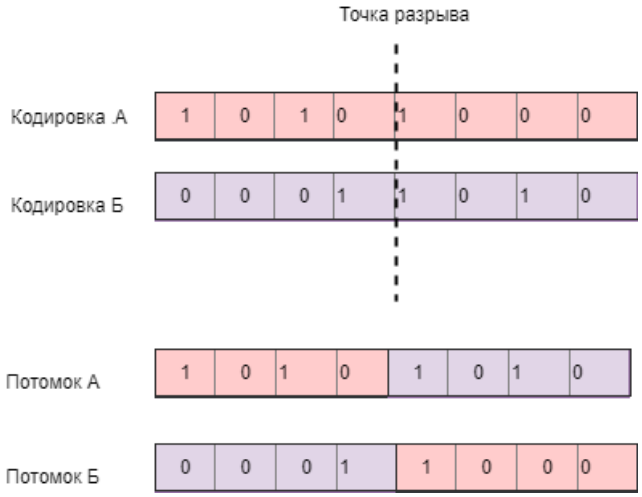

Рисунок 1.16 — одноточечный кроссовер

— ограниченный одноточечный кроссовер - аналогично одноточечному кроссоверу, но точка разрыва выбирается из диапазона, где родительские кодировки не совпадают;

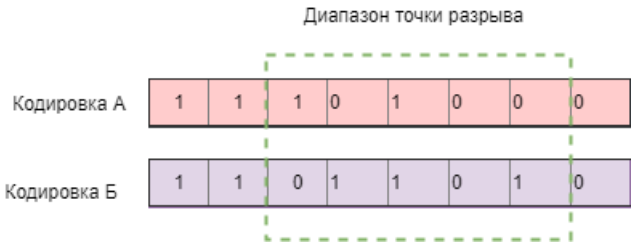

Рисунок 1.17 — ограниченный одноточечный кроссовер

— двуточечный кроссовер - кодировки родителей разрываются в двух точках в одних и тех же местах е и из полученных кусков формируются кодировки потомков;

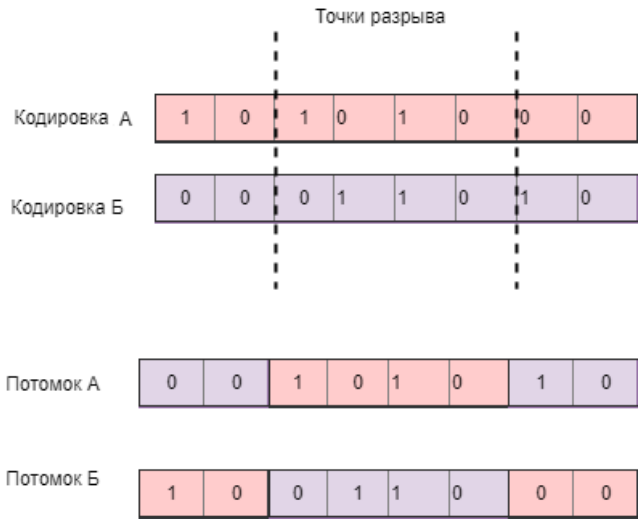

Рисунок 1.18 — двуточечный кроссовер

— ограниченный двуточечный кроссовер - аналогично ограниченному одноточечному кроссоверу;

— многоточечный кроссовер;

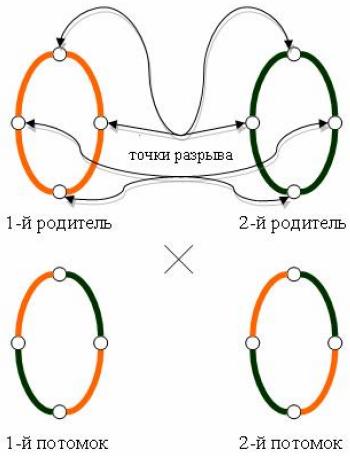

Рисунок 1.19 — многоточечный кроссовер

— триадный кроссовер - помимо двух родительских особей выбирается ещё одна особь, чья кодировка - преобразованная или нет - служит маской;

— перетасовочный кроссовер - аналогично предыдущим операторам, однако на каждом этапе изменяются также и родительские особи, что позволяет сократить число операций.

1.7.3 Мутация особей

Так как оператор кроссовера не вносит в популяцию новую генетическую информацию, для создания генетического разнообразия

применяется оператор мутации. Оператор мутации необходим для препятсвования преждевременной сходимости к локальному эксремуму. [11]

Классические операторы мутации:

— точечная мутация;

Рисунок 1.20 — точечная мутация

— дополнение;

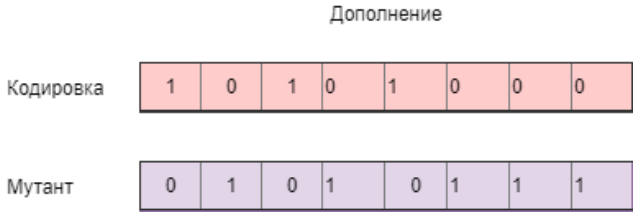

Рисунок 1.21 — дополнение

— инверсия;

Рисунок 1.22 — инверсия

— сальтация;

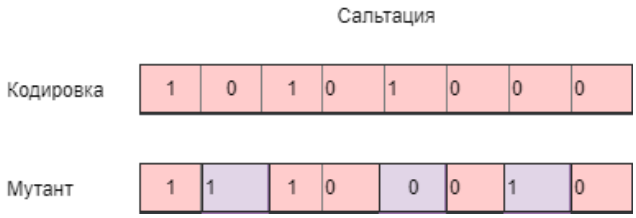

Рисунок 1.23 — сальтация

— транслокация.

Рисунок 1.24 — транслокация

1.7.4 Выбор нового поколения

Для отбора особей в новое поколение существует несколько стратегий: [12]

— отбор усечением:

а) все особи сортируются по значению функции приспособленности;

б) задается пороговое число T, определяющее какая часть популяции будет рассматриваться для дальнейшего отбора;

в) среди учесенной популяции случайным образом выбирается необходимое количество особей.

— элитарный отбор;

а) создается промежуточная популяция, состоящая как из особей-родителей, так и из особей-потомков и особей-мутантов;

б) все особи сортируются по значению функции приспособленности;

в) отбирается N самых лучших особей и заносятся в новое поколение;

г) из оставшихся особей случайным образом выбираются особи до размера поколения.

— отбор вытеснением;

а) создается промежуточная популяция, состоящая как из особей-родителей, так и из особей-потомков и особей-мутантов;

б) все особи сортируются по значению функции приспособленности;

в) выбирается особь и оценивается её сходство с имеющимися в новой популяции особями;

г) при наличии двух особей с одинаковым значением функции приспособленности выбирается особь, чей генотип ещё не присутствует в поколении.

— отбор отжигом. [14]

а) задается новый управляющий параметр - температура T;

б) оценивается вероятность попадания особи в новое поколение; Вероятность попадания в новую популяцию: $p = \frac{1}{1+e^{\frac{m_i - m_j}{T}}}$

Первым поколениями соответствует большее значение температуры, однако с каждым поколением температура снижается и выбираются особи с наименьшим значением функции приспособленности.
в) если значение p оказывается больше случайного числа x, $x \in [0, 1]$, выбирается особь a_i, иначе особь a_j.

2 Конструкторский раздел

2.1 Архитектура программного продукта

Разрабатываемый метод можно представить с помощью функциональной схемы IDEF0. Основной задачей является сжатие таблицы реляционной базы данных. В качестве входного параметра выступает конфигурационный файл.

Разрабатываемый программный продукт состоит из следующих частей:

а) модуль работы с базой данных и конфигурациями;
б) модуль фрактального сжатия данных;
в) модуль генетического алгоритма для выбора оптимального домена.

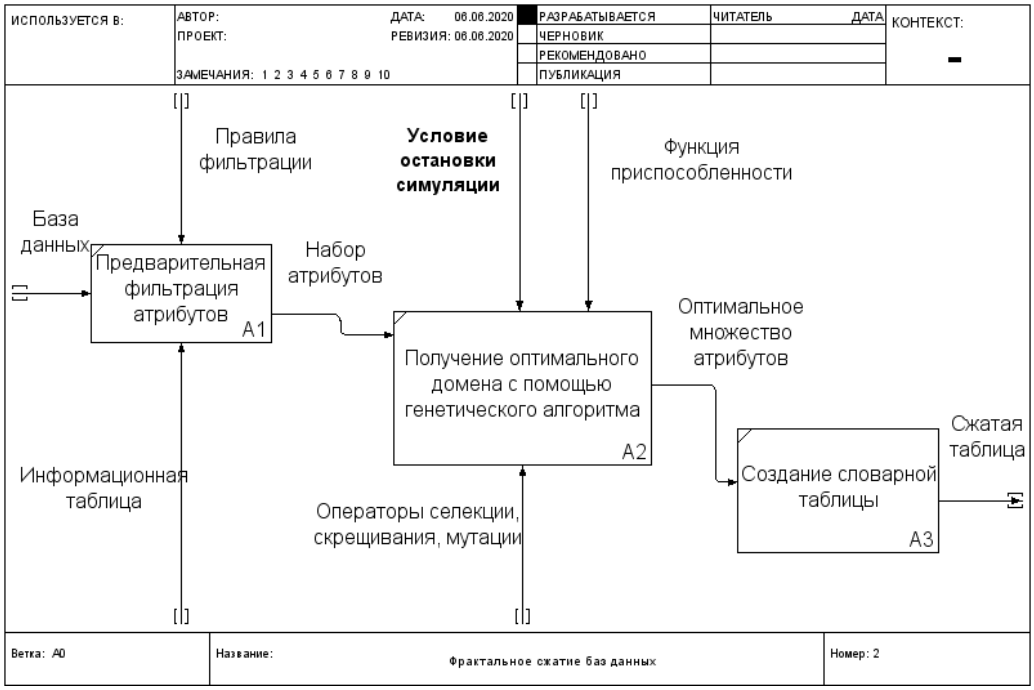

Рисунок 2.1 — IDEF0-диаграмма программного комплекса

2.2 Определение множества атрибутов для дальнейшего сжатия

Так как с ростом количества атрибутов возрастает сложность алгоритма, на первом этапе работы фрактального сжатия проводится первичная фильтрация атрибутов на сжимаемые и несжимаемые. Данный алгоритм описывает процесс фильтрации исходного множества атрибутов.

Входные данные: множество атрибутов, минимальный коэффициент сжатия.
Выходные данные: множество отфильтрованных для дальнейшего сжатия атрибутов.

На первом этапе алгоритма происходит обращение к информационной таблице, хранящей метаданные для атрибутов. Из этой таблицы для каждого атрибута определяется наличие или отсутствие ограничений, тип данных и максимальный размер для данного типа данных.

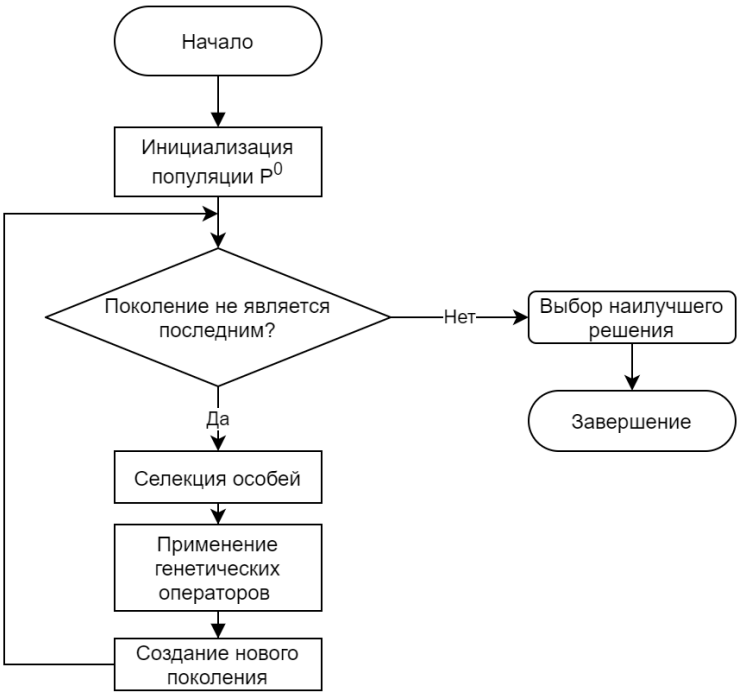

Рисунок 2.2 — Схема определения сжимаемости атрибута

2.3 Генетический алгоритм

Для определения оптимального домена для сжатия используется генетический алгоритм. Генетический алгоритм позволяет решить проблему решения экстремальной комбинаторной задачи переборного типа. Генетический алгоритм не гарантирует нахождение глобально самого оптимального решения, однако алгоритм позволяет найти оптимальное решение за меньшее время, чем остальные алгоритмы поисковой оптимизации.

Входные данные алгоритма: множество атрибутов, параметры генетического алгоритма.
Выходные данные: оптимальный домен.

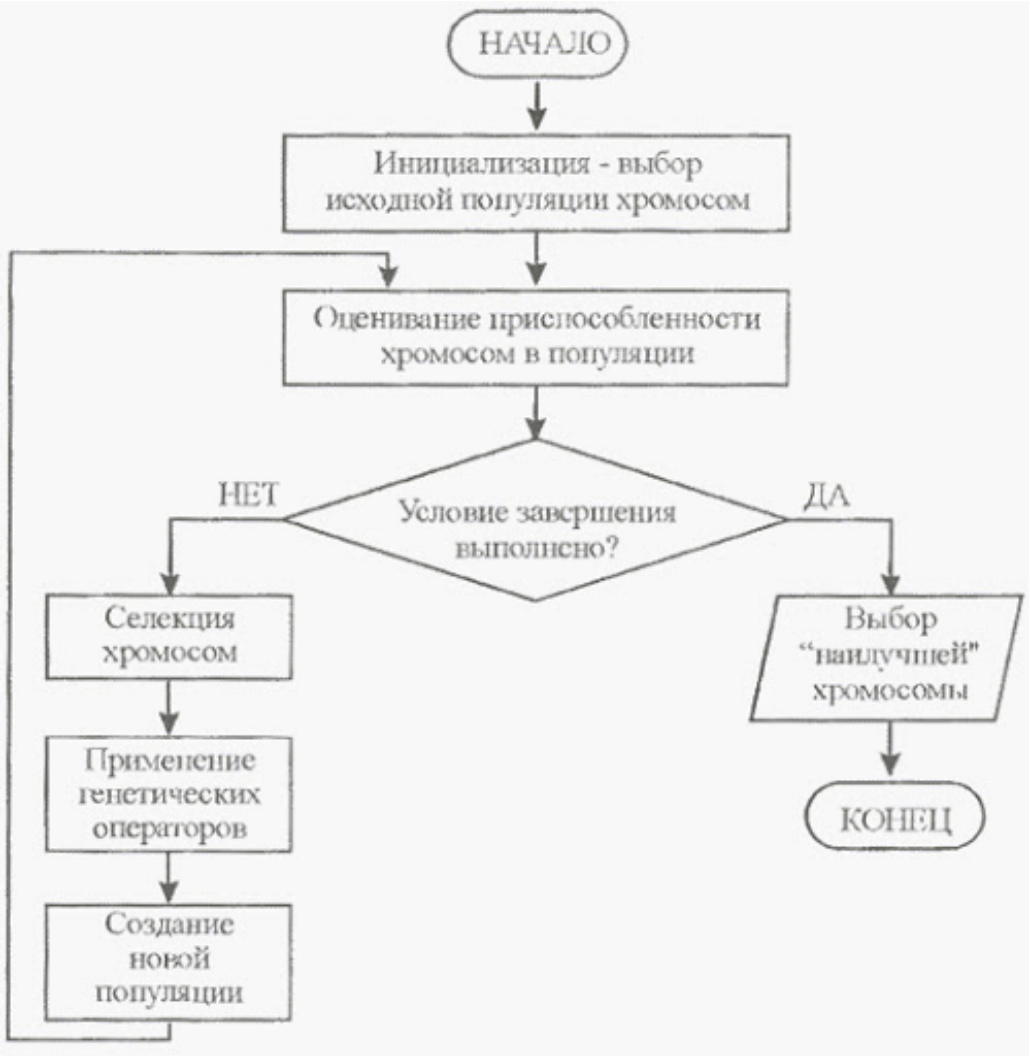

Рисунок 2.3 — Общая схема алгоритма

Характеристиками генетического алгоритма являются:

а) численность популяции;

б) число брачных пар;
в) число мутантов;
г) вероятность скрещивания;
д) вероятность мутации;
е) число поколений.

2.3.1 Создание особи

Генетический алгоритм формирует особь с помощью алгоритма создания особи. Создание особи происходит в два этапа. На первом этапе алгоритм получает случайную величину X, $X\tilde{U}ni(2,2^N - 1)$. Равномерное распределение гарантирует равную вероятность для каждого домена быть выбранным в начальное поколение P^0. На втором для особи a_i^0 получается её кодировка c_i^0 с помощью представления случайной величины X в бинарном виде.

Входные данные: N - число атрибутов, участвующих в кодировании. Выходные данные: кодировка c_i^0, представляющая собой двоичное представление кодируемого ею домена.

2.3.2 Алгоритм получения значения функции приспособленности особи

Одним из ключевых моментов генетического алгоритма является получение значения функции приспособленности. Функция приспособленности определяет, насколько близко является полученное значение к цели алгоритма. Именно функция приспособленности направляет работу генетического алгоритма в сторону оптимального решения. В данном алгоритме функцией приспособленности является функция отношения количества всех кортежей на количество уникальных кортежей с учетом размера домена. Для доменов, содержащих в себе один атрибут, значение функции приспособленности является нулем. Для доменов, содержащих в себе большее количество уникальных кортежей, чем заданное алгоритмом пороговое число, значение функции приспособленности также является нулем.

Входные данные алгоритма: кодировка c_i^t.

Выходные данные: значение функции приспособленности m_i^t особи a_i^t. Внутренними характеристиками является пороговое число K, характеризующее минимальную степень сжатия таблицы.

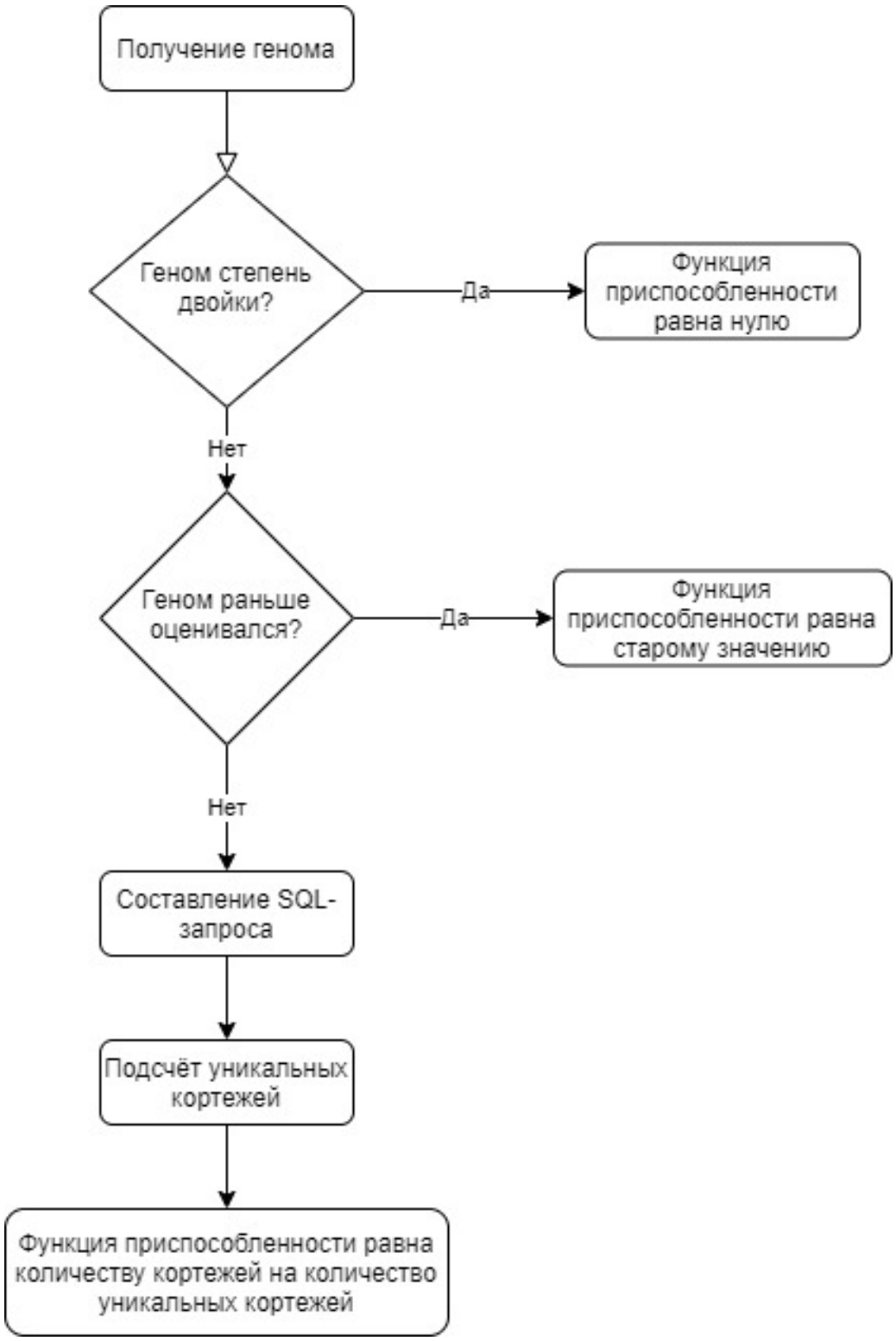

Рисунок 2.4 — Схема получения оценки приспособленности особи

2.3.3 Селекция особей

Генетический алгоритм формирует родительскую пару на основе оператора скрещивания. Для данной работы была выбрана селективная система скрещивания. Эта система является расширением универсальной системы скрещивания, где каждые особи a_i^t и a_j^t имеют равную возможность

образовать брачную пару. В данной системе скрещивания к дальнейшему скрещиванию допускаются только те особи, чьё значение приспособленности не меньше среднего значения приспособленности популяции. Такой подход позволяет более быструю сходимость генетического алгоритма по сравнению с универсальной системой скрещивания.

2.3.4 Стратегия формирования нового поколения

Не менее важным компонентом генетического алгоритма является стратегия замены поколения P^t на поколение $P^t + 1$. Для данной работы была выбрана элитная поколенческая схема. На первом этапе из поколения P^t выбирается особь с максимальным значением функции приспособленности, после чего из множества потомков и мутантов выбирается с помощью оператора селекции v наиболее приспособленных особей, v+1 = N, где N - размер популяции.

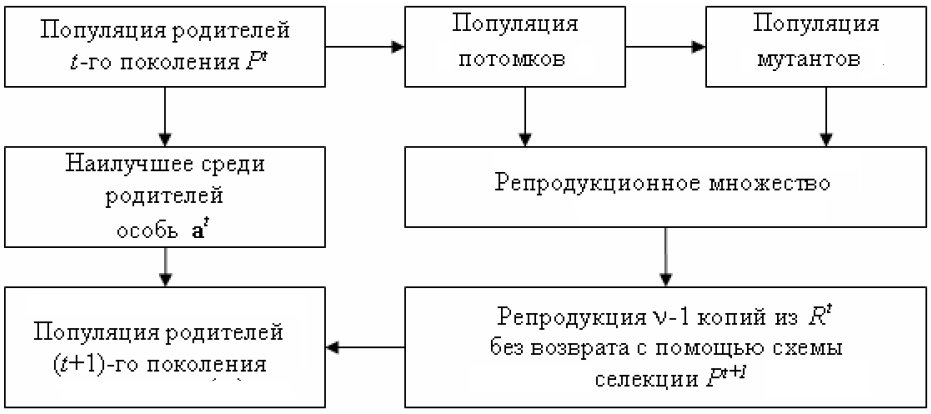

Рисунок 2.5 — Схема поколения

Селекция особей происходит в несколько этапов. Для селекции особей в данной работе используется алгоритм стохастического выбора с возвращением, также называемый схемой выбора с помощью рулеточного колеса. В данной схеме каждой из особей a_i^t назначается сектор рулеточного колеса, величина которого пропорциональна значению функции приспособленности. Алгоритм разбит на следующие этапы:

а) вычисление суммы значений функции приспособленности для данного поколения; $F = \sum_{i=1}^{N} m_i$

