ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Образовательная программа
«Прикладная математика и информатика»

КУРСОВАЯ РАБОТА

На тему: Различные способы построения графовых структур данных для поиска ближайшего соседа

Тема на английском: Several Ways to Form Graph Based Nearest Neighbour Search Structure

Студент / студентка 2-го курса
группы № БПМИ2110/21ПМИ-2:
Рябков Игорь Дмитриевич
(Ф.И.О.)
Научный руководитель:
Пономаренко Александр Александрович
(Ф.И.О.)
Доцент, НН Кафедра прикладной математики и информатики
(должность, звание)

Содержание

1 Введение

В последнее десятилетие наша жизнь стала тесно связана с удобными приложениями и сервисами. Многие из них базируются на рекомендательных системах, для предоставления целевого товара на основе наших интересов. Некоторые используют компьютерное зрение для решения огромного кол-ва задач (от масок в социальных сетях, до автопилота в электроавтомобилях). Поиск синонимов и системы автоматического дополнения текста(T9) также используются каждый день миллионами пользователей. Это лишь малая часть задач, которые можно решить используюя алгоритм поиска ближайшего соседа (или поиска K-ближайших соседей).

Вот ещё некоторые задачи, о которых хотелось бы упомянуть:

- Поиск дубликатов (определить являются ли 2 текстовых документа одинаковыми)
- Задача кластеризации (Определить, как какой группе относится выбранный объект)
- Поиск ближайших географических объектов (карты)
- Поиск схожих фрагментов в фильмах или музыке

Именно поэтому так важно искать новые подходы для улучшения скорости данного алгоритма. Чтобы достичь поставленную цель, необходимо разработать структуру данных, которая сможет наиболее эффективно осуществлять две операции добавления и поиска. Вариантов подходящих структур - огромное множество. Например, некоторые могут быть построены на базе вектора, списка, дерева, графа (в виде сети). Некоторые поддерживают точные поиск, а некоторые только приближённый. Некоторые формируются по средствам детерминированных алгоритмов, а некоторые используют рандомизированный подход.

Моё исследование буддет в основном основываться на изучении графовых структур (в виде сетей), так как они более современные и эффективные (подробнее ниже). Перед собой я ставлю следующие задачи:

- Изучить какие структуры для поиска ближайшего соседа существуют
- сравнить их асимптотику построения этих структур и поиска внутри них, выявить преимущества и недостатки
- Исследовать феномен тесного мира.
- Обосновать выбор именно графовых структур, а также подробнее исследовать те из них, которые имеют свойствр тесного мира.

- Выдвинуть несколько предположений о модификациях, которые смогли бы улучшить имеющиеся методы.
- Разработать необходимые абстракции для работы с подобными структурами
- На основе сравнительного анализа определить наилучшую конфигурацию параметров в разработанных классах для наибольшей эффективности алгоритма поиска ближайшего соседа
- А также проверить эффективность предложенные ранее модификации

2 Основная часть

2.1 История развития структур для поиска ближайшего соседа

2.2 Феномен тесного мира

Для того, чтобы понять, каким должен быть граф для оптимальной работы нашего алгоритма, я предлагаю обратиться к структурам, которые возникают само собой в природе, к графам, которые формирует общество.

2.2.1 Теория 6 рукопожатий

Венгерский писатель Karinthy Frigyes Ernő в 1928г в рассказе "Звенья цепи"впервые сформулировал данную проблему. Согласно ей, любые два человека на планете связаны через 5-6 общих знакомых.

Рассуждая над этой проблемой, Stanley Milgram - американский социальный психолог и педагог в своей статье "The Small World Problem"описал проведённый им эксперимент: