Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ

по преддипломной практике

Место прохождения практики: ЗАО «Научсофт»

Сроки прохождения практики: с 30.03.2023 по 26.04.2023

|  |  |
| --- | --- |
| Руководитель практики от  Предприятия  **\_\_\_\_\_\_\_\_\_\_\_\_**Розум Е.Н.  (подпись руководителя) | Студент группы 893551  **\_\_\_\_\_\_\_\_\_\_\_\_\_**Драгун О.В.  (подпись студента)  Руководитель практики от БГУИР  Рогов М.Г. |
|  |

Минск 2023

# 1 ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

## 1.1 Возможные решения для исходной задачи

Исходная задача нахождения оптимального маршрута является np-сложной задачей, с экспоненциальным ростом возможных решений по мере роста количества участвующих в задаче элементов. Для её решения существует множество возможных алгоритмом, перечислим некоторые из них и коротко опишем каждый.

* + 1. **Жадный алгоритм**

Жадный алгоритм для решения оптимизации транспортной задачи работает по принципу нахождения наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, начиная с наименьших затрат. Этот алгоритм прост в реализации и может дать приближенный результат за короткое время, но не гарантирует нахождение оптимального решения.

* + 1. **2-Opt алгоритм**

2-Opt - это алгоритм оптимизации маршрута, использующий принцип локального поиска. Алгоритм пытается улучшить качество текущего решения, переставляя порядок двух произвольных вершин маршрута, и проверяет, улучшится ли стоимость маршрута. Этот алгоритм может работать быстро и дать хороший результат в некоторых случаях, но не гарантирует нахождение глобального оптимума.

* + 1. **Жадный 2-Opt алгоритм**

Жадный 2-Opt - это алгоритм, который комбинирует принципы жадного алгоритма и 2-Opt. Он начинает с поиска наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, а затем применяет 2-Opt для оптимизации маршрута. Этот алгоритм может работать быстро и давать хороший результат, но не гарантирует нахождение глобального оптимума.

* + 1. **3-Opt алгоритм**

3-Opt - это алгоритм оптимизации маршрута, который расширяет принципы 2-Opt, переставляя порядок трех вершин маршрута, а не двух, как в 2-Opt. Этот алгоритм может работать лучше, чем 2-Opt, но имеет большую вычислительную сложность.

* + 1. **Жадный 3-Opt алгоритм**

Жадный 3-Opt - это алгоритм, который комбинирует принципы жадного алгоритма и 3-Opt. Он начинает с поиска наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, а затем применяет 3-Opt для оптимизации маршрута. Этот алгоритм может работать быстро и давать хороший результат, но не гарантирует нахождение глобального оптимума.

* + 1. **Генетический алгоритм**

Генетический алгоритм для решения оптимизации транспортной задачи использует принципы эволюционной биологии, чтобы найти оптимальное решение. Алгоритм создает популяцию случайных решений и выполняет операции скрещивания и мутации, чтобы создавать новые решения. Затем оценивается стоимость каждого решения, и те, которые показывают лучшие результаты, используются для создания следующего поколения решений. Процесс повторяется, пока не будет найдено оптимальное решение.

* + 1. **Алгоритм иммитации отжига**

Имитации отжига - это алгоритм, который использует техники стохастического моделирования для оптимизации транспортной задачи. Он начинает со случайного решения и выполняет последовательность случайных изменений в решении, которые могут как улучшить, так и ухудшить стоимость решения. Вероятность принятия изменения, которое ухудшает решение, определяется температурой системы, которая постепенно снижается во время выполнения алгоритма. Этот алгоритм может работать хорошо для оптимизации сложных задач, но требует настройки параметров.

* + 1. **Нейронная сеть**

Нейронная сеть - это алгоритм, который использует искусственные нейронные сети для оптимизации транспортной задачи. Нейронная сеть обучается на данных о предыдущих решениях транспортной задачи, чтобы предсказывать оптимальный маршрут для новых данных. Она может работать хорошо для оптимизации задач с большим количеством переменных, но требует большого объема данных и вычислительных ресурсов для обучения.

* + 1. **Муравьиная колония**

Муравьиная колония - это алгоритм, который использует принципы поведения муравьев для оптимизации транспортной задачи. Алгоритм создает колонию виртуальных муравьев, которые перемещаются по графу между источниками и потребителями, оставляя феромоны на своем пути. Чем больше феромонов оставляют муравьи на определенном маршруте, тем больше вероятность, что другие муравьи выберут этот маршрут. Этот алгоритм может работать хорошо для задач оптимизации.

