МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Преподаватель кафедры ИС

_____ В.В. Переведенцев

"___"_____ 2017 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

"___"_____ 2017 г.

РАЗРАБОТКА И АНАЛИЗ ЭФФЕКТИВНЫХ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ПРОСТЕЙШЕЙ ЗАДАЧИ РАЗМЕЩЕНИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2017.15-039-1401.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____Р.Э. Шангин

Автор работы,
студент группы КЭ-402
_____Е.А. Князев

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

"___"_____ 2017 г.

Челябинск-2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Задачи оптимального размещения предприятий применяются повсеместно, например, при планировании и реконструкции производства, проектировании сетей обслуживания, стандартизации и других областях. Существенный интерес к таким задачам связан, также, со сложностью их решения, особенно при больших размерностях задач. Поэтому для исследования и решения рассматриваемых задач требуется применение методов исследования операций и математической оптимизации [7].

Одним из важных направлений исследования в рассматриваемой области является анализ и решение задач оптимального размещения объектов. Многие задачи оптимального размещения предприятий могут быть сформулированы следующим образом. Даны пункты возможного размещения предприятий и множество клиентов, которые нуждаются в обслуживании. Требуется открыть предприятия в некоторых из указанных пунктов, прикрепить к ним клиентов и выполнить обслуживание таким образом, чтобы полученное решение было оптимальным, т.е. с наименьшими транспортными затратами и стоимостью открытия предприятий. Под обслуживанием может пониматься, например, транспортировка продукции от поставщиков к ее потребителям. Многие из рассматриваемых задач являются NP – трудными [11].

Среди задач оптимального размещения обычно выделяют два класса - задачи размещения взаимосвязанных объектов и задачи размещения - распределения. Основное отличие в том, что в задачах первого класса нужно найти места расположения объектов, связи между которыми считаются установленными изначально. В задачах второго типа связи устанавливаются в результате их решения. В работе рассматривается классический представитель класса задач размещения-распределения – простейшая задача размещения.

Цель и задачи

Целью данной работы является разработка эффективных алгоритмов для решения простейшей задачи размещения и создание мобильного приложения для решения простейшей задачи размещения использующее разработанные алгоритмы.

Для достижения указанной цели необходимо решить следующие задачи:

1) произвести анализ требований и постановку задачи;
2) разработать алгоритм решения задачи размещения;
3) произвести обзор программных средств разработки мобильных приложений;
4) произвести проектирование архитектуры приложения;
5) реализовать приложение для платформы IOS использующее разработанные алгоритмы;
6) протестировать разработанное приложение.

Структура и объем работы

Работа состоит из введения, 4 разделов, заключения, списка библиографии. Объем работы составляет 39 страниц, объем библиографии – 16 источников.

Краткое содержание работы

Во введении описаны актуальность исследуемой темы, цели и задачи исследования.

В первой главе описывается простейшая задача размещения, рассмотрены общие понятия генетического алгоритма. Описан разрабатываемый алгоритм решения простейшей задачи размещения.

Во второй главе описаны требования к разрабатываемой системе, приведена диаграмма вариантов использования, описана диаграмма классов.

В третьей главе описаны инструментальные средства разработки, взаимодействие компонентов в разрабатываемом приложении, приведены скриншоты интерфейса.

В четвертой главе проведен вычислительный эксперимент и представлен график, показывающий эффективность разработанного алгоритма в сравнении с классическим.

В заключении перечислены основные результаты работы.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Простейшая задача размещения

Простейшая задача размещения (Simple plant location problem) [8] в классической постановке формулируется следующим образом. Дано конечное множество пунктов возможного размещения предприятий и конечный список клиентов. Предприятия производят некоторый продукт в неограниченном количестве. Известны стоимости размещения предприятий в указанных пунктах и затраты на удовлетворение спроса каждого клиента. Требуется разместить предприятия и закрепить за ними клиентов таким образом, чтобы суммарные затраты являлись минимальными.

Пусть множество $I = \{1, \dots, N\}$ задает перечень возможных пунктов размещения предприятий по производству некоторого однородного продукта. Тогда, для любого пункта размещения $i \in I$ существует величина $c_i \geq 0$, показывающая затраты на открытие предприятия.

Перечень потребителей задается множеством $J = \{1, \dots, M\}$. Для каждой пары i, j известна величина $g_{ij} \geq 0$ затрат на производство и доставку продукции потребителю. Задача состоит в том, чтобы найти такое множество открываемых предприятий $S \subseteq I$, $S \neq \varnothing$, которое с минимальными затратами позволяет удовлетворить потребности всех потребителей. С использованием введенных обозначений математическая формулировка задачи может быть записана следующим образом:

$$F(S) = \sum_{i \in S} c_i + \sum_{j \in J} \min_{i \in S} g_{ij} \to \min_{S \subseteq I} \quad . \qquad (1)$$

1.1. Общие понятия генетического алгоритма

Генетические алгоритмы

Генетические алгоритмы [4] предназначены для решения задач оптимизации. Примером подобной задачи может служить обучение нейронной сети [12], то есть подбора таких значений весов, при которых достигается минимальная ошибка. При этом в основе генетического алгоритма лежит

метод случайного поиска [5]. Основным недостатком случайного поиска является то, что нам неизвестно сколько понадобится времени для решения задачи. Для того, чтобы избежать таких расходов времени при решении задачи, применяются методы, проявившиеся в биологии. При этом используются методы открытые при изучении эволюции и происхождения видов. Как известно, в процессе эволюции выживают наиболее приспособленные особи. Это приводит к тому, что приспособленность популяции возрастает, позволяя ей лучше выживать в изменяющихся условиях.

Впервые подобный алгоритм был предложен в 1975 году Джоном Холландом (John Holland) в Мичиганском университете. Он получил название «репродуктивный план Холланда» и лег в основу практически всех вариантов генетических алгоритмов [1]. Алгоритм работы классического генетического алгоритма представлен на рисунке 1.

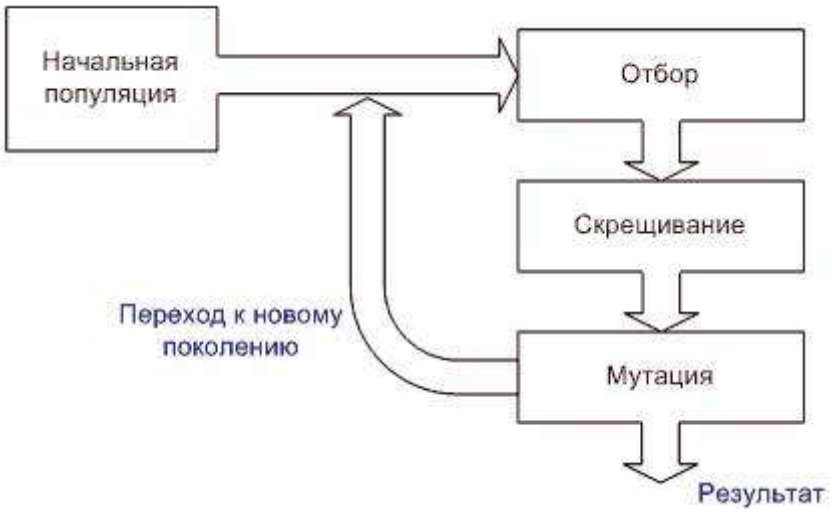

Рис. 1. Принцип работы классического генетического алгоритма

В качестве начальной популяции выбирается случайный набор из N векторов из пространства R^n. Распределение исходной популяции должно выбираться, исходя из особенностей решаемой оптимизационной задачи – обычно используется выборка из n-мерного равномерного или нормального распределения с заданными математическим ожиданием и дисперсией, хотя

существуют другие более сложные способы выборки с учетом особенностей конкретной решаемой задачи.