б) вычисление вероятности выбора для каждой особи в полокении; $p(a_i^t) = \frac{m_i}{F}$

в) генерируется по равномерному закону распределения число x;

г) в новое поколение $P^t + 1$ копируется особь, для которой выполняется условие $p_i < x \leqslant p_i + 1$

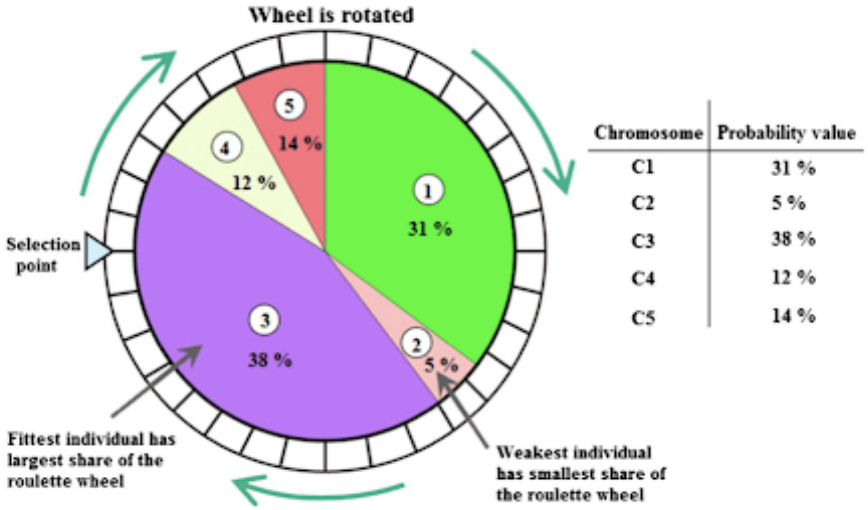

Рисунок 2.6 — Схема поколения

2.3.5 Скрещивание особи

Для обмена генами между особями-родителями используется оператор кроссовера. Операор кроссовера применяется к кодировкам c' и c'' с вероятностью p_c, являющейся скоростью кроссовера и характеризующей вероятность того, что родительские кодировки образуют новые кодировки. Если оператор кроссовера не срабатывает, кодировки-потомки являются точными копиями кодировок-родителей. В противном случае кодировки-потомки являются комбинацией кодировок-родителей.

Для данной работы в качестве оператора скрещивания был выбран одноточечный кроссовер. Алгоритм может быть представлен в следующем виде:

а) согласно схеме селекции выбираются родительские кодировки c' и c'';

б) с помощью генератора случайных чисел выбирается число r, $r \in [1, n]$, n - количество атрибутов;

в) кодировки особей-родителей разрываются в точке r, образовывая 4 участка кодировок;

$(c'_1 ... c'_r)(c'_r ... c'_n)$

$(c''_1 ... c''_r)(c''_r ... c''_n)$

г) из получернных частичных кодировок формируются кодировки особей-потомков;

$$c^1 = (c_1^{`}...c_r^{`})(c_r^{``}...c_n^{``})$$

$$c^2 = (c^{``}{}_1...c^{``}{}_r(c_r^{`}...c_n^{`})$$

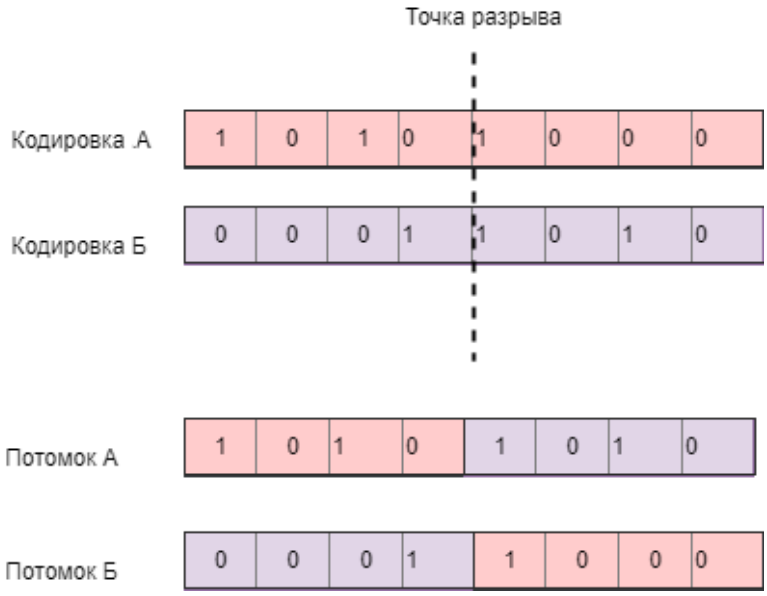

Рисунок 2.7 — Одноточечный кроссовер

2.3.6 Мутации особи

Так как оператор кроссовера не вносит в популяцию новую генетическую информацию, для обеспечения генетической изменчивности используется оператор мутации. Оператор мутачии изменяет значение кодировки в одном или нескольких генах. Мутации являются стохастическими и не зависят от кодировки особи или от значения функции её приспособленности. Для данной работы был выбран оператор точечной мутаци. Операор кроссовера применяется к кодировкам c' и c'' с вероятностью p_m, являющейся скоростью мутации и характеризующей вероятность того, что родительская кодировка образует потомка-мутанта.

Алгоритм мутации:

а) с помощью генератора случайных чисел выбирается число i, $i \in [1, n]$, n - количество атрибутов;

б) i-тый бит кодировки особи инверсируется.

Именно оператор мутации обеспечвает генетическое разнообразие в популяции.

Рисунок 2.8 — Схема мутации

3 Технологический раздел

3.1 Выбор средств разработки

В качестве языка программирования выбран язык Golang. Golang является компилируемым многопоточным языком программирования и был разработан для разработки высокоэффективных программ для распределенных систем. Язык обладает минималистичным синтаксисом и не содержит сложных и замысловатых конструкций, характерных для таких языков как C++ и Java. Одним из главных плюсов языка являются горутины - легковесные процессы. Язык поддерживает такие парадигмы программирования, как процедурное, объектно-ориентированное, функциональное и параллельное программирование.

3.2 Выбор среды разработки

Средой разработки программного комплекса была выбрана IDE Golang компании JetBrains, специализирующейся на производстве инструментов для профессиональной разработки программного обеспечения.

Данная среда выбрана из-за следующих удобств и преимуществ:

а) наличие статического анализатора;

б) возможность генерации методов по интерфейсу;

в) наличие полноценного отладчика как для самого кода программы, так и для тестов;

г) наличие поддержки инструментов go, таких как go test, go vet и go;

д) интеграция с системой контроля версий (VCS) Git;

е) начилие встроенного терминала (promt, powershell или bash);

ж) возможность создания и настройка нескольких конфигураций запуска;

з) поддержка Docker.

3.3 Выбор используемых библиотек

Для данной работы главной используемой библиотекой является

а) pgx - для работы с базой данных;
б) viper - для работы с конфигурациями;
в) goga - для работы с генетическим алгоритмом.

Библиотека pgx является драйвером языка Go для работы исключительно с PostgreSQL. Основными плюсами данной библиотеки является её быстрота и производительность, достигнутая засчёт применения низкоуровневых операций, поддержка более 70-типов, характерных для Postgres и реализованных вне стандарта SQL, наличие человекопонятных ошибок, возможность конфигурировать соединения с базой данных, поддержка логирования "из коробки поддержка автоматически подготовленных запросов.

Библиотека viper является библиотекой для конфигурирования приложений на языке Golang. Библиотека позволяет задавать дефолтные значения для конфигурируемых параметроы, получать параметры конфигурации из файлов формата JSON, YAML, DOTENV, TOML, HCL, изменение параметров конфигурирования во время исполнения программы, чтение параметров конфигурации из флагов командной строки или из буффера.

Библиотека goga является библиотекой, реализующей общий подход к генетическим алгоритмам. Библиотека позволяет реализовать собственные вариации генетических алгоритмов путем передачи обощенному алгоритму специфичных компонент. Такими компонентами являются компоненты симуляции, селекции и скрещивания. Компонента симуляции позволяет специфицировать работу с конкретным геномом и по сути является реализацией функции приспособленности. Компонента селекции является реализацией выбора нового поколения из текущего на основе заданных правил. Компонента скрещивания реализацией выбора геномов-родителей и создания нового генома на основе выбранного подхода. Библиотека содержит в себе реализации распространненых подходов и позволяет облегчить разработку генетических алгоритмов на языке Go.