Ввиду ограничения мощности процессора на множестве мобильных устройств, рекомендации из последних исследований в области решений задач оптимизации [1], [2], [3], ограничения по времени на реализацию минимально рабочей версии, наиболее перспективным вариантом выглядит Генетический алгоритм (далее по тексту – ГА). Рассмотрим его подробнее.

## 1.2 Общая схема ГА

Генетический алгоритм (ГА) - это оптимизационный метод, использующий механизмы естественного отбора и генетической рекомбинации для решения различных задач. Он включает в себя 6 главных шагов (рисунок 1.1).

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.1 − Схема ГА

* + 1. **Инициализация**

Создание начальной популяции из случайных решений (хромосом). Может содержать параметр обязательной уникальности между хромосомами. Также для задач с большим числом параметром может иметь место неслучайная генерация хромосом, что, однако, не является классической инициализацией в ГА.

* + 1. **Оценка**

Оценка каждого индивида в популяции на основе фитнесс-функции которая определяет качество решения. Для транспортной задачи это либо суммарная дистанция между начальной и конечной точкой, со всеми остальными точками между ними, либо суммарное ожидаемое время в пути.

* + 1. **Селекция**

После оценки генетических кодов происходит выбор лучших кодов для создания следующего поколения. Это делается на основе ранжирования генетических кодов по их фитнес-функции. Лучшие генетические коды имеют больший шанс быть выбранными для скрещивания и создания следующего поколения. Существуют разные стратегии селекции, которые мы рассмотрим далее.

* + 1. **Скрещивание**

Скрещивание генов от двух разных родителей. Выбранные хромосомы скрещиваются, чтобы создать новые хромосомы для следующего поколения. Скрещивание может происходить по различным правилам, таким как одноточечное или многоточечное скрещивание, или смешивание генов

* + 1. **Мутация**

Некоторые генетические коды в следующем поколении могут подвергаться мутации, что приводит к изменению их генетического материала. Мутация может случайно изменить гены генетического кода, что иногда помогает избежать застревания в локальном оптимуме.

* + 1. **Повторение**

Все эти шаги повторяются, пока не будет достигнуто желаемое решение или пока не будет исчерпан лимит итераций. При каждом повторении популяция становится более оптимизированной, поскольку лучшие генетические коды имеют больший шанс продолжить свое существование.

## 1.3 Кодирование в ГА

Рассмотрим различные варианты кодирования хромосомы, возможные для ГА:

* + 1. **Двоичное кодирование**

Этот метод заключается в представлении каждой переменной решения в двоичной форме, используя битовую строку. Так, каждый бит представляет собой одно из двух возможных значений (0 или 1), которые соответствуют определенному значению переменной. Например, если переменная имеет диапазон значений от 0 до 15, то для ее кодирования потребуется 4 бита.

Пусть имеется 9 генов, каждый из которых может принимать значение 0 или 1. Тогда хромосома может выглядеть следующим образом:

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.2 − Пример двоичного кодирования

* + 1. **Восьмеричное кодирование**

Этот метод использует восьмеричную систему счисления для кодирования каждой переменной. Таким образом, каждый символ в строке соответствует одному значению переменной. Например, если переменная имеет диапазон значений от 0 до 63, то для ее кодирования потребуется 2 символа восьмеричной системы.

Изображение выглядит как диаграмма

Автоматически созданное описаниеПусть имеется 9 генов, каждый из которых может принимать значение от 0 до 7. Тогда хромосома может выглядеть следующим образом:

Рисунок 1.3 − Пример восьмеричного кодирования

* + 1. **Шестнадцатеричное кодирование**

Этот метод использует шестнадцатеричную систему счисления для кодирования каждой переменной. Таким образом, каждый символ в строке соответствует одному значению переменной. Например, если переменная имеет диапазон значений от 0 до 255, то для ее кодирования потребуется 2 символа шестнадцатеричной системы.

Пусть имеется 9 генов, каждый из которых может принимать значение от 0 до 15. Тогда хромосома может выглядеть следующим образом:



Рисунок 1.4 − Пример шестнадцатеричного кодирования

* + 1. **Перестановочное кодирование**

Этот метод используется для кодирования переменных, которые представляют собой наборы элементов, упорядоченных в определенном порядке. Каждый элемент пронумерован, и кодирование осуществляется путем представления порядка элементов в виде перестановки. Например, для набора из 5 элементов возможно 5! (120) различных перестановок.