На этапе отбора нужно из всей популяции выбрать определённую её долю, которая останется «в живых» на этом этапе эволюции. По итогам отбора из N особей популяции H должны остаться M особей, которые войдут в итоговую популяцию L. Остальные особи погибают.

Скрещивание в разных алгоритмах определяется по-разному – оно зависит от представления данных. Главное требование к скрещиванию – чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

К мутациям относится все то же самое, что и к скрещиванию: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать m(N) особей, а затем изменить их в соответствии с заранее определёнными операциями мутации.

Генетические алгоритмы оптимизации являются алгоритмами случайно-направленного поиска и применяются в основном там, где:

- сложно сформулировать задачу в виде, пригодном для более быстрых алгоритмов локальной оптимизации (например, для градиентных алгоритмов, где возможно "мгновенное" вычисление градиента сложной функции, представленной в виде искусственной нейронной сети, с помощью алгоритма обратного распространения ошибки);
- необходимо оптимизировать запросы в базах данных;
- используется задача компоновки:
- используется задача размещения;
- необходима настройка и обучение искусственной нейронной сети.

Представление объектов

В наиболее часто встречающейся разновидности генетического алгоритма для представления генотипа объекта применяются битовые строки. При этом каждому атрибуту объекта в фенотипе соответствует один ген в

генотипе объекта. Ген представляет собой битовую строку, чаще всего фиксированной длины, которая представляет собой значение этого признака.

Основные генетические операторы

Как известно в теории эволюции важную роль играет то, каким образом признаки родителей передаются потомкам. В генетических алгоритмах за передачу признаков родителей потомкам отвечает оператор, который называется скрещивание (его также называют кроссовер или кроссинговер). [3] Этот оператор определяет передачу признаков родителей потомкам. Действует он следующим образом.

1. Из популяции выбираются две особи, которые будут родителями.

2. Потомок определяется как конкатенация части первого и второго родителя.

Также, существует генетический оператор мутации, предназначенный для того, чтобы поддерживать разнообразие особей с популяции. При использовании данного оператора каждый бит в хромосоме с определенной вероятностью инвертируется, т.е., значение бита подменяется на противоположное.

Кроме того, используется еще и так называемый оператор инверсии, который заключается в том, что хромосома делится на две части, и затем они меняются местами.

Для функционирования генетического алгоритма достаточно этих двух генетических операторов, но на практике применяют еще и некоторые дополнительные операторы или модификации этих двух операторов. Кроме того, в некоторых реализациях алгоритма оператор мутации представляет собой инверсию только одного случайно выбранного бита хромосомы.

Обобщенная схема функционирования генетического алгоритма

Схема функционирования генетического алгоритма [2] в виде последовательности шагов (в классическом варианте) представлена ниже.

1. Инициировать начальный момент времени $t=0$. Случайным образом сформировать начальную популяцию, состоящую из k особей. $B_0 = \{A_1, A_2, \ldots, A_k\}$.

2. Вычислить приспособленность каждой особи $F(A_i) = fit(A_i)$, $i=1 \ldots k$ и популяции в целом $F(t) = fit(B(t))$ (также иногда называемую термином фитнес-функция). Значение этой функции определяет насколько хорошо подходит особь, описанная данной хромосомой, для решения задачи.

3. Выбрать особь A_{c2} из популяции. $A_{c1} = Get(B(t))$.

4. С определенной вероятностью (вероятностью кроссовера) выбрать вторую особь из популяции $A_{c1} = Get(B(t))$ и произвести оператор кроссовера $Ac = Crossing(A_c, A_{c1})$.

5. С определенной вероятностью (вероятностью мутации) выполнить оператор мутации $A_c = mutation(A_c)$.

6. Поместить полученную хромосому в новую популяцию $insert(B(t)+1, A_c)$.

7. Выполнить операции, начиная с пункта 3, k раз.

8. Увеличить номер текущей эпохи $t=t+1$.

9. Если выполнилось условие останова, то завершить работу, иначе переход на шаг 2.

1.2. Разработка алгоритма для решения простейшей задачи размещения

В данной выпускной квалификационной работе рассматривается простейшая задача размещения предприятий относительно клиентов. Для сравнения результата было решено реализовать три генетических алгоритма [15]:

– простейший генетический алгоритм без мутации (тип алгоритма – 1);

– генетический алгоритм без мутации, с применением локального поиска (тип алгоритма – 2);

– генетический алгоритм с применением локального поиска и малым процентом мутации (тип алгоритма – 3).

Рассмотрим подробнее принцип работы каждого типа генетического алгоритма из трех описанных раннее. [6], которые необходимо реализовать.

Пошагово опишем алгоритмы.

Перед началом алгоритма необходимо произвести подготовку данных.

Необходимо просчитать кратчайшие пути от клиента до ближайшего поставщика.

Необходимо инициализировать переменную Рекорд и присвоить ей значение 0.

Инициировать начальный момент времени $t=0$. Случайным образом сформировать начальную популяцию, состоящую из 30 особей $B = \{A_1, A_2, \dots, A_{30}\}$ [2].

Рассмотрим первый тип алгоритма – простейший генетический алгоритм без мутации и локального поиска.

Шаг 1. Просчитать стоимость решения для каждого поставщика и клиента по следующей формуле (2).

Шаг 2. Присвоить переменной Рекорд значение минимального решения.

Шаг 3. Выбрать пять случайных решений из популяции.

Шаг 4. Выбрать два самых дешевых решения из пяти, отобранных на шаге 3.

Шаг 5. С вероятностью в 30% произвести операцию скрещивания, в результате которого получатся два новых решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Шаг 6. Отсортировать массив из 30 решений по стоимости.

Шаг 7. Два самых дорогих решения заменить на два получившихся.

Шаг 8. Если обновление переменной Рекорд не происходило на протяжении 20 итераций – закончить работу алгоритма, в противном случае, перейти к Шагу 3.

Исходя из проведенного вычислительного эксперимента, результаты которого описаны в главе 4, алгоритм первого типа недостаточно эффективен. Для улучшения результата добавляем оператор локального поиска.

Рассмотрим второй тип алгоритма – генетический алгоритм без мутации, с применением локального поиска.

За основу взят алгоритм первого типа и добавлен оператор локального поиска.

Алгоритмы локального поиска — группа алгоритмов, в которых поиск ведется только на основании текущего состояния, а ранее пройденные состояния не учитываются и не запоминаются. Основной целью поиска является не нахождение оптимального пути к целевой точке, а оптимизация некоторой целевой функции. Алгоритм локального поиска представлен на рис. 2.

