МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ ПОИСКА ЦЕНТРАЛЬНЫХ ВЕРШИН В ГРАФАХ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Власова Андрея Александровича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных компьютерных науках одно из центральных мест занимает математическая модель графа. Графовая модель позволяет описывать целый ряд систем и явлений, которые встречаются в различных предметных областях. Изучение структуры того или иного графа, или же выявление его свойств и особенностей может приносить невероятную практическую пользу. В связи с этим работы, посвященные этой теме, вызывают наибольший интерес как со стороны исследователей теоретиков, так и со стороны практиков.

Одной из важных характеристик любого графа можно назвать радиус графа и расположение его центральных вершин. Получив данные параметры, можно судить об общей структуре графа или же о его особенностях. Современные графовые модели могут насчитывать десятки сотен тысяч вершин, поэтому для эффективного решения задач зачастую бывает недостаточно использовать классические алгоритмы, а необходимо реализовать некоторый эвристический подход, одним из которых является генетический алгоритм.

Основной целью работы было следующее — разработать и исследовать генетический алгоритм, способный решать задачу поиска центральных вершин. Для достижения этой цели были поставлены следующие задачи:

- реализовать идеи генетических алгоритмов с учетом рассматриваемой задачи,
- реализовать существующие алгоритмы для поиска центральных вершин,
- создать программное приложение, позволяющее проводить запуски алгоритма с различными параметрами,
- исследовать параметры алгоритма и выявить его слабые и сильные стороны.

1 Описание задачи и алгоритмы для ее решения

Для описания задачи сначала необходимо дать формальное определение понятию граф. Граф — это упорядоченная пара множеств (V, E), где V — множество вершин графа, а E — множество упорядоченных и неупорядоченных пар вершин — дуг или ребер. В случае, когда вершины в парах упорядочены, говорят, что граф является ориентированным, иначе — неориентированным.

В случает неориентированного графа также используется понятие связности графа — граф является связным, если между любой парой вершин существует по крайней мере один путь. Кроме этого графы можно разделить на взвешенные или невзвешенные — в случае взвешенного графа каждое ребро имеет некоторый вес — положительное или отрицательное число, в случае невзвешенного графа каждое ребро имеет вес равный единице.

В данной работе рассматривается задача поиска центральных вершин в графах. Для начала можно дать определение эксцентриситета вершины. Эксцентриситетом вершины называется максимальное из расстояний от этой вершины до всех остальных вершин в графе. С использованием этого определения можно сказать, что центральными вершинами в графе называются вершины с минимальным значением эксцентриситета, при чем само значение этого минимального эксцентриситета представляет собой радиус графа.

Радиус графа и эксцентриситет вершины являются одними из базовых параметров, которые наиболее часто встречаются в прикладных задачах, либо необходимы при исследовании свойств графа. Этим объясняется довольно большое количество работ, в которых изучается данная задача. Для решения данной задачи разработаны различные подходы. С одной стороны существуют точные алгоритмы, которые позволяют находить центральные вершины, используя различные эвристики или способы представления решения, с другой стороны существуют стохастические подходы для решения все той же задачи.

К последним можно отнести генетические алгоритмы, которые моделируют эволюционное развитие популяции. Данное развитие реализуется за счет процесса мутации, скрещивания и естественного отбора. Работа алгоритма заключается в итерационном применении этих трех составляющих к популяции, которая описывает набор возможных решений. При этом, так как данные алгоритмы являются вероятностными, то не гарантируется, что найденное решение будет всегда верным.

2 Разработка генетического алгоритма и его исследование

Предлагаемый в данной работе генетический алгоритм реализует следующую идею: пусть существует некоторое абстрактное или реальное изображение графа, причем в центре этого изображения находятся центральные вершины графа. Тогда, если популяция генетического алгоритма представляет собой набор вершин, то этот набор может быть представлен в виде некоторого «шара», внутри которого находится центральные вершины графа. Из этой идеи следует, что для нахождения центральных вершин необходимо, чтобы этот абстрактный шар сжимался к центру. Именно этот смысл заложен в работу оператора скрещивания. В добавок к этому алгоритм должен сжимать эту «сферу» к глобальному центру, не сбиваясь в локальные оптимальные значение, за это отвечает оператор мутации. Естественный отбор соответственно занимается выбором вершин с оптимальными эксцентриситетами.