Пусть имеется 9 генов, которые могут принимать любое из 9 возможных значений без повторений. Тогда хромосома может выглядеть следующим образом:

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.5 − Пример перестановочного кодирования

* + 1. **Кодирование значения**

Этот метод используется для кодирования переменных, которые представляют собой дискретные значения, такие как целые числа, булевы или категориальные переменные. Каждое значение переменной назначается уникальный код, который может быть использован для генерации новых вариантов комбинаций. Например, для переменной, которая может принимать одно из четырех возможных значений (например, "красный", "зеленый", "синий", "желтый"), каждому значению может быть назначен уникальный код (например, 00, 01, 10, 11).

Пусть имеется 9 генов, каждый из которых может принимать значение от 1 до 5. Тогда хромосома может выглядеть следующим образом:

Изображение выглядит как текст, часы

Автоматически созданное описание

Рисунок 1.6 − Пример кодирования значения

* + 1. **Кодирование деревом**

Этот метод используется для кодирования структур данных, таких как деревья или графы. Каждый узел в дереве или графе кодируется отдельно, и кодирование осуществляется путем представления каждого узла в виде строки или последовательности символов. Например, для бинарного дерева каждый узел может быть закодирован с использованием 3 символов, соответствующих значению узла и его двум потомкам.

Пусть имеется дерево, состоящее из 9 узлов. Каждый узел может быть либо листом, либо внутренним узлом, и иметь метку соответствующую его значению. Тогда хромосома может быть закодирована в виде последовательности меток узлов дерева:



Рисунок 1.7 − Пример кодирования деревом

Задача поиска оптимального маршрута идеально кодируется при помощи перестановочного кодирования. Стоит оценить количество всех возможных перестановок для различного кол-ва городов, чтобы иметь представление о сложности задачи по мере роста количества городов. В таблице 1.1 приведено количество перестановок для разного числа городов

Таблица 1.1 – Количество перестановок для разного числа городов

|  |  |
| --- | --- |
| Число городов | Количество перестановок |
| 10 | 3628800 |
| 50 | 30414093201713378043612608166064768844377641568960512000000000000 |
| 90 | 1485715964481761497309522733620825737885569961284688766942216863704985393094065876545992131370884059645617234469978112000000000000000000000 |
| 130 | 6466855489220473672507304395536485253155359447828049608975952322944781961185526165512707047229268452925683969240398027149120740074042105844737747799459310029635780991774612983803150965145600000000000000000000000000000000 |

Для 130 городов это ~6.5 в 219 степени. При взгляде на это число становится понятно, насколько сложную задачу способен решить генетический алгоритм при корректной имплементации.

## 1.4 Селекция в ГА

Селекция – важный шаг генетического алгоритма, который определяет, будет ли конкретная хромосома участвовать в процессе скрещивания или нет. Скорость сходимости ГА зависит от давления селекции. Хорошо известными методами селекции являются колесо рулетки, ранг, турнир, Больцман и стохастическая универсальная выборка.

Селекция *колесом рулетки* отображает все возможные хромосомы на колесе с частью колеса, выделенной для них в соответствии с их значением фитнес-функции. Затем это колесо случайным образом вращается для выбора конкретных хромосом, которые будут участвовать в формировании следующего поколения [4]. Однако данная селекция страдает от многих проблем, например ошибками, вызванными его стохастическим характером.

Де Йонг и Бриндл модифицировали метод выбора колеса рулетки, чтобы устранить ошибки, введя концепцию детерминизма в процедуру селекции. *Селекция рангом* – это модифицированная форма селекции колесом рулетки. Она использует ранги вместо значения фитнес-функции. Ранги присваиваются хромосомам в соответствии с их фитнес-функцией, так что каждая хромосома получает шанс быть выбранной в соответствии со своим рангом. Метод ранговой селекции снижает вероятность преждевременной сходимости решения к локальным минимумам [4].

Техника турнирной селекции была впервые предложена Бриндлом в 1983 году. Хромосомы выбираются парами в соответствии с их значениями фитнес-функции из стохастического колеса рулетки. После селекции особи с более высоким значением фитнесс-функции добавляются в пул следующего поколения [4]. В этом методе селекции каждая особь сравнивается со всеми n-1 другими особями, если она достигает конечной популяции[4].