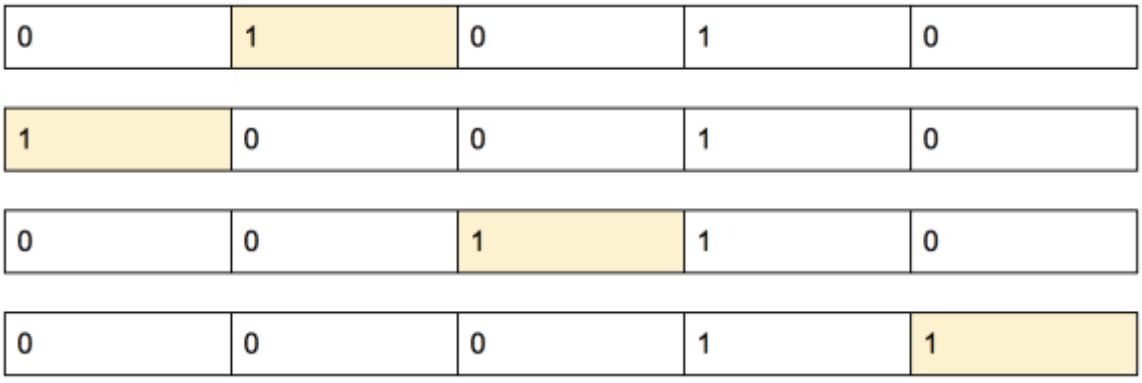

Рис. 2. Локальный поиск

Добавляется еще один шаг после скрещивания. Произвести локальный поиск [9], путем попыток сдвига значимых единиц на места незначимых нулей. Пересчитать решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Исходя из вычислительного эксперимента, результаты которого описаны в главе 4, алгоритм второго типа показывает более качественный результат. Появляется большая вероятность попадания в локальный минимум и зацикливания алгоритма, для предотвращения этого и получения более качественного решения добавляем оператор мутации, положительный эффект которого подтверждается во многих работах [4, 6, 15].

Рассмотрим алгоритм третьего типа – генетический алгоритм с применением мутации и локального поиска.

Добавляется еще один шаг после локального поиска. С вероятностью в 5% произвести мутацию путем попытки замены генов хромосом на противоположные. Пересчитать получившиеся решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Пошагово опишем получившийся алгоритм третьего типа.

Шаг 1. Просчитать стоимость решения для каждого поставщика и клиента по следующей формуле (2).

Шаг 2. Присвоить переменной Рекорд значение минимального решения.

Шаг 3. Выбрать пять случайных решений из популяции.

Шаг 4. Выбрать два самых дешевых решения из пяти, отобранных на шаге 3.

Шаг 5. С вероятностью в 30% произвести операцию скрещивания, в результате которого получатся два новых решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Шаг 6. Произвести локальный поиск [9], путем попыток сдвига значимых единиц на места незначимых нулей. Пересчитать решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Шаг 7. С вероятностью в 5% произвести мутацию путем попытки замены генов хромосом на противоположные. Пересчитать получившиеся решения. В случае, если стоимость новых решений оказалась дешевле, чем Рекорд – обновить Рекорд.

Шаг 8. Отсортировать массив из 30 решений по стоимости.

Шаг 9. Два самых дорогих решения заменить на два получившихся.

Шаг 10. Если обновление переменной Рекорд не происходило на протяжении 20 итераций – закончить работу алгоритма, в противном случае, перейти к Шагу 3.

Формула расчета стоимости решения выглядит следующим образом.

$$Cost = Turnover * \text{Distance} + Bu\sin essOpenCost, \quad (2)$$

где: *Cost* – стоимость решения;

Turnover – грузооборот клиента;

Distance – дистанция до клиента;

BusinessOpenCost – стоимость открытия предприятия – поставщика.

1.3. Выводы

В данной главе была описана постановка задачи (простейшей задачи размещения), исследуемой в данной выпускной квалификационной работе.

Рассмотрены генетические алгоритмы, основные представления объектов в них, а также, генетические операторы. Приведена схема функционирования генетического алгоритма в классическом представлении. Приведено описание алгоритмов, подлежащих разработке. Рассмотрены три типа алгоритма, которые были выведены исходя из вычислительного эксперимента.

2. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

В данном разделе описано проектирование разрабатываемого приложения, выявлены основные требования, разработана диаграмма вариантов использования. Представлена диаграмма основных классов приложения, указано их взаимодействие между собой.

2.1. Требования к разрабатываемой системе

При проектировании приложений принято учитывать два основных типа требований:

– функциональные требования – какое поведение должна предлагать система;

– нефункциональные требования – особое свойство или ограничение, накладываемое на систему.

В результате общения с заказчиком (в роли заказчика выступает научный руководитель), были выявлены основные функциональные и нефункциональные требования к создаваемому мобильному приложению.

Функциональные требования:

– при запуске приложения пользователю должен быть предоставлена краткая информация о решаемой задаче;

– пользователь должен иметь возможность создавать новую задачу;

– пользователь должен иметь возможность редактировать список клиентов;

– пользователь должен иметь возможность редактировать список поставщиков;

– пользователь должен иметь возможность создавать карту с поставщиками;

– пользователь должен иметь возможность создавать карту с клиентами;

– пользователь должен иметь возможность указывать клиентов на карте, тем самым задавая их местоположение;

– пользователь должен иметь возможность указывать поставщиков на карте, тем самым задавая желаемое место размещения предприятия;

– пользователь должен иметь возможность указать необходимые данные о стоимости открытия предприятия - поставщика в указанном на карте месте;

– пользователь должен иметь возможность указать необходимые данные о грузообороте клиента при добавлении клиента на карту.

Нефункциональные требования:

– приложение должно функционировать на распространенной операционной системе компании Apple: iOS;

– мобильное приложение должно поддерживаться устройствами iPhone Начиная с версии 5s, устройствами iPad начиная с версии Air.

– мобильное приложение должно поддерживаться устройствами с установленной операционной системой iOS, начиная с версии 9.0;

– мобильное приложение должно работать только в режиме портретной ориентации экрана;

– в мобильном приложении должен быть задействован шаблон MCV;

– мобильное приложение должно быть разработано на языке программирования Swift;

– оптимальное размещение предприятий не будет рассчитано, если устройство пользователя не имеет выхода в сеть «Интернет». Пользователю будет выведено сообщение «необходимо установить соединение с интернетом»

2.2. Варианты использования системы

В системе можно выделить одного актера, взаимодействующего с приложением. Пользователь – это актер, использующий приложения для вычисления оптимального размещения предприятия по заданным координатам клиентов и поставщиков.

Варианты использования приложения представлены на рис. 3.