3.4 Система контроля версий

Во время процесса разработки мною была использованна система контроля версий Git. Система контроля версий с помощью репозиториев решает проблемы с переносом программного кода на другие устройства, его резервным копированием, а так же дает возможность разделения версий продукта во время разработки, что позволяет при внесении изменений или модифакциях всегда иметь рабочию версию проекта.

Из всех систем была выбрана именно Git по следующим причинам:

а) удобная система ветвения;

б) интеграция с выбранной средой разработки;

в) наличие большого количества платформ, поддерживающих репозитории Git.

3.5 Требования к компьютеру

Разрабатываемая система имеет ряд требований к компьютеру пользователя, в том числе:

а) язык программирования Golang, версия не ниже 1.13.
б) СУБД Postgres
в) библиотеки языка Golang, указанные в файле go.mod.

Файл go.mod хранит в себе используемые программным комплексом пакеты с указанием версий пакетов.

```
1
2  require (
3  github.com/cockroachdb/apd v1.1.0
4  github.com/gofrs/uuid v3.2.0+incompatible
5  github.com/gonum/floats v0.0.0-20181209220543
6  github.com/gonum/internal v0.0.0-20181124074243
7  github.com/gonum/stat v0.0.0-20181125101827
8  github.com/jackc/fake v0.0.0-20150926172116-812a484cc733
9  github.com/jackc/pgx v3.6.2+incompatible
10 github.com/lib/pq v1.3.0
11 github.com/orcaman/concurrent-map v0.0.0-20190826125027
12 github.com/pkg/errors v0.9.1
13 github.com/shopspring/decimal v0.0.0-20200419172439
14 github.com/spf13/viper v1.6.3
15 github.com/tomcraven/goga v0.0.0-20160417173515
16 golang.org/x/crypto v0.0.0-20200414173820
17 golang.org/x/text v0.3.2
18 )
```

Рисунок 3.1 — Файл go.mod для реализованного программного комплекса

3.6 Структура программного обеспечения

3.6.1 Интерфейсы

Особенностью языка Go является наличие в нем специального типа данных, интерфейса. Интерфейсы являются именованными коллекциями сигнатур методов и являются абстракцинй поведения других типов данных. Интерфейсы определяют функционал, но не реализуют его. Реализация интерфейса в языке Go происходит неявно: для этого достаточно реализовать методы, указанные в интерфейсе. Так как интерфейс является типом данных, можно создавать переменные данного типа. Такой переменной возможно присвоить переменную любого типа, реализовывающего методы, соответствующие сигнарутам интерфейса.

Используемые в данном программном комплексе интерфейсы:

а) интерфейс для базы данных (источника);

Данный интерфейс определяет следующие функции:

1) служебную функцию Open(), функцию создания соединения с базой данных;

2) служебную функцию Ping(), функцию проверки соединения с базой данных;

3) служебную функцию Close(), функцию закрытия соединения с базой данных;

4) функцию GetMeta(), функцию получения метаданных из информационной таблицы базы данных;

5) функцию GetConstraint(), функцию получения ограничения из информационной таблицы базы данных;

6) функцию GetValues(), функцию получения количества значений атрибута;

7) функцию GetUniqueValues(), функцию получения количества уникаольных значений атрибута;

8) функцию GetPriority(), функцию получения приоритета атрибута;

9) функцию GetFitness(), функцию получения значения функции приспособленности домена;

10) функцию Compress(), функцию сжатия данных в новую базу данных.

```
type InputDB interface {
    Open() error
    Ping() error
    Close() error

    GetMeta(name string) ([]string, []string, error)
    GetConstraint(table, col string) ([]string, error)
    GetValues(name string) (uint, error)
    GetUniqueValues(name, name2 string) (uint, error)
    GetPriority(name, name2 string) (uint, error)
    GetFitness(names []string) int
    Compress(DB OutputDB)
}
```

Рисунок 3.2 — Интерфейс сжимаемой базы данных

б) интерфейс для базы данных (цели);
Данный интерфейс определяет следующие функции:

1) служебную функцию Open(), функцию создания соединения с базой данных;
2) служебную функцию Ping(), функцию проверки соединения с базой данных;
3) служебную функцию Close(), функцию закрытия соединения с базой данных;
4) служебную функцию CreateTables(), функцию создания таблиц базы данных;
5) служебную функцию InsertCompressed(), функцию заполнения базы данных.

```
type OutputDB interface {
    Open() error
    Ping() error
    Close() error
    Write() error
}
```

Рисунок 3.3 — Интерфейс целевой базы данных

в) интерфейс для симулятора; Данный интерфейс определяет следующие функции:

1) OnBeginSimulation, функция, выполняемая перед симуляцией;
2) Simulate, функция симуляции и полчения оценки приспособленности для генома;

3) OnEndSimulation, функция, выполняемая после симуляции;

4) ExitFunc, функция, определяющая условия выхода из цикла симуляции.

```
type ISimulator interface {
    OnBeginSimulation()
    Simulate(*IGenome)
    OnEndSimulation()
    ExitFunc(*IGenome) bool
}
```

Рисунок 3.4 — Интерфейс симулятора

г) интерфейс для битсета Данный интерфейс определяет следующие функции:

1) Go, функция, создающая кодировку особи с учетом поставленной задачи.

```
type IBitsetCreate interface {
    Go() Bitset
}
```

Рисунок 3.5 — Интерфейс битсета

д) интерфейс для генома
Данный интерфейс определяет следующие функции:

1) GetFitness, функция, возвращающая значение функции приспособленности особи;

2) SetFitness, функция, устанавливающая значение функции приспособленности особи;

3) GetBits, функция, возвращающая кодировку особи.

```
type IGenome interface {
    GetFitness() int
    SetFitness(int)
    GetBits() *Bitset
}
```

Рисунок 3.6 — Интерфейс генома

3.6.2 Реализации

Используемые в системе классы:

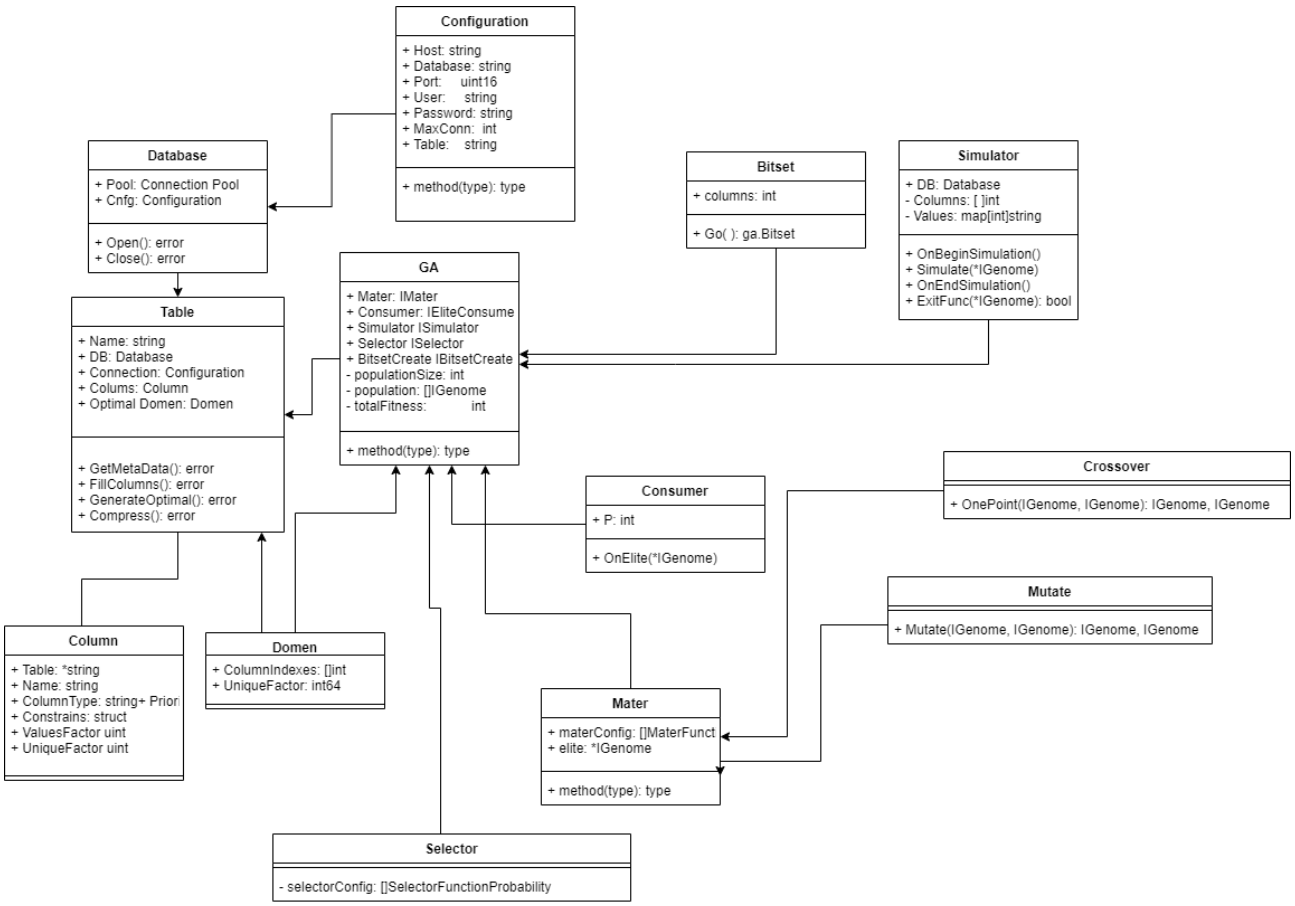

Рисунок 3.7 — Диаграмма классов

Класс таблицы является основным классом разработанного программного комплекса и содержит в себе информацию о сжимаемой таблице:

а) K - пороговый коэффициент сжимаемой базы данных;

б) Name - имя сжимаемой таблицы;

в) DB - ссылка на класс сжимаемой базы данных;

г) Columns - массив ссылок на структуры, представляющие атрибуты;

д) Compressible - массив ссылок на структуры, представляющие атрибуты, отобранные для дальнейшего сжатия;

е) OptimalDomen - массив ссылок на структуры, представляющие собой атрибуты, вошедшие в оптимальный домен.

```
type Table struct {
    K                int
    Name             string
    DB               database.InputDB
    ConnectionConfig interface{}
    Columns          []*Column
    Compressible     []*Column
    Domens           Collection
    OptimalDomens    OptimalCollection
}
```

Рисунок 3.8 — Класс таблицы

Класс базы данных содержит в себе информацию о сжимаемой базе данных:

а) Config - структура, содержащая информацию о параметрах подключения к базе данных;

б) Pool - пул соединений, выделенных базой данных.

```
type PgxDB struct {
    Config  cf.Config
    Pool    *pgx.ConnPool
}
```

Рисунок 3.9 — Класс базы данных

Класс конфигурации содержит в себе информацию о подключении к базе данных:

а) Host - адрес сервера;
б) Database - имя базы данных;
в) Port - порт для подключения к серверу;
г) User - имя пользователя для подключения к базе данных;
д) Password - пароль ждя подключения к базе данных;
е) MaxConn - максимальное количество соединений с базой данных;
ж) Table - имя сжимаемой таблицы.

```
type Config struct {
    Host     string `json:"input.host"`
    Database string `json:"input.database"`
    Port     uint16 `json:"input.port"`
    User     string `json:"input.user"`
    Password string `json:"input.password"`
    MaxConn  int    `json:"input.max_conn"`
    Table    string `json:"input.table"`
}
```

Рисунок 3.10 — Класс конфигурации

Класс столбца содержит в себе информацию об атрибуте:

а) Table - ссылка на имя таблицы;

б) Name - название атрибута;

в) ColumnType - тип данных атрибута;

г) Priority - максимальный размер для данного типа данных;

д) Constrains - структура, хранящая флаги наличия стандартных ограничений у атрибута;

е) ValuesFactor - количество значений атрибута;

ж) UniqueFactor - количество уникальных значений атрибута.

```
type Column struct {
    Table *string
    Name       string
    ColumnType string
    Priority uint

    Constrains struct {
        PrimaryKey bool
        Key        bool
        Exclusion  bool
        Sequence   bool
        ReferenceKey bool
    }
    ValuesFactor uint
    UniqueFactor uint
}
```

Рисунок 3.11 — Класс колонки

Класс домена содержит в себе информацию о выбранном оптимальном домене:

а) ColumnIndexes - массив индексов атрибутов, вошедших в оптимальный домен;

б) UniqueFactor - количество уникальных кортежей.

```
type Domen struct {
    ColumnIndexes []int
    UniqueFactor  int64
}
```

Рисунок 3.12 — Класс домена

Класс битсета содержит в себе информацию о кодировке особи:

а) columnsNum - фиксированный размер кодировки особей.

```
type myBitsetCreate struct {
    columnsNum int
}
```

Рисунок 3.13 — Класс битсета

Класс генетического алгоритма содержит в себе информацию о параметрах генетического алгоритма:

а) Mater - класс, реализующий оператор кроссовера;

б) Simulator - класс, реализующий получения оценки функции приспособленности особи;

в) EliteConsumer - класс, реализующий схему селекции для особи;

г) Selector - класс, реализующий селекцию особей;

д) BitsetCreate - класс, реализующий создание кодировок особей начального поколения;

е) populationSize - размер популяции;

ж) population - массив структур, представляющих особей;

з) totalFitness - суммарное значение функции приспособленности поколения;

и) genomeSimulationChannel - буферизированный канал для коммуникации между легковесными потоками;

к) exitFunc - условие остановки генетического алгоритма;

л) waitGroup - примитив синхронизации легковесных потоков;

м) parallelSimulations - количество легковесных потоков для проведения симуляции.

```
type GeneticAlgorithm struct {
    Mater           IMater
    EliteConsumer   IEliteConsumer
    Simulator       ISimulator
    Selector        ISelector
    BitsetCreate    IBitsetCreate

    populationSize          int
    population              []IGenome
    totalFitness            int
    genomeSimulationChannel chan *IGenome
    exitFunc                func(*IGenome) bool
    waitGroup               *sync.WaitGroup
    parallelSimulations     int
}
```

Рисунок 3.14 — Класс обобщенного генетического алгоритма

Класс схемы селекции содержит в себе следующую информацию:

а) currentIter - номер текущего поколения.

```
type myEliteConsumer struct {
    currentIter int
}
```

Рисунок 3.15 — Класс селекции

Класс оператора скрещивания содержит в себе следующую информацию:

а) materConfig - параметры оператора скрещивания;
б) elite - ссылка на лучшую особь текущего поколения.

```
type mater struct {
    materConfig []MaterFunctionProbability
    elite       *IGenome
}
```

Рисунок 3.16 — Класс скрещивания

Класс симуляции содержит в себе следующую информацию:

а) DB - ссылка на объект базы данных;

б) columns - массив имен атрибутов, допущенных до сжатия;

в) columnValues - ассоциативный массив, хранящий количество уникальных атрибутов для каждого из доменов.

```
1 type stringMaterSimulator struct {
2     DB           *database.PgxDB
3     columns      []string
4     size         int
5     columnValues map[int]int
6 }
```

Рисунок 3.17 — Класс симуляции

4 Исследовательский раздел

4.1 Описание данных, используемых для тестирования

Для тестирования работы метода сжатия базы данных Music Label Dataset. Данный датасет является синтетически сгенерированным и содержит две таблицы: таблицу артистов и таблицу альбомов.

Тестирование рабооты метода проводилось на таблице альбомов. Таблица содержит следующие столбцы:

— id - идентификатор альбома;

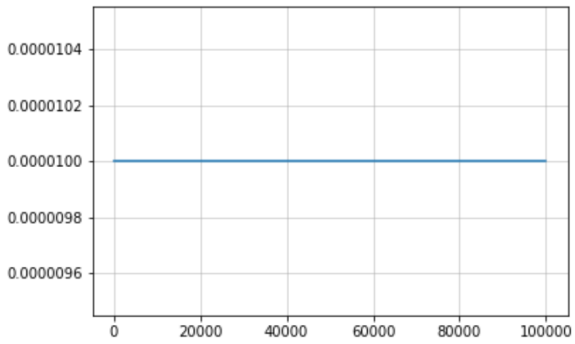

Рисунок 4.1 — Распределение идентификатор альбома

— artist_id - идентификатор артиста;

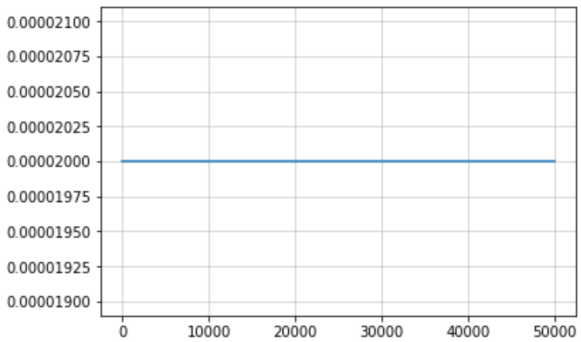

Рисунок 4.2 — Распределение идентификатор артиста

— album_title - название альбома;
— genre - жанр альбома;
— year_of_pub - год публикации;

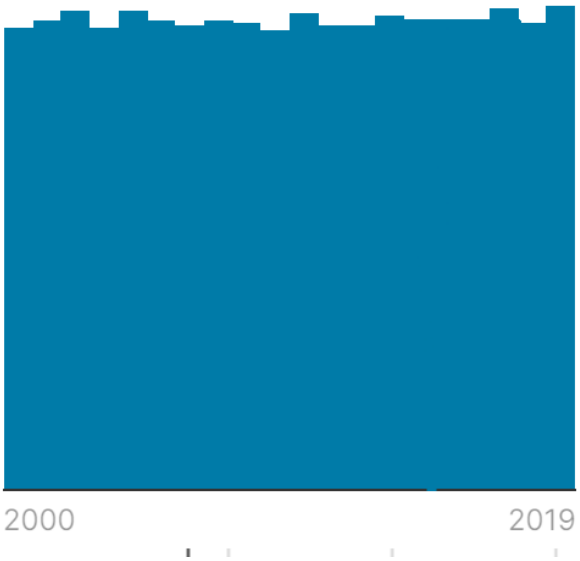

Рисунок 4.3 — Распределение года публикации

— num_of_tracks - количество треков в альбом;

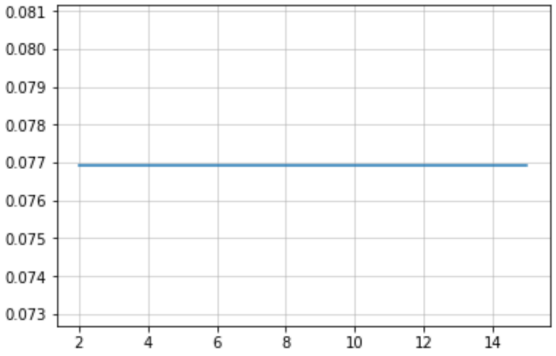

Рисунок 4.4 — Распределение количества треков в альбоме

— num_of_sales - количество продаж;

Рисунок 4.5 — Распределение количества продаж

— rolling_stone_critic - оценка критика;
— mtv_critic - оценка критика;
— music_maniac_critic - оценка критика.

Таблица 4.1 — Описание тестовых данных

№	Название столбца	Тип данных	Ограничения	Количество уникальных значе
1	id	serial	primary key	99999
2	artist_id	serial	freign key	43225
3	album_title	text	-	67815
4	genre	text	-	38
5	year	int	-	20
6	tracks	int	-	14
7	sales	int	-	95161
8	RS stat	float	-	10
9	MTV stat	float	-	10
10	MM stat	flat	-	10

4.2 Время работы в зависимости от количества потоков

Было проведено исследование скорость работы программного комплекса в зависимости от количества потоков. В реализованном программном комплексе распараллелен модуль генетического алгоритма.

Для исследования скоростных характеристик был использован компьютер на базе процессора Intel Core i7-6600U, содержащий 16 гигабайт оперативной памяти. Модуль тестирования запускался с жестокго диска под операционной системой Windows. Жесткий диск имел среднюю скорость передачи данных при чтении 2629 Мбайт/с и 798 Мбайт/с при записи.

Тестирование проводилось на исходной таблице, пороговый коэффициент K взят за единицу. Замеры времени проводились для 1, 2, 4, 8 и 16 потоков. Временные замеры проводятся путём многократного проведения эксперимента и деления результирующего времени на количество итераций эксперимента.

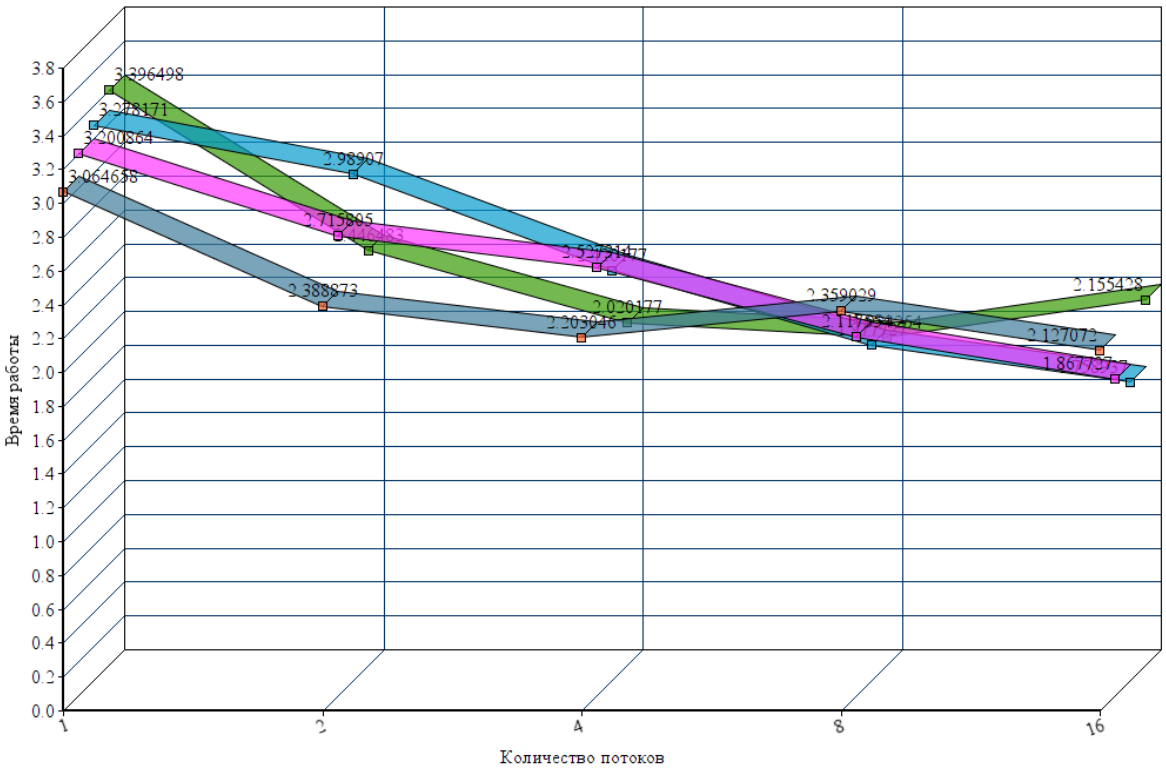

Рисунок 4.6 — Зависимость времени работы от количества потоков

4.3 Время работы в зависимости от коэффициента сжатия

Было проведено исследование скорость работы программного комплекса в зависимости от порогового коэффициента уникальности. Пороговый коэффициент уникальности K определяется как соотношение общего количества значений к количеству уникальных значений. Как

видно из приведенной таблицы, с повышением порогового коэффициента уменьшается количество столбцов, отфильтрованных для дальнейшего сжатия. С уменьшением количества столбцов возрастает скорость работы, однако количество столбцов не влияет на количество уникальных кортежей в итоговом домене.

Таблица 4.2 — Зависимость времени работы от коэффициента уникальности

№	K	Столбцы для сжатия	Коэффициент сжатия домена	Время работы алгоритма
1	1	9	999	4.8s
2	2	7	999	3.9s
3	4	6	999	2.6s
4	8	6	999	2.6s
5	16	6	999	2.6s
6	32	6	999	2.6s
7	64	6	999	2.6s
8	128	6	999	2.6s
9	256	6	999	2.6s
10	512	6	999	2.6s
11	1024	6	999	2.6s
12	2048	5	999	1.07s
13	4096	3	999	524ms

4.4 Эффективность работы в зависимости от объема таблицы

Было проведено исследование скорость работы программного комплекса в зависимости от объема таблицы. Для тестирования данные из исходной таблицы были продублированы N раз в тестируемую таблицу, значение N соответствует номеру эксперимента. Количество строк влияет на скорость выполнения SELECT-запросов, используемых при оценке приспособленности особей, что напрямую влияет на общее время выполнения программы.

Таблица 4.3 — Зависимость времени работы от размера таблицы

№	K	Количество строк	Время работы
1	8	10000	1,5
2	8	20000	1,8
3	8	30000	2,6
4	8	40000	3,4
5	8	50000	3,7

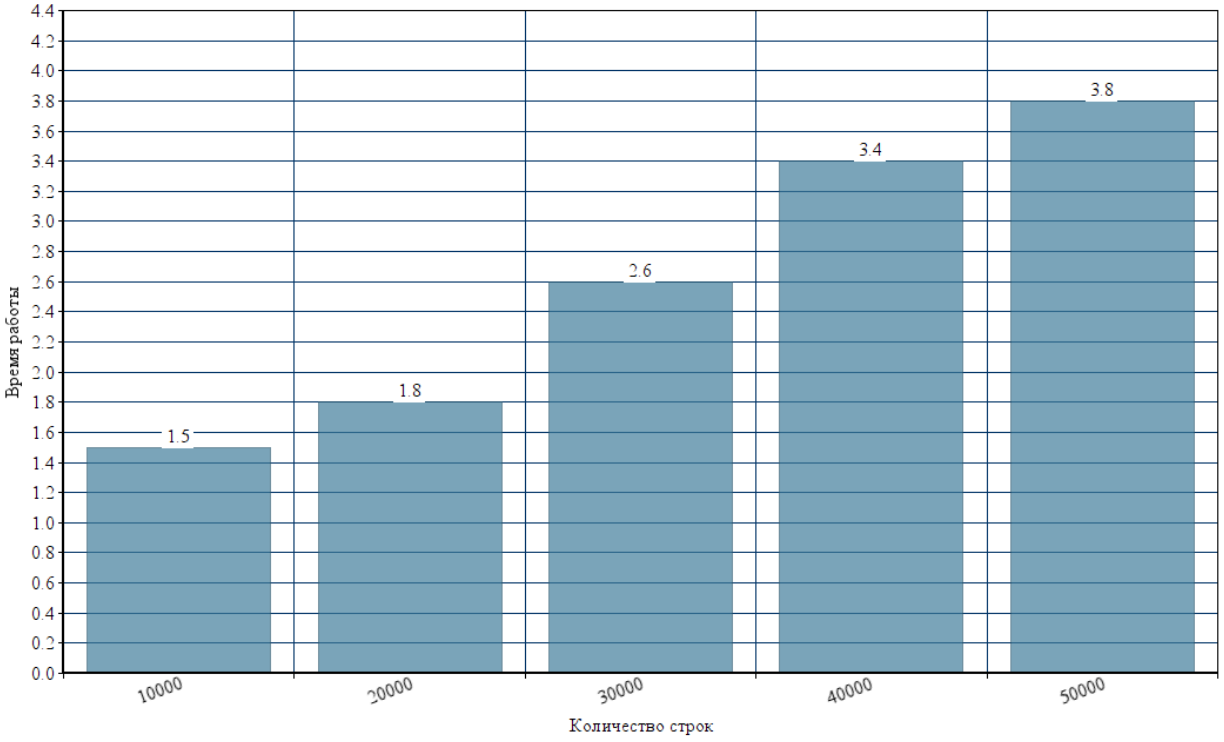

Рисунок 4.7 — Визуализация результатов работы

4.5 Эффективность работы в зависимости от количества поколений

Было проведено исследование работы программного комплекса в зависимости от объема таблицы. Для тестирования выбрана исходная таблица и зафиксирован пороговой коэффициент K равный восьми. На

этапе предварительной фильтрации в дальнейшем сжатии участвует шесть атрибутов. Исследование проводилось на размере начальной популяции от одного до ста двеннадцати, что соответствует количеству всех возможных доменов размера от 2 до 5, т.е. количеству вариантов при полном переборе.

Таблица 4.4 — Зависимость количества уникальных кортежей от размера начальной популяции

№	K	Размер популяции	Количество уникальных кортежей
1	8	1	2499
2	8	30	999
3	8	60	999
4	8	90	999
5	8	112	999

Заключение

В данной выпускной квалификационной работе были успешно достинуты все поставленные задачи:

а) была проанализирована предметная область и существующие виды баз данных;

б) были проанализированы существующие методы сжатия баз данных;

в) был реализован программный комплекс, реализующий предложенный метод.

В результате выполнения задач, был разработан метод фрактального сжатия реляционных баз данных. Для выбора оптимального домена для сжатия были применены генетические алгоритмы.

Данный метод имеет следующие возможные направления развития:

а) модификация метода для работы с No-SQL базами данных;

б) модификация метода с учетом типового разнообразия СУБД;

в) модификация метода для сжатия как самих атрибутов, так и их элементов.

Список использованных источников

1. Мандельброт, Б. Фрактальная геометрия природы. / Б. Мандельброт — Москва: Институт компьютерных исследований, 2002. — 656 с.

2. С. Л. Деменюк "Просто Фрактал Санкт-Петербрг, Страта, 2020 г.

3. Чернышев А. Н. Методы сжатия баз данных // Математика и информационные технологии в нефтегазовом комплексе. — 2015. — № 2. — с. 105–113.

4. Медведкова И. Е., Бугаев Ю. В., Чикунов С. В. Базы данных: учеб. пособие. — Воронеж: Изд-во ВГУИТ, 2014. — 105 с

5. Дейт, К.Дж. Введение в системы баз данных, 8-е издание. / К.Дж. Дейт — М.: Издательский дом «Вильямс», 2006. — 1328 с.

6. Редмонд Эрик, Уилсон Джим Р. Семь баз данных за семь недель - М.: Издательский дом "ДМК Пресс 2018

7. Т.Ю. Лымарь, Т.С. Мантрова, Н.Ю. Староверова "Алгоритм фрактального поиска в реляционных базах данных"— Челябинск: Издательский центр ЮУрГУ, 2014. — С. 369.

8. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации // Учебнометодические материалы по программе повышения квалификации, Нижний Новгород, 2007. — с. 33–80.

9. Батищев, Д.И. Генетические алгоритмы решения экстремальных задач [Текст]/ Д.И. Батищев ; Нижегородский госуниверситет. — Нижний Новгород : 1995.с. — 62с

10. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы [Текст]/ Под ред. В.М. Курейчика. — 2-е изд., испр. и доп. — М.: ФИЗМАТЛИТ, 2006. — 320 с. — ISBN 5-9221-0510-8.

11. Т.В. Панченко "Генетические алгоритмы Астрахань: Издательский дом «Астраханский университет», 2007

12. Fogel D. B. Evolutionary computation: towards a new philosophy of machine intelligence [Текст]/D. B. Fogel. — Piscatway: IEEE Press, 2000. — ISBN 0-7803-3481-7

13. Mitchell M. An Introduction to Genetic Algorithms [Текст]/M. Mitchell. — Cambridge: MIT Press, 1999 —158 с. — ISBN 0-262-13316-4 (HB), 0-262-63185-7 (PB)

14. Генетические алгоритмы на сайте Санкт-Петербугского государственного университета информационных технологий, механики и оптики. — http://rain.ifmo.ru/cat

15. Генетические алгоритмы, Исаев А. — http://www.algolist.manual.ru

ПРИЛОЖЕНИЕ А

```
{
  "input": {
    "host": "127.0.0.1",
    "user": "postgres",
    "password": "password",
    "database": "diploma",
    "table": "labels",
    "port": 5432,
    "max_conn": 50
  },
  "output": {
    "host": "127.0.0.1",
    "user": "postgres",
    "password": "password",
    "database": "diploma",
    "table": "compressed",
    "port": 5432,
    "max_conn": 50
  }
}
```

Рисунок 4.1 — Пример файла конфигурации