*Стохастическая универсальная выборка* является расширением существующего метода селекции колесом рулетки. Этот метод использует случайную начальную точку в списке особей из поколения и выбирает новую особь через равные промежутки [5]. Это дает равные шансы всем хромосомам быть отобранными для участия в скрещивании следующего поколения. Хотя в случае задачи коммивояжера стохастическая универсальная выборка работает хорошо, но по мере увеличения размера задачи традиционный выбор колеса рулетки работает относительно хорошо [6].

Выбор Больцмана основан на методах энтропии и дискретизации, которые используются в моделировании Монте-Карло. Это помогает в решении проблемы преждевременной сходимости [7]. Вероятность выбора наилучшей хромосомы очень высока, хотя она выполняется за очень короткое время. Однако существует вероятность потери информации. Этого можно избезать при помощи элитизма [8]. Выбор элитизма был предложен К. Д. Джонгом (1975) для улучшения эффективности выбора колеса рулетки. Это гарантирует, что элитарная хромосома в поколении всегда передается следующему поколению. Если хромосома с наивысшим значением фитнес-функции не присутствует в следующем поколении после обычной процедуры селекции, то элитарная особь также автоматически включается в следующее поколение [4]. Сравнение вышеупомянутых методов селекции показано в таблице 1.2.

Таблица 1.2 – Сравнение разных техник селекции в ГА

|  |  |  |
| --- | --- | --- |
| Техника селекции | Преимущества | Недостатки |
| Колесо рулетки | Легок в имплементации;  Прост;  Непредвзят. | Риск преждевременной конвергенции;  Зависит от дисперсии, присутствующей в фитнес-функции. |
| Ранг | Сохраняет разнообразие;  Непредвзят. | Медленная сходимость;  Требуется сортировка;  Вычислительно дорого. |
| Турнир | Сохраняет разнообразие;  Параллельная реализация;  Сортировка не требуется. | Потеря разнообразия при большом размере турнира. |
| Больцман | Достигает глобального оптимума. | Вычислительно дорого. |
| Стохастическая универсальная выборка | Быстрый метод;  Непредвзят. | Преждевременная конвергенция. |
| Элитизм | Сохраняет лучшую особь в популяции; | Лучшая хромосома может быть потеряна из-за операторов скрещивания и мутации. |

## 1.5 Скрещивание в ГА

Существует множество вариантов скрещивания в зависимости от выбранной системы кодирования для конкретной задачи. Рассмотрим подробнее операции скрещивания, применимые к перестановочному кодированию, поскольку именно это кодирование используется в задаче поиска оптимального маршрута. Будем использовать общепринятые названия на английском, поскольку общепринятых устоявшихся названий в русскоязычной литературе не встречается.

* + 1. **Order crossover**

Order crossover характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов родителя;

2) Сформировать прото-ребенка путем копирования подстроки родителя в соответствующие позиции;

3) Удалить гены, присутствующие в подстроке у второго родителя. Результирующая последовательность генов содержит гены, которых не достает прото-ребенку;

4) Поставить гены на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены далее (рисунок 1.8)

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.8 − Order crossover

* + 1. **Position-based crossover**

Position-based crossover характеризуется следующим набором шагов:

1) Выбрать случайный набор генов родителя на случайных позициях;

2) Сформировать прото-ребенка копированием набора в соответствующие позиции;

3) Удалить гены, присутствующие в наборе у второго родителя. Результирующая последовательность генов содержит гены, которых не достает прото-ребенку;

4) Поставить гены на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены далее (рисунок 1.9)

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.9 − Position-based crossover

* + 1. **Order-based crossover**

Order crossover характеризуется следующим набором шагов:

1) Выбрать случайный набор генов первого родителя на случайных позициях;

2) Удалить гены, присутствующие в наборе с предыдущего шага у второго родителя;

3) Поставить гены выбранные у первого родителя на позиции удаленных генов второго родителя

4) Поставить оставшиеся гены второго родителя на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены далее (рисунок 1.10)

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.10 − Order-based crossover

* + 1. **CX crossover**

CX crossover характеризуется следующим набором шагов:

1) Найти цикл, определяющий соответствующие позиции генов между родителями;

2) Скопировать гены из цикла в прото-ребенка на соответствующие первому родителю позиции;

3) Определить оставшиеся города у ребенка, удалив те города, которые уже есть в цикле у другого родителя;

4) Дополнить прото-ребенка оставшимися генами для получения ребенка.