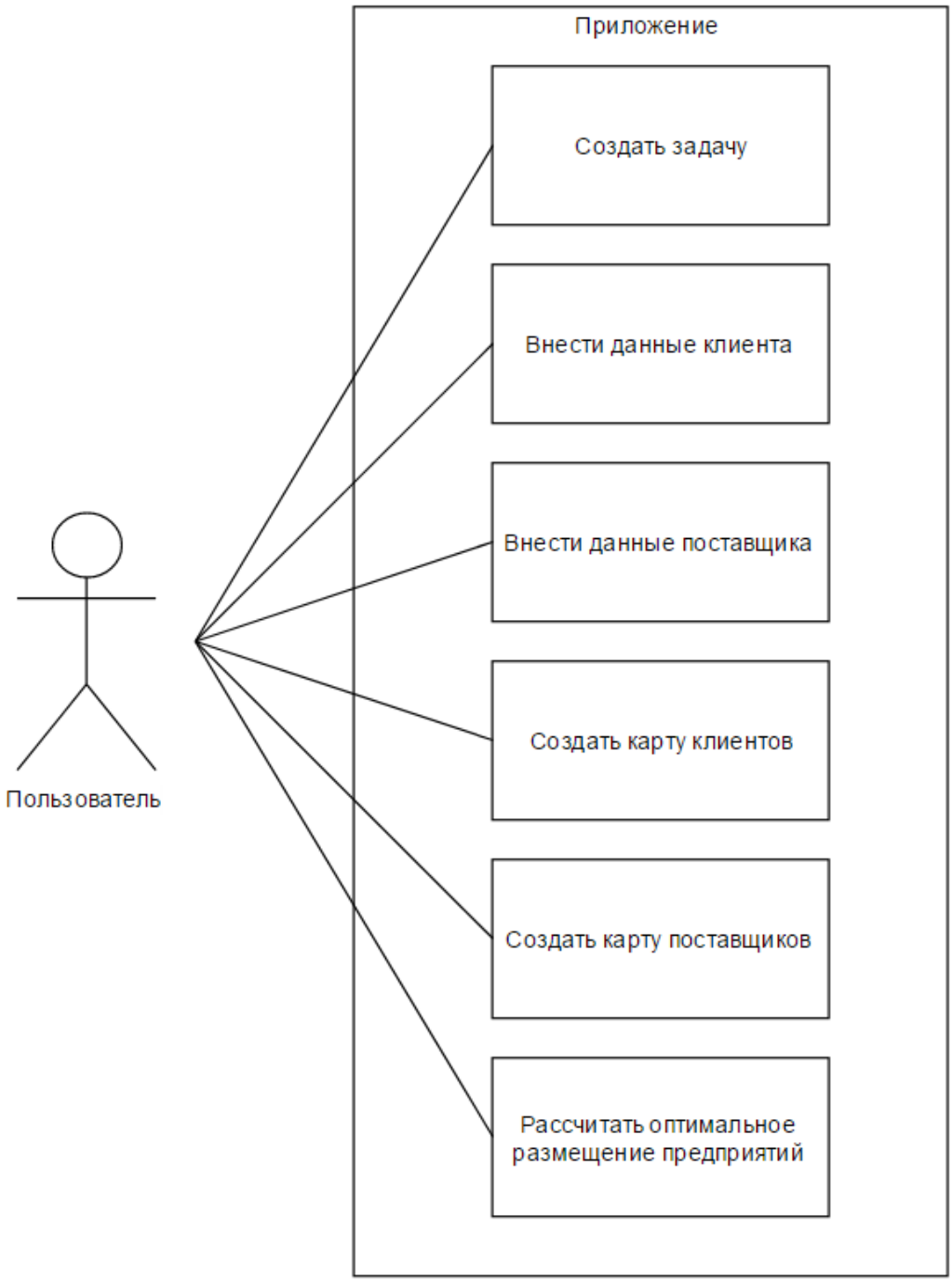

Рис. 3. Диаграмма вариантов использования

В разрабатываемом приложении возможны следующие варианты использования:

– создать задачу – пользователь может создать новую задачу для вычисления оптимального размещения предприятий;

– внести данные клиента – пользователь может внести данные по грузообороту конкретного клиента;

– внести данные поставщика – пользователь может добавить стоимость размещения предприятия относительно указанного местоположения;

– создать карту клиентов – пользователь может создать карту клиентов по внесенным данным о местоположении клиентов и их грузообороте;

– создать карту поставщиков – пользователь может создать карту поставщиков по внесенным данным о местоположении поставщика и затратам на открытие предприятия;

– рассчитать оптимальное размещение предприятий – пользователь может рассчитать оптимальное размещение предприятий на карте относительно клиентов с учетом материальных затрат на открытие и обслуживание, зависящее от грузооборота каждого клиента.

2.3. Диаграмма классов

При проектировании и реализации разрабатываемого приложения была реализована диаграмма классов, представленная на рис. 4.

На диаграмме представлены основные классы мобильного приложения:

Solution – класс, создающий изначальную популяцию решений, считающий изначальную стоимость всех решений.

CalculateController – класс, производящий все вычисления алгоритма. Класс использует модель Solution. Получив данные контроллер передает их в модель Crossover.

Crossover – класс, производящий скрещивание. Произведя вычисления модель передает данные в LocalSearch.

LocalSearch – класс, производящий локальный поиск. Произведя вычисления модель передает данные в Mutation.

Mutation – класс, производящий мутацию. Произведя вычисления передает данные в CalculateController.

MapViewController – класс, отображающий полученные данные от котроллера на карте.

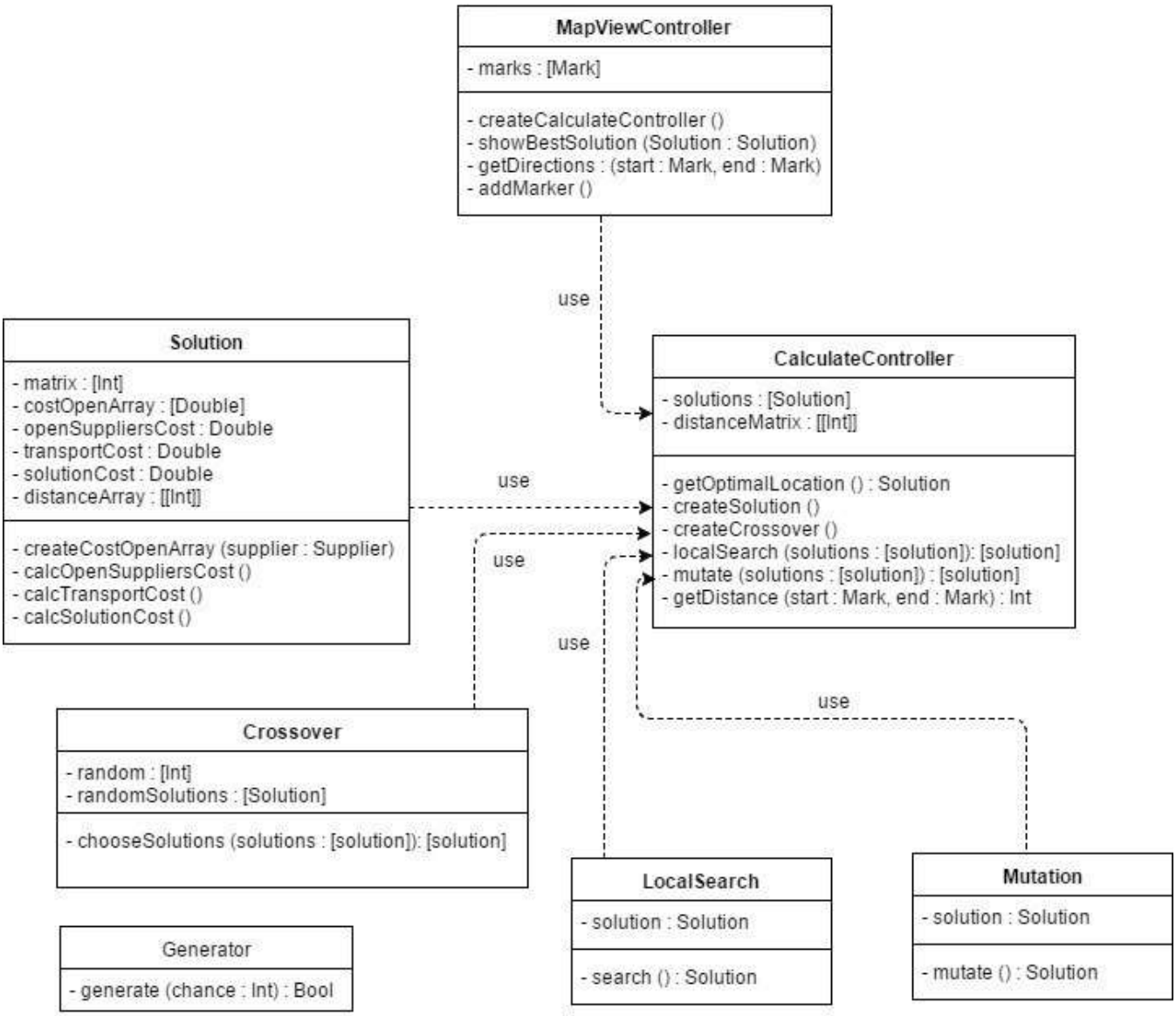

Рис. 4. Диаграмма классов приложения

2.4. Выводы

В данной главе были рассмотрены основные требования к системе, которые были учтены при разработке. В соответствии с требованиями к разрабатываемому приложению была создана диаграмма вариантов использования и диаграмма классов.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В данном разделе описаны особенности разработки приложения, согласно требованиям, описанным в главе 2. Описаны основные инструменты, используемые при реализации, компоненты, из которых состоит приложение.

3.1. Инструменты, используемые при реализации

Для реализации приложения и последующей интеграции в мобильное приложение на платформе iOS был выбран язык программирования Swift и среда разработки XCode.

Среда разработки XCode – интегрированная среда разработки программного обеспечения под платформы MACOS и iOS, разработанная корпорацией Apple [10].

Для обеспечения взаимодействия с картой был использован интерфейс программирования приложений (API) корпорации Google – Google maps.

Язык программирования Swift – открытый мультипарадигмальный компилируемый язык программирования общего назначения. Создан компанией Apple в первую очередь для разработчиков iOS и MACOS [13].

Язык программирования Swift реализует шаблон MVC (model-view-controller, «модель-представление-контроллер») – схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные [16].

Основной идеей шаблона Model-View-Controller (MVC) заключается в том, что контроллер (Controller) и представление (View) зависят от модели (Model), но, в свою очередь, модель никак не зависит от компонентов этих двух компонентов.

3.2. Взаимодействие компонентов

При запуске приложения инициализируются простейшая View, отвечающая за логику работы приложения в главном меню. Далее, если пользователь хочет создать задачу, происходит нажатие на кнопку «Создать задачу», вызывается метод модели Solution CreateTaskTapped. Исходный код метода CreateTaskTapped представлен на рис. 5.

```
@IBAction func createTaskTapped(_ sender: Any) {
    let alertController = UIAlertController(title: "Новая задача", message: "Введите название новой задачи", preferredStyle: .alert)

    let confirmAction = UIAlertAction(title: "Добавить", style: .default) { (_) in
        if let field = alertController.textFields?[0] {
            let task = Task.mr_createEntity()
            task?.name = field.text!
            task?.lastTime = NSDate()
            self.performSegue(withIdentifier: "showFields", sender: task)
            NSManagedObjectContext.mr_default().mr_saveToPersistentStoreAndWait()
            self.tasks = Task.mr_findAll() as! [Task]
            self.savedTasksTable.reloadData()

        } else {
            // user did not fill field
        }
    }

    let cancelAction = UIAlertAction(title: "Отменить", style: .cancel) { (_) in }
    alertController.addTextField { (textField) in
        textField.placeholder = "Название задачи"
    }

    alertController.addAction(confirmAction)
    alertController.addAction(cancelAction)

    self.present(alertController, animated: true, completion: nil)
}
```

Рис. 5. Метод CreateTaskTapped

Двумя основными компонентами, отвечающими за вычислительный процесс и визуализацию, являются CalculateController и MapViewController.

CalculateController включает в себя набор вычислительных методов, при помощи которых происходит подсчет оптимального размещения, скрещивание особей популяции, применение мутаций, применение алгоритма локального поиска.

MapViewController производит все действия, связанные с визуализацией полученных от CalculateController и других контроллеров, данных.

Чтобы рассчитать оптимальное размещение пользователь в основном меню нажимает на кнопку «Рассчитать оптимальное размещение», открывается список карт поставщиков и клиентов. Исходный код CalculateController.ShowBestSolution приведен на рис. 6.

```
func showBestSolution(solution: Solution) {
    let clients = task.clients?.allObjects as! [Client]
    let suppliers = task.suppliers?.allObjects as! [Supplier]

    let cli = clients[clientIndex].marks?.allObjects as! [Mark]
    let supp = suppliers[supplierIndex].marks?.allObjects as! [Mark]
    for i in 0..<cli.count {

        let clientMarkLat = cli[i].lat
        let clientMarkLon = cli[i].lon

        let marker = GMSMarker(position: CLLocationCoordinate2D(latitude: clientMarkLat, longitude: clientMarkLon))
        marker.icon = GMSMarker.markerImage(with: .blue)
        marker.map = map

        for j in 0..<supp.count {

            let supplierMarkLat = supp[j].lat
            let supplierMarkLon = supp[j].lon

            let marker = GMSMarker(position: CLLocationCoordinate2D(latitude: supplierMarkLat, longitude: supplierMarkLon))
            marker.map = map
        }

    }
```

Рис. 6. Функция подсчета оптимального размещения

Классы разделены логически таким образом, что модификация любого из них не требует модификации другого, т.е., модификация каждого класса может осуществляться независимо.

В соответствии с данной схемой разделения данных по шаблону MVC реализованы все остальные классы системы.

3.3. Интерфейс

При запуске приложения пользователь наблюдает главную страницу приложения, на которой изображена краткая информация о задаче, кнопка «Далее» и возможность переключения между задачами (данная опция реализована для будущих исследований и разработок). Главное меню приложения представлено на рис. 7.

При нажатии на кнопку «Далее» открывается страница «Список задач» со списком задач и если созданных задач нет, то пользователю предлагается создать новую задачу. Страница создания задачи и страница просмотра уже существующих задач представлены на рис. 8 и рис. 9.

Рис. 7. Главное меню приложения

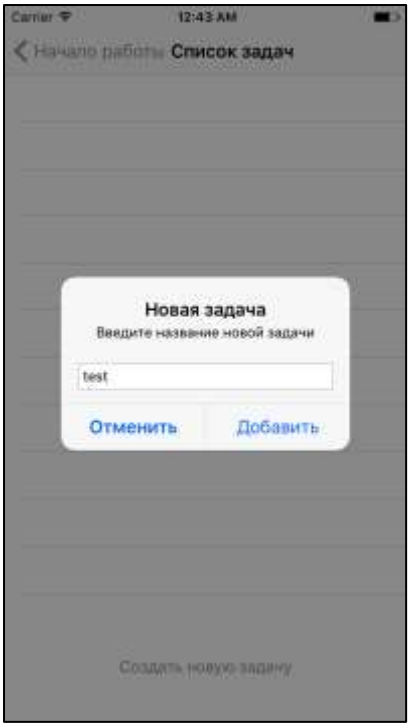

Рис. 8. Создание задачи

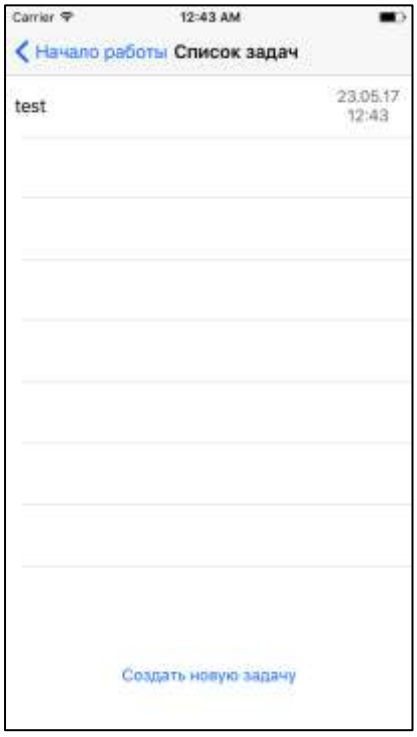

Рис. 9. Просмотр списка задач

После создания задачи при выборе (нажатии) на название задачи открывается меню «Области размещения», которое позволяет совершить ряд следующих действий: редактировать список клиентов, редактирование списка поставщиков, создание карты, расчет оптимального размещения. Интерфейс меню «Области размещения» продемонстрирован на рис. 10.