В качестве начальной популяции генерируется случайный набор уникальных вершин в количестве N, причем вероятность попадания в начальную популяцию одинакова для всех вершин. Естественный отбор занимается выбором оптимальных вершин для продолжения работы алгоритма. Для каждой вершины находится ее эксцентриситет при помощи обхода в ширину, после чего методом колеса рулетки, при этом приоритет отдается вершинам с меньшим эксцентриситетом. Скрещивание реализовано следующим образом: в качестве родителей выбираются две вершины из популяции, после чего между ними находится кротчайший путь, после чего из этого пути выбирается одна вершина в качестве потомка. Именно такая реализация данного этапа позволяет алгоритму сходиться к центральным вершинам, причем на работу этого этапа влияет параметр p_c. В процессе работы этапа мутации перебираются все вершины в популяции и для каждой вершины находятся ее соседи — вершины смежные с ней, после чего с вероятностью p_m вершина заменяется на одного из своих соседей.

2.1 Исследование алгоритма

В качестве языка программирования для исследования алгоритма был выбран язык программирования C++. Кроме этого в качестве среды разработки, в которой производилась реализация алгоритма, была выбрана
`Visual Studio 2017`, а вычислительная машина, на которой были выполнены

все испытания обладает оперативной памятью с размером 6.0 GB и процессором AMD A8-7410 с частотой 2.20 GHz.

Кроме этого для запусков тестов алгоритма необходимы наборы графов с разными свойствами. Для создания графов использовалась Python библиотека NetworkX, которая предоставляет возможности для генерации данных на основе хорошо известных моделей случайных графов: графа Эрдеша-Реньи, графа Барабаши-Альберт и геометрического случайного графа.

2.2 Результаты вычислительных экспериментов

Для выявления сильных и слабых сторон предложенного алгоритма его сравнение проводилось с рядом точных алгоритмов, а также с алгоритмом «N4N». Для модели Эрдеша-Реньи в качестве параметра p было выбрано значение 5%, для модели Барабаши-Альберт $m = 2$, а для геометрического случайного графа $r = 0.1$.

Для сравнения по временным результатам были реализованы тривиальный алгоритм и алгоритм с улучшенной асимптотикой. Эти алгоритмы и генетический алгоритм описанный в данной работе запускались на трех моделях случайных графов, с количеством вершин 500, 1000, 1500, 2000, 2500, 5000. Исходя из практических экспериментов в качестве размера популяции выбрано значение 50, для оператора скрещивания 0.7, для оператора мутации 0.1, кроме этого число итераций ограничено числом 20. Из полученных результатов был сделан вывод, что созданный генетический алгоритм дает выигрыш по времени в несколько раз по сравнению с точными алгоритмами.

Также так как эвристический алгоритм не гарантирует получение точного ответа, а допускает некий процент ошибки, то для изучения точности алгоритма были проведены тесты позволяющие выявить процент неправильных ответов. Для этого алгоритм запускался на все тех же графах, при этом так как размерности графа, позволяют за приемлемые временные затраты с помощью точного алгоритма найти центральные вершины, то зная эту информацию можно говорить о проценте неправильных ответов. Для сравнения брался алгоритм «N4N», после чего оба алгоритма запускались 100 раз, что позволило подсчитать процент ошибки.

Из полученных результатов стало ясно, что созданный алгоритм не уступает одному из существующих эвристических алгоритмов.

2.3 Исследование параметров генетического алгоритма

В описанных ранее результатах в качестве значений параметров генетического алгоритма были выбраны значения $N = 50$, $p_c = 0.7$ и $p_m = 0.1$. Однако эти значения выбирались в качестве тестовых для того, чтобы проверить жизнеспособность идей, заложенных в алгоритм. При этом эти параметры ключевым образом влияют как на точность алгоритма, так и на время его выполнения. В связи с этим был также проведен еще ряд экспериментов, в которых исследовались эти параметры. Для начала были произведены запуски, в которых перебирались значения для p_m и p_c от 0 до 1. Для каждого значения этих параметров измерялось время работы алгоритма и его процент ошибок.

Из этих результатов экспериментов было видно, что время работы алгоритма растет по мере увеличения как параметра p_m, так и параметра p_c. Вместе с тем стало ясно, что и процент неверных ответов падает по мере увеличения тех же параметров. Очевидно, что необходимо найти некоторые оптимальные значения для того, чтобы процент неверных ответов был невелик и одновременно с этим необходимо, чтобы время работы было минимальным. Для того чтобы соединить воедино два этих фактора была введена функция:

$$F(pm, pc) = \alpha\, time(pm, pc) + \beta\, error(pm, pc), \tag{1}$$

которая учитывает время работы и процент ошибок, причем параметр α отвечает за уровень значимости временных затрат, а параметр β отвечает за значимость процента ошибок. Эта функция позволила выявить оптимальные значения для параметров p_m, p_c и N. И эксперименты с ее использованием показали, что данные параметры должны выбираться для каждого графа индивидуально, но в целом прослеживается четкая тенденция к тому, что наилучшим образом алгоритм работает с параметрами p_m и p_c в диапазоне 0.4-0.6 и для параметра N в диапазоне 20-30.

3 Описание приложения, созданного для работы с генетическим алгоритмом

Кроме этого для работы с генетическим алгоритмом создано отдельное приложение. Приложение представляет собой веб-сайт, через который предоставляется возможность для работы с генетическим алгоритмом. С одной стороны у пользователя есть доступ к исследованию алгоритма и его запуску с различными параметрами, а с другой пользователь может заняться исследованием собственного графа и запустить алгоритм на нем.

Пользователю предоставляется возможность для работы с несколькими формами, на которых он может либо загружать граф из файла для поиска его центральных вершин либо исследовать параметры алгоритма на уже сохраненных графах, которые лежат в базе данных.

3.1 Описание технологий и архитектуры приложения

В качестве языка программирования для создания проекта был выбран объектно-ориентированный язык C#. Вместе с тем, для создания клиент-серверного приложения был использован фреймворк `ASP.NET MVC 5`, который позволяет создавать веб-приложения с использованием архитектуры `MVC`. Кроме этого в качестве системы объектно-реляционного отображения используется технология `Entity Framework 6`. При этом приложение разделено на три слоя абстракции — уровень доступа к данным, уровень бизнес-логики и уровень визуального представления.

3.2 Структура базы данных

Для создания базы данных используется технология `Entity Framework 6`, который позволяет использовать подход `Code-first`, согласно которому были созданы классы `Graph`, `GraphInfo`, `Edge`, `User`, описывающие модели данных:

После чего фреймворк на основании созданных моделей создал структуру базы данных и связи между таблицами.

3.3 Уровень доступа к данным

Для гибкой и стандартизированной работы с базой данных был создан ряд интерфейсов, в которые были вынесены основные методы для доступа к данным и их изменениям. Классы, реализующие эти интерфейсы представляют собой уровень доступа к данным, при этом использование интерфейсов

позволяет с легкостью изменять реализацию этих классов, а также упрощает процесс тестирования.

Вместе с тем для использования технологии Entity Framework созданы классы GraphContext и UserContext, которые наследуются от класса System.Data.Entity.DbContext, что позволяет получить возможность для легкого доступа к базе данных без написания SQL запросов. Классы GraphContext и UserContext содержат в себе поля типа DbSet<Graph> и DbSet<User>, через которые происходит добавление, чтение или изменение данных в базе данных. Кроме этого в этих классах описан статический конструктор, внутри которого указан способ начальной инициализации базы данных, за счет классов UserContextInitializer и GraphContextInitializer. Эти классы наследуются от класса CreateDatabaseIfNotExists, что позволяет фреймворку выполнить начальное заполнение данными, если база данных еще не существует, за счет кода, который описан в переопределенном методе Seed. Внутри метода, описанного в классе GraphContextInitializer, происходит чтение нескольких созданных графов из текстовых файлов, в методе с этим же именем в классе UserContextInitializer добавляется пользователь с логином admin и паролем admin.

3.4 Уровень бизнес-логики

Уровень бизнес-логики представляет собой похожую структуру, как и уровень доступа данных — так же созданы ряд интерфейсов и классы, которые их реализуют. При этом многие из этих интерфейсов похожи на те, которые описаны в уровне доступа к данным, однако именно на этом уровне происходит запуск генетического алгоритма с различными параметрами и анализ загруженных графов. Классы GraphBL и UserBL, реализуют интерфейсы IGraphBL и IUserBL. Они содержат ссылки на объекты, реализующие интерфейсы IGraphDao и IUserDao. При добавлении нового пользователя происходит проверка на существование записи с таким же логином, а кроме этого происходит хеширование пароля.

При добавлении нового графа в базу данных в классе, который отвечает за работу с моделью графа, происходит проверка графа на связность и проверка на размеры графа. При неудачном прохождении проверки выбрасывается исключение, которое отлавливается на уровне представления.

Кроме этого для работы с генетическим алгоритмом создан интерфейс

IAlgorithm:

Интерфейс определяет набор методов, в которых будет реализована логика для работы с генетическим алгоритмом. Вместе с тем класс Algorithm реализует данный интерфейс и в нем содержится вся логика работы с алгоритмом — получение результатов поиска центральных вершин, замеры времени работы и процента неверно найденных решений.

Результаты измерений возвращаются из методов при помощи классов FindingVertexResponse и ResearchAlgorithmResponse.

3.5 Уровень представления

Так как проект представляет собой веб-приложение, то уровень, отвечающий за пользовательский интерфейс реализован при помощи технологии ASP .NET MVC 5. В связи с чем весь код этого уровня разделен на:

— контроллеры, которые отвечают на HTTP запросы клиента с помощью представлений,
— представления, которые написаны с использованием технологии Razor, позволяющей внедрять серверный C# код,
— модели данных, внутри которых происходит передача данных от клиента серверу и обратно.

3.5.1 Контроллеры

Для взаимодействия с клиентской частью приложения и обработки пользовательских данных было создано несколько контроллеров: HomeController, GraphController, LoginController, ResearchController.

Класс HomeController содержит один метод Index, который отвечает на GET-запрос и возвращает домашнюю страницу.

Класс GraphController включает в себя методы, определяющие URL-адреса при взаимодействии с которыми клиентской части приложения предоставляется возможность запускать генетический алгоритм для поиска центральных вершин или добавлять новый граф в базу данных. Класс ResearchController содержит методы, через которые пользователю предоставляется возможность запускать генетический алгоритм с различными параметрами (p_c, p_m, N), а также граф на котором будет тестироваться алгоритм.

В классе LoginController определены методы, за счет которых происходит регистрация и аутентификация пользователей. Пользователь, который

вошел в систему получает возможность для добавления новых графов и добавление новых пользователей, залогиненному пользователю предоставляются права администратора.

3.5.2 Модели данных и валидация

Основными классами, с помощью объектов которых происходит передача данным в контроллеры, `AddGraphRequest`, `CreateUserRequest`, `LoginUserRequest`, `ResearchRequest`, при этом результаты вычислений возвращаются из контроллеров в виде представлений, которые представляют собой HTML разметку с внедрением данных, переданных через классы `AlgorithmResultResponse`, `FindingVertexResponse`, `ResearchAlgorithmResponse`.

При этом очевидно, что при введенные пользовательские данные должны удовлетворять некоторым условиям. Для того, чтобы передаваемые данные можно было проверить из любого участка кода для некоторых свойств были использованы атрибуты валидации `Required`, `Compare`, `StringLength`:

При таком использовании атрибутов валидации проверить модель на соответствие выдвинутым требованиям можно при помощи следующего кода:

внутри любого из контроллеров, где `ModelState` — свойство класса `Controller`, которое инкапсулирует состояние модели, переданной в качестве параметра запроса. В случае неудачного прохождения валидации в свойство `ModelState` при помощи метода `AddModelError` добавляется сообщение об ошибке, которое затем будет вставлено в HTML разметку. Атрибут `Requred` установлен для логина и пароля, вводимого пользователем, что гарантирует тот факт, что в базу данных не будет помещена запись с пустыми полями. В добавок к этому у свойства `ConfirmPassword` установлен атрибут `Compare`, который требует, чтобы свойство, отвечающее за хранение пароля, совпадало со свойством, отвечающим за хранение повтора пароля. Также используется атрибут `StringLength`, в котором устанавливаются минимальная и максимальная длина логина и пароля.

3.5.3 Аутентификация

Как уже отмечалось ранее доступ к возможности добавлять графы в базу данных и добавлять туда же новых пользователей имеют доступ только пользователи, которые вошли в систему. В связи с этим в качестве техноло-

гии аутентификации в созданном приложении используется аутентификация с помощью форм. При успешном прохождении проверки на принадлежность пользователю введенного им пароля, при помощи следующей строки кода клиентская часть получает cookie-файлы, которые затем будут присоединяться ко всем остальным запросам:

Для того, чтобы к определенным методам был доступ только авторизированным пользователям к каждому методу применяется атрибут `Authorize`, который гарантирует проверку на доступность для пользователя этих методов. При этом для пользователя вошедшего в систему несколько изменяется HTML разметка, что достигается при помощи использования свойства `User.Identity.IsAuthenticated`.

3.6 Хеширование паролей

Очевидно, что хранение паролей пользователей в открытом виде представляет собой подход нарушающий основные требования к безопасности приложения. В связи с чем каждый пароль при регистрации пользователя хешируется и полученный хеш сохраняется в базе данных. Хеширование паролей происходит в классе `Encryption`, где определены публичные методы `CreatePassword` и `CheckPassword`.

В конструктор класса передается строка с паролем, «соль» — псевдослучайная последовательность байт, которая используется для повышения криптоустойчивости хеша и параметр, отвечающий за искусственную временную задержку, которая позволяет избежать попытки грубого перебора. В базу данных сохраняется полученный хеш и сгенерированная «соль». При проверке подлинности пароля из базы данных извлекается хеш с «солью», после чего введенный пароль хешируется с сохраненной «солью» и результат сравнивается с тем, что было сохранено в базе данных.

3.7 Внедрение зависимостей

Ранее описывались независимые уровни, на которые разделено приложение, при этом гибкость и заменяемость каждого из уровней гарантируется существованием интерфейсов. Каждый из уровней содержит ссылки на объекты, которые реализуют тот или иной интерфейс, при этом эти объекты передаются в качестве параметров в конструкторы. Для того, чтобы гарантировать тот факт, что во все конструкторы будут переданы одни и те же реализации

интерфейсов и избежать дублирования кода в созданном приложении используется IoC-контейнер `Ninject`, который связывает интерфейсы с объектами, которые их реализуют и предоставляет их при необходимости. Связывание интерфейсов и реализации происходит в методе класса `NinjectRegistrations`:

При этом в глобальном файле запуска приложения происходит регистрация этого класса в качестве основного способа разрешения зависимостей.

ЗАКЛЮЧЕНИЕ

В рамках работы поставленная цель была достигнута. Для поиска центральных вершин был предложен генетический алгоритм, а кроме этого созданы программные приложения позволившие его исследовать. Исходя из полученных результатов, можно сказать, что во многом время и качество работы алгоритма зависит от параметров p_m, p_c и N, и при правильно подобранных значениях алгоритм не уступает по своим характеристикам разработанным ранее алгоритмам. Кроме этого создано веб-приложение для работы с генетическим алгоритмом.

Результаты исследований были представлены в статье «Ball-Shrinking Genetic Search Algorithm for Finding Central Vertices in Graphs», на Студенческих научных конференциях факультета КНиИТ СГУ в 2019 и 2020 году, а также на VIII Международной молодежной научно-практической конференции «Математическое и компьютерное моделирование в экономике, страховании и управлении рисками» на базе СГУ.