Схематично шаги изображены далее (рисунок 1.11)

Изображение выглядит как диаграмма

Автоматически созданное описаниеИзображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.11 − CX crossover

* + 1. **Subtour exchange crossover**

Subtour exchange crossover характеризуется следующим набором шагов:

1) Найти максимальный подпуть, содержащийся в обоих родителях;

2) Первого ребенка сформировать скопировав родителя 1 с заменой подпути на подпуть от родителя 2;

3) Второго ребенка сформировать скопировав родителя 2 с заменой подпути на подпуть от родителя 1.

Схематично шаги изображены далее (рисунок 1.12)

Изображение выглядит как текст, электроника, клавиатура

Автоматически созданное описание

Рисунок 1.12 − Subtour exchange crossover

## 1.6 Мутация в ГА

Мутация является последним шагом в ГА. Она позволяет избежать застревания алгоритма в локальном оптимуме и вырождения генома. Существует множество вариантов мутации в зависимости от выбранной системы кодирования для конкретной задачи. Рассмотрим подробнее операции мутации, применимые к перестановочному кодированию, поскольку именно это кодирование используется в задаче поиска оптимального маршрута. Будем использовать общепринятые названия на английском, поскольку общепринятых устоявшихся названий в русскоязычной литературе не встречается.

* + 1. **Inversion mutation**

Inversion mutation характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов;

2) Мутировать хромосому путем перестановки генов в ней.

Схематично шаги изображены далее (рисунок 1.13)

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.13 − Inversion mutation

* + 1. **Insertion mutation**

Insertion mutation характеризуется следующим набором шагов:

1) Выбрать случайный ген хромосомы;

2) Мутировать хромосому путем перемещения выбранного гена на случайную позицию, сдвинув остальные гены соответствующе.

Схематично шаги изображены далее (рисунок 1.14)

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1.14 − Insertion mutation

* + 1. **Displacement mutation**

Displacement mutation характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов;

2) Переместить гены на случайную позицию, сдвинув все остальные гены.

Изображение выглядит как диаграмма

Автоматически созданное описаниеСхематично шаги изображены далее (рисунок 1.15)

Рисунок 1.15 − Displacement mutation

* + 1. **Exchange mutation**

Exchange mutation характеризуется следующим набором шагов:

1) Выбрать два случайных гена;

2) Мутировать хромосому, обменяв выбранные гены местами.

Схематично шаги изображены далее (рисунок 1.16)

Изображение выглядит как текст, графическая вставка

Автоматически созданное описание

Рисунок 1.16 − Exchange mutation

* + 1. **Heuristic mutation**

Heuristic mutation характеризуется следующим набором шагов:

1) Выбрать λ генов на случайных позициях;

2) Сгенерировать соседей путем всех возможных перестановок выбранных генов;

3) Сравнить фитнес-функции всех соседей и выбрать лучшего в качестве мутировавшей хромосомы.

Схематично шаги изображены далее (рисунок 1.17)

Изображение выглядит как текст, электроника, калькулятор, клавиатура

Автоматически созданное описание

Рисунок 1.17 − Heuristic mutation

# 2 ОБЗОР ТЕХНОЛОГИЙ

## 2.1 Операционная система Android

Android – это операционная система, разработанная компанией Google для мобильных устройств, таких как смартфоны, планшеты, умные часы и телевизоры. Она была выпущена в 2008 году и быстро стала одной из самых популярных мобильных операционных систем в мире. ОС основана на ядре Linux и использует открытые стандарты и протоколы [9].

Одним из главных преимуществ Android является его открытость. Это означает, что разработчики могут создавать приложения и настраивать операционную систему под свои нужды. Это способствует более широкому выбору приложений для пользователей и повышает конкуренцию на рынке. Кроме того, Android имеет богатый набор инструментов для разработчиков, включая среду разработки Android Studio и открытые API для создания интеграций с другими приложениями и сервисами.

Вторым преимуществом Android является его экосистема приложений. Google Play, официальный магазин приложений для большинства смартфонов Android. Это означает, что пользователи могут легко найти и загрузить приложения для различных целей, от игр до продуктивности и социальных сетей. Что дает доступ к пользователям по всему земному шару. Это важно, если мобильное приложение подразумевает его развитие и окупаемость.