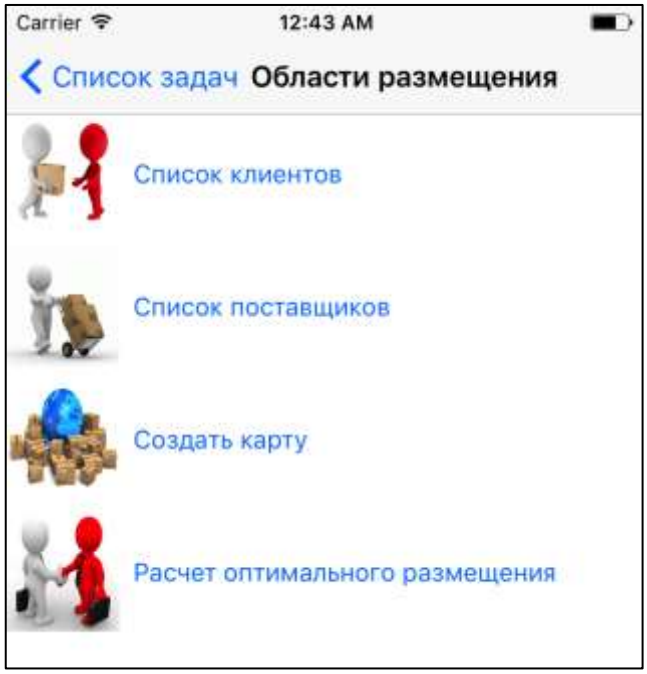

Рис. 10. Области размещения

Для дальнейшей работы с программой пользователю необходимо создать карты поставщиков и клиентов. Работа с картами поставщиков и клиентов происходит следующим образом.

1. Создаем карту поставщиков/клиентов путем нажатия на кнопку «Создать карту».

2. На карте поставщиков/клиентов нажатием в нужное место указываем местоположение поставщика/клиента.

3. Для каждого нового маркера (так обозначается положение поставщика и клиента) указываем стоимость открытия предприятия (для поставщиков) и грузооборот (для клиентов). Формы указания данных для поставщиков и клиентов представлены на рис. 11 и рис. 12.

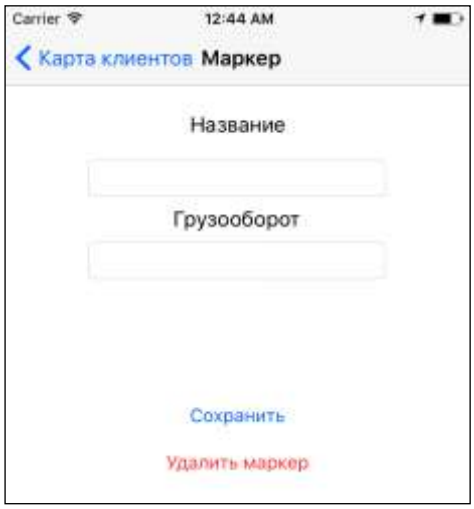

Рис. 11. Создание клиента

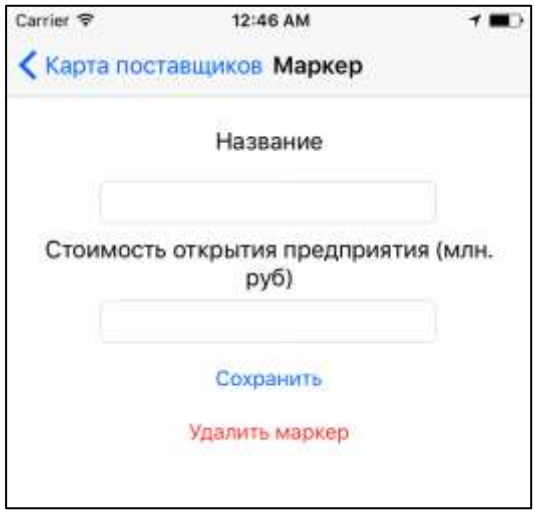

Рис. 12. Создание поставщика

После внесения соответствующих данных о необходимых поставщиках и клиентах существует возможность просмотра карты поставщиков и клиентов в режиме редактирования. Карты поставщиков и клиентов представлены на рис. 13 и рис. 14.

Рис. 13. Карта клиентов

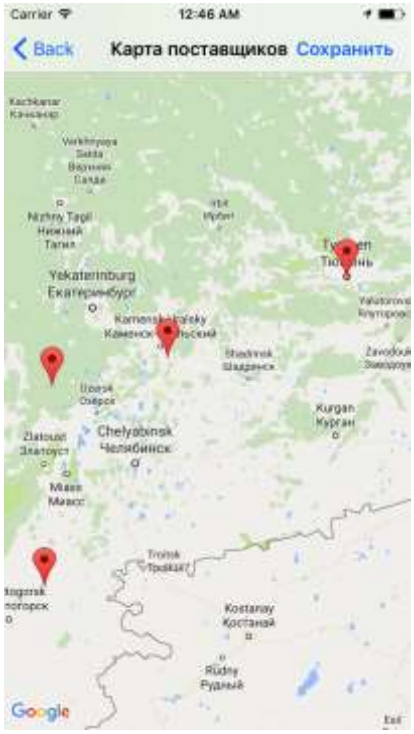

Рис. 14. Карта поставщиков

Для получения от приложения информации о размещении предприятий-поставщиков пользователю необходимо выбрать две карты: карту поставщиков и карту клиентов, на основании которых будет сформирована результирующая карта с оптимальным расположением поставщиков. Выбор будет возможен при нажатии на кнопку «Расчет оптимального размещения». Данная форма представлена на рис. 15.

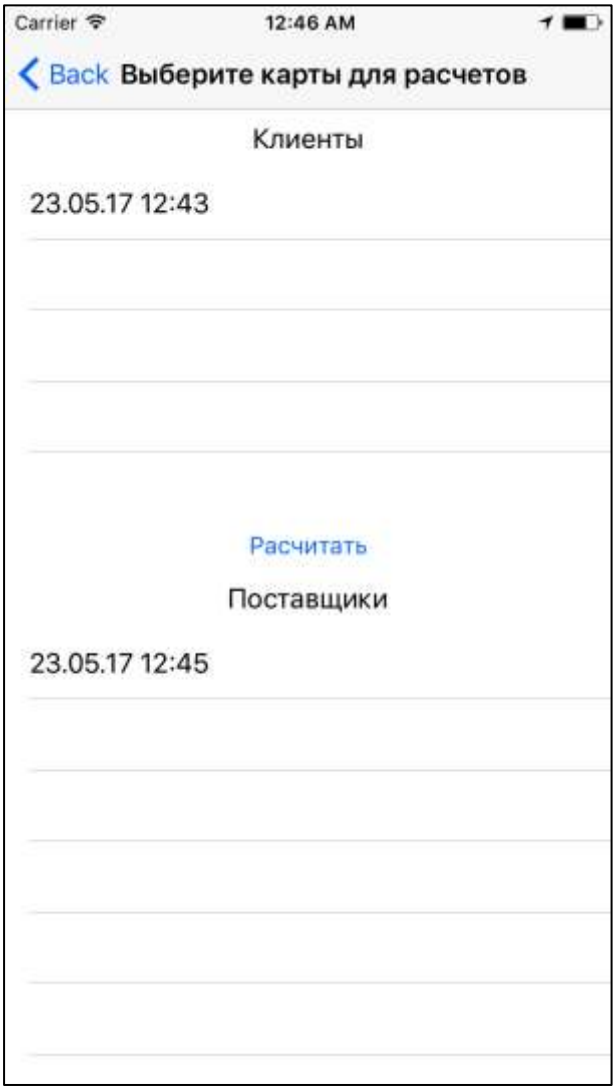

Рис. 15. Выбор карт поставщиков и клиентов

После ввода всей информации и нажатии кнопки «Рассчитать» пользователь получит оптимальные места расположения предприятий поставщиков в соответствии с формулой, приведенной в главе 1. Данная форма представлена на рис. 16.

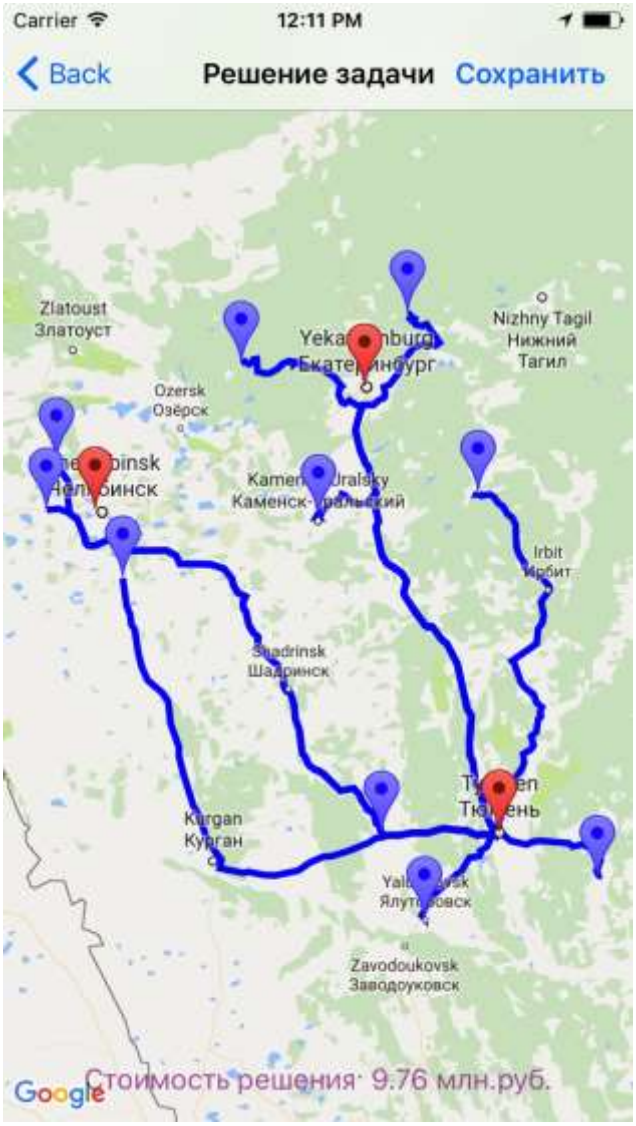

Рис. 16. Результат решения задачи размещения

3.4. Выводы

В данном разделе были описаны особенности разработки приложения, позволяющего рассчитать оптимальное размещение поставщиков относительно клиентов, рассмотрены основные инструменты, используемые при реализации, а также представлены компоненты и модели интерфейса, из которых состоит приложение.

4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов.

4.1. Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения исходным функциональным требованиям заказчика. Проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям. Функциональное тестирование системы представлено в табл. 1.

Табл. 1. Функциональное тестирование системы

№	Название теста	Входные данные	Ожидаемый результат	Полученный результат	Тест пройден?
1	Создать задачу	Пользователь находится на странице списка задач	Открывается окно где пользователь вводит название задачи, новая задача отображается в списке задач	Создается новая задача с заданным пользователем названием и отображается в списке задач	Да

Продолжение табл. 1.

№	Название теста	Входные данные	Ожидаемый результат	Полученный результат	Тест прой ден?
2	Внести данные клиента	Пользователь находится на экране добавления данных	Пользователь вводит данные клиента. (грузооборот) Сохранение данных.	В созданную метку клиента добавляются внесенные данные (грузооборот)	Да
3	Внести данные поставщика	Пользователь находится на экране добавления данных	Пользователь вводит данные поставщика (стоимость открытия), данные сохраняются	В созданную метку поставщика добавляются внесенные данные (стоимость открытия)	Да
4	Создать карту клиентов	Пользователь находится в меню, нажимает кнопку создать карту, выбирает «карта клиентов»	Создается пустая карта клиентов	Была создана пустая карта клиентов	Да
5	Создать карту поставщиков	Пользователь находится в меню, нажимает кнопку создать карту, выбирает «карта поставщиков»	Создается пустая карта поставщиков	Была создана пустая карта поставщиков	Да

Окончание табл. 1.

№	Название теста	Входные данные	Ожидаемый результат	Полученный результат	Тест пройден?
6	Рассчитать оптимальное размещение	Пользователь находится на экране выбора карт клиентов и поставщиков	Пользователь выбирает нужную карту клиентов и нужную карту поставщиков, нажимает на кнопку рассчитать, открывается экран с картой на которой отображено оптимальное размещение	Отобразилась карта с оптимальным размещением	Да

4.2. Нефункциональное тестирование

Нефункциональное тестирование включает в себя тестирование нефункциональных требований к системе. Нефункциональное тестирование системы представлено в табл. 2.

Табл. 2. Нефункциональное тестирование системы

№	Название теста	Входные данные	Ожидаемый результат	Полученный результат	Тест пройден?
1	Отправление запроса при отсутствии интернета	Пользователь находится на экране выбора карт клиентов и поставщиков.	Оптимальное размещение не будет рассчитано, будет выведено	Оптимальное размещение не было рассчитано, было выведено	Да

Окончание табл. 2.

№	Название теста	Входные данные	Ожидаемый результат	Полученный результат	Тест пройден?
		Выбирает нужную карту клиентов и нужную карту поставщиков, нажимает на кнопку рассчитать при отсутствии интернета	сообщение «необходимо установить соединение с интернетом»	сообщение «необходимо установить соединение с интернетом»	

4.3. Вычислительный эксперимент

Как было описано в главе 1, в данной работе подлежали реализации три вариации генетического алгоритма решения простейшей задачи размещения.

В ходе проведения вычислительных экспериментов удалось получить реальные данные, которые показывают, что применение к классическому алгоритму метода локального поиска и мутации позволяют найти более оптимальное решение.

Сравнительный график поиска наиболее оптимального решения приведен на рис. 17.