Третьим преимуществом Android является его адаптивность и масштабируемость. Android поддерживает широкий диапазон устройств, от недорогих смартфонов до мощных планшетов и флагманских смартфонов. Это означает, что разработчики могут создавать приложения для разных устройств, и пользователи могут выбирать устройство, соответствующее их потребностям и бюджету. Кроме того, Android имеет богатый набор функций и возможностей, которые могут быть настроены и масштабированы в соответствии с потребностями пользователя.

Одним из главных недостатков Android является фрагментация. Это означает, что разные устройства могут использовать разные версии операционной системы и иметь разные функции и возможности. Это может создавать эффект "разброса" среди пользователей, что затрудняет установку одних и тех же приложений и создание универсальных решений для всех устройств. Кроме того, фрагментация может привести к более медленным обновлениям системы и безопасности, поскольку различные производители могут задерживать или не выпускать обновления для своих устройств.

Вторым недостатком Android является проблема безопасности. Android как операционная система стала целью для многих кибератак, поскольку многие производители устройств не выпускают регулярные обновления безопасности. Кроме того, многие пользователи не осознают важность защиты своих устройств и могут устанавливать приложения из неизвестных источников, что может привести к уязвимостям и нарушению конфиденциальности.

Третья достаточно существенная проблема – количество потенциально платежеспособных пользователей значительно ниже, по сравнению с ОС iOS.

## 2.2 Android studio

Android Studio - это интегрированная среда разработки (IDE) для создания приложений для мобильной операционной системы Android [10]. Это официальная IDE, поддерживаемая Google, и предоставляет разработчикам инструменты, необходимые для создания высококачественных приложений для мобильных устройств на базе Android. Главное окно интегрированной среды разработки представлено далее (рисунок 2.1).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.1− Главное окно IDE

Android Studio основан на IntelliJ IDEA, известной IDE для разработки на языках программирования Java и Kotlin. Это означает, что Android Studio предоставляет множество инструментов, которые существенно упрощают процесс разработки для разработчиков, знакомых с IntelliJ IDEA.

Основными функциями Android Studio являются редактор кода, отладчик, графический редактор макетов и средства для сборки и тестирования приложений. Android Studio также поддерживает использование многих других инструментов и технологий, таких как Git для контроля версий, Gradle для сборки проектов и Android Virtual Device Manager (AVD Manager) для эмуляции устройств Android.

Одна из главных причин, по которой разработчики предпочитают использовать Android Studio, это потому, что оно обеспечивает полную интеграцию с платформой Android и предоставляет разработчикам множество средств и ресурсов для создания мобильных приложений, таких как мощные средства отладки, симуляторы и тестовые инструменты.

## 2.3 Язык программирования Kotlin

Kotlin - это статически типизированный язык программирования, который был разработан JetBrains и выпущен в 2011 году [11]. Kotlin является языком программирования, совместимым с Java, и может быть использован для написания приложений для платформы Java Virtual Machine (JVM), а также для Android и других платформ.

Kotlin был создан для улучшения производительности и эффективности программистов, сделав программирование более простым, чистым и безопасным. Он предлагает множество удобных функций, таких как нулевые ссылки, расширения функций, анонимные объекты и многое другое.

Одно из главных преимуществ Kotlin - это то, что он является языком программирования, который легко читается и понимается, даже для тех, кто никогда не работал с Java или другими языками программирования. Благодаря его интуитивно понятному синтаксису Kotlin позволяет разработчикам писать код более быстро и эффективно, что особенно важно в мобильной разработке.

Кроме того, Kotlin также предлагает множество инструментов и библиотек, которые делают разработку мобильных приложений еще более простой и удобной. Kotlin имеет мощную систему типов, которая обеспечивает безопасность и надежность кода, что особенно важно для мобильных приложений, работающих с чувствительными данными.

В целом, Kotlin является отличным выбором для разработки мобильных приложений из-за его легкочитаемости, безопасности и возможности использования для мультиплатформенной разработки. Благодаря растущей популярности Kotlin Multiplatform, его использование в мобильной разработке будет только расти, что позволит разработчикам создавать приложения для различных платформ быстрее и проще, используя один и тот же язык программирования.

## 2.4 Kotlin multiplatform mobile

Kotlin Multiplatform Mobile (KMM) – это технология, которая позволяет разработчикам создавать приложения для различных мобильных платформ, используя Kotlin [12]. Она позволяет сократить время разработки и облегчить поддержку кода, поскольку разработчики могут использовать один и тот же код на нескольких платформах, включая iOS и Android.

KMM позволяет разработчикам писать общий код для бизнес-логики, моделей данных и других функций приложения, которые не зависят от конкретной платформы. Код, который зависит от платформы, такой как пользовательский интерфейс, может быть написан в специфических для каждой платформы модулях. Более того, в марте-апреле 2023 года Compose Multiplatform для системы iOS перешел в стадию Alpha, что означает возможность использовать один код и для UI. Такой подход позволяет разработчикам сократить время на написание кода, упростить его поддержку и сократить затраты на разработку и тестирование.

Одно из главных преимуществ KMM - это то, что он предлагает эффективное взаимодействие между общим кодом и специфическими для платформ модулями. Разработчики могут использовать общий код на всех платформах, однако при необходимости модифицировать его под специфические для каждой платформы требования, например, при вызове специфичных API, таких как Bluetooth или камера.

Кроме того, KMM обеспечивает высокую производительность и масштабируемость приложений, что является важным фактором при разработке мобильных приложений. Он также поддерживает интеграцию с различными инструментами и библиотеками, что позволяет разработчикам использовать свои любимые инструменты при разработке мобильных приложений.

Наконец, KMM является открытым исходным кодом, что позволяет разработчикам получать доступ к исходному коду и вносить свои вклады в развитие технологии. Это также означает, что KMM является бесплатным для использования и не требует дополнительных затрат на покупку или лицензирование.

В целом, Kotlin Multiplatform Mobile представляет собой мощную и гибкую технологию, которая позволяет разработчикам создавать мобильные приложения для различных платформ, используя Kotlin.

## 2.5 пользовательский интерфейс Compose

Compose UI - это новый декларативный подход к созданию пользовательского интерфейса в Android, основанный на языке программирования Kotlin [13]. Он предоставляет простой и интуитивно понятный способ создания пользовательского интерфейса, используя компоненты, называемые Composables.

Composables - это функции Kotlin, которые описывают пользовательский интерфейс как дерево компонентов, каждый из которых может содержать другие компоненты. Компоненты Compose можно реализовать как существующие элементы пользовательского интерфейса, такие как кнопки, поля ввода и т.д., или создать свои собственные компоненты, которые могут быть повторно использованы в других частях приложения.

Compose UI упрощает процесс создания пользовательского интерфейса и позволяет разработчикам более быстро и легко создавать красивые и функциональные пользовательские интерфейсы. Он также предоставляет функциональность для управления состоянием приложения и реактивного программирования.

Compose Multiplatform UI - это расширение Compose UI, которое позволяет разработчикам создавать многоплатформенные приложения с использованием Compose. Это означает, что разработчики могут использовать Compose для создания пользовательского интерфейса на различных платформах, включая Android, iOS.

Compose Multiplatform UI использует ту же модель программирования, что и Compose UI, что позволяет разработчикам переиспользовать код для создания пользовательского интерфейса на различных платформах. Он также предоставляет поддержку для различных элементов пользовательского интерфейса, включая текст, изображения, списки, формы и т.д.

В целом, Compose UI и Compose Multiplatform UI представляют собой мощные инструменты для разработки пользовательского интерфейса в Android и многоплатформенных приложений, которые упрощают процесс создания красивых и функциональных приложений.

## 2.6 Сетевые запросы

Retrofit и OkHTTP - это две библиотеки для выполнения сетевых запросов в Android-приложениях, написанных на языке Java или Kotlin. Retrofit предоставляет удобный API для описания REST-сервисов и их методов, которые могут быть вызваны из приложения для получения данных с сервера. OkHTTP, в свою очередь, предоставляет мощный и гибкий механизм для выполнения HTTP-запросов с различными параметрами и настройками.

В первой версии нашего мобильного приложения мы будем использовать Retrofit и OkHTTP для выполнения сетевых запросов. С их помощью мы сможем легко и эффективно получать данные с сервера, включая текстовые, бинарные и JSON-данные. Мы сможем использовать Retrofit для создания и настройки интерфейсов API, а затем вызывать методы API с помощью OkHTTP для получения данных. Retrofit и OkHTTP предоставляют обширную документацию и множество примеров использования, что делает их очень популярными инструментами для выполнения сетевых запросов в Android-приложениях.

Во второй стадии, когда мы расширим наше приложение на платформу IOS и Kotlin Multiplatform Mobile, мы будем использовать Ktor вместо Retrofit и OkHTTP. Ktor - это легковесный и многофункциональный веб-фреймворк, написанный на языке Kotlin, который предоставляет инструменты для выполнения сетевых запросов, разработки REST API и обработки HTTP-запросов. Он обладает высокой производительностью и хорошей масштабируемостью, мощными средствами для обработки ошибок, кеширования данных и поддержки множественных протоколов. Кроме того, использование Ktor в Kotlin Multiplatform Mobile позволит нам создавать общий код для выполнения сетевых запросов на различных платформах, что значительно упростит и ускорит процесс разработки.

Таким образом, в первой версии нашего мобильного приложения мы будем использовать Retrofit и OkHTTP для выполнения сетевых запросов, а во второй стадии, при расширении на платформу IOS и Kotlin Multiplatform Mobile, мы перейдем на Ktor, который обеспечит высокую производительность и надежность выполнения сетевых запросов на всех целевых платформах, что будет способствовать более быстрой и эффективной работе нашего мобильного приложения.

## 2.7 Локальная база данных

В первой версии нашего мобильного приложения для работы с локальной базой данных мы будем использовать библиотеку ROOM. ROOM - это библиотека, предоставляемая Google, которая позволяет использовать абстракцию базы данных SQLite для более простой и эффективной работы с базой данных в приложении. ROOM предоставляет аннотации для определения сущностей базы данных, DAO-интерфейсов для выполнения запросов к базе данных и других элементов, таких как миграции, которые позволяют изменять схему базы данных без потери данных.

Во второй версии нашего мобильного приложения, при расширении на платформу IOS и Kotlin Multiplatform Mobile, мы будем использовать библиотеку sqldelight. SqlDelight - это библиотека для работы с базой данных, которая предоставляет общую схему базы данных, которую можно использовать на разных платформах, таких как Android, iOS и серверные приложения. SqlDelight генерирует типобезопасный API для доступа к базе данных, основанный на SQL-запросах и позволяет писать SQL-запросы в статически типизированном Kotlin-коде, что уменьшает вероятность ошибок при работе с базой данных.

Использование библиотек ROOM и SqlDelight обеспечивает эффективную и удобную работу с локальной базой данных в нашем мобильном приложении, как на Android, так и на iOS в будущем.

## 2.8 Платные подписки

Финальная стадия первого шага подразумевает начало получения прибыли. Для этого мы будем использовать систему подписок, предоставляемую Play Market на Android-платформе. Для простоты разработки она будет содержать всего один план, который разблокирует возможность добавления неограниченного числа пользовательских точек на карте. Это обусловлено тем, что для подсчета матрицы дистанции-время каждая очередная точка требует платных сетевых запросов, соответственно оставлять этот функционал бесплатным при запуске нельзя.

Система подписок от Play Market - это механизм, который позволяет разработчикам мобильных приложений предлагать подписки для своих пользователей на определенные услуги или функциональность в приложении. Подписки могут быть как ежемесячными, так и годовыми, и предоставлять пользователю доступ к дополнительным функциям, контенту и ресурсам, которые недоступны для пользователей, не подписавшихся на эти услуги.

Основные преимущества системы подписок от Play Market включают:

1) Удобство для пользователей – подписки могут быть оформлены прямо в приложении и не требуют от пользователя дополнительных действий, например перехода на веб-сайт разработчика;

2) Гибкость и настраиваемость – разработчики могут настраивать различные уровни подписок с различными ценами и продолжительностью подписки;

3) Расширенная аналитика – система подписок от Play Market предоставляет разработчикам много данных и аналитики, связанных с поведением пользователей и процессом подписки, что позволяет им улучшать продукт и повышать его доходность.

Вторая версия нашего мобильного приложения будет включать аналогичный функционал, предоставляемый App Store на iOS-платформе. Мы также будем использовать систему подписок.

Предоставление платных подписок нашим пользователям позволит нам зарабатывать деньги на нашем приложении, что будет способствовать развитию проекта и его поддержке в долгосрочной перспективе. Мы будем собирать отзывы и мнения пользователей о нашем приложении и постоянно работать над его улучшением, чтобы удовлетворить потребности наших пользователей и привлечь новых.