На графике показаны три вариации алгоритма:

– простейший генетический алгоритм (тип алгоритма – 1);

– генетический алгоритм с применением локального поиска (тип алгоритма – 2);

– генетический алгоритм с применением локального поиска и малым процентом мутации (тип алгоритма – 3)

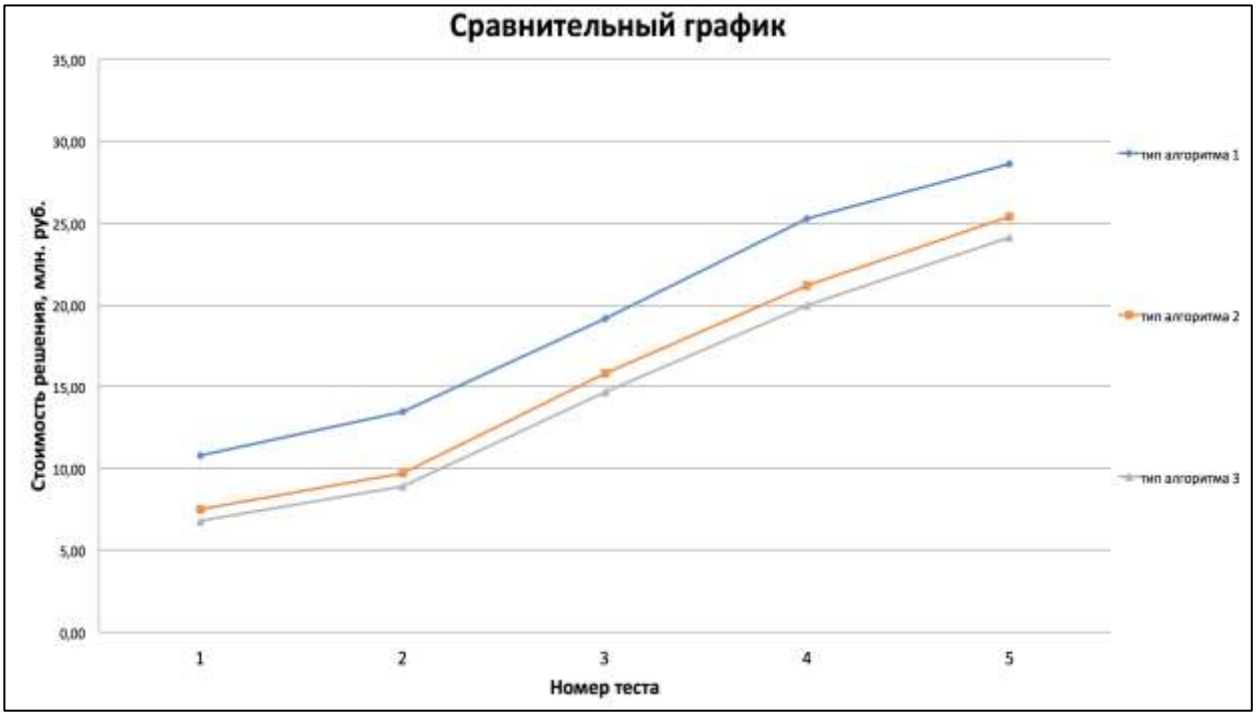

Рис. 17. График найденного решения в зависимости от применённого алгоритма

Для сравнения эффективности разработанного алгоритма полученные данные были сопоставлены с данными рассчитанными в IBM ILOG CPLEX и вычислена средняя относительная погрешность от оптимума.

Табл. 3. Сравнение результатов решения

Размерность задачи	**Простейший генетический алгоритм**	**Генетический алгоритм с применением локального поиска**	**Генетический алгоритм с применением локального поиска и мутации**	**IBM ILOG CPLEX**
5x5	9%	5.5%	3.3%	0%
10x10	8.3%	5.1%	3.6%	0%

Окончание табл. 3.

Размерность задачи	**Простейший генетический алгоритм**	**Генетический алгоритм с применением локального поиска**	**Генетический алгоритм с применением локального поиска и мутации**	**IBM ILOG CPLEX**
15x15	8.1%	4.9%	4%	0%
20x20	8.8%	5%	3.4%	0%

Вычислительный эксперимент показал, что на больших объемах данных алгоритм третьего типа получал значения в среднем очень близкие к оптимуму.

Табл. 4. Время работы алгоритмов

Размерность задачи	**Простейший генетический алгоритм**	**Генетический алгоритм с применением локального поиска**	**Генетический алгоритм с применением локального поиска и мутации**
10x10	0,5 с.	0,5 с.	0,5 с.
50x50	0,6 с.	0,8 с.	0,8 с.
100x100	1 с.	2 с.	2 с.
200x200	4 с.	5 с.	5,5 с.
300x300	6 с.	7 с.	7 с.
400x400	8,7 с.	10 с.	10 с.
500x500	12 с.	14 с.	14,8 с.

Решение задач с большой размерностью было выведено за приемлемое время.

4.4. Выводы

В результате проведенного тестирования было установлено соответствие разработанного приложения исходным функциональным и нефункциональным требованиям заказчика, произведен вычислительный эксперимент, в ходе которого была подтверждена эффективность разработанного алгоритма. Генетический алгоритм с применением локального поиска и мутации получает значение в среднем очень близкие к оптимуму за приемлемое время.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были решены следующие задачи.

1. Разработаны и реализованы три типа алгоритма для решения простейшей задачи размещения.

2. Проведены вычислительные эксперименты по анализу эффективности реализованных в данной выпускной квалификационной работе алгоритмов.

3. Реализовано мобильное приложение для решения простейшей задачи размещения, использующее реализованные математические алгоритмы.

Перспективным развитием является распараллеливание вычислений с целью решения задач большой и сверхбольшой размерности.

ЛИТЕРАТУРА

1. Holland J.N. Adaptation in Natural and Artificial Systems. – Michigan: Univ. Michigan Press, 1975. – 207 p.

2. Solving the simple plant location problem by genetic algorithm. [Электронный ресурс] UML: https://www.researchgate.net/publication/2488533_Solving_The_Simple_Plant_Location_Problem_By_Genetic_Algorithm (дата обращения: 20.05.2017).

3. Ананий В. Левитин Алгоритмы: введение в разработку и анализ. – М.: Изд.дом «Вильямс», 2009. – 576 с.

4. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы: Учебное пособие. – 2-е изд. – М.: Физматлит, 2006. – 400 с.

5. Ежов А.А., Шумский С.А. Нейрокомпьютинг и его применения в экономике и бизнесе. – М.: МИФИ, 1998. – 222 с.

6. Емельянов В.В., Курейчик В.В., Курейчик В.М. Теория и практика эволюционного моделирования. – М.: Физматлит, 2003. – 432 с.

7. Еремеев А.В. Разработка и анализ генетических и гибридных алгоритмов для решения задач дискретной оптимизации. Дисс. канд.физ.-мат.наук. Омск, 2000. – 119 с.

8. Козлов В.Н. Системный анализ, оптимизация и принятие решений: Учебное пособие. – СПб.: Государственный Политехнический университет, 2013. – 174 с.

9. Колоколов А.А. Применение регулярных разбиений в целочисленном программировании. // Известия вузов. Математика, 1993. – С. 11-30.

10. Кочетов Ю.А. Методы локального поиска для дискретных задач размещения. – Новосибирск: Российская Академия Наук, 2009. – 249 с.

11. Нойбург М. Программирование для iOS 7. Основы Objective-C, Xcode и Cocoa = iOS 7 Programming Fundamentals: Objective-C, Cocoa, and Xcode Basics. – М.: «Вильямс», 2014. – 384 с.

12. Рассел Дж. Генетический алгоритм. – М.: VSD, 2012. – 94 с.

13. Скобцов Ю.А. Основы эволюционных вычислений. – Донецк: ДонНТУ, 2008. – 326 с.

14. Усов В. Swift: Основы разработки приложений под iOS. – СПб.: Питер, 2016. – 304 с.

15. Хайкин С., Нейронные сети: Полный курс. – М.: Вильямс, 2006. – 1104 с.

16. Чедвик Дж. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC = Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC. – М.: «Вильямс», 2013. – 432 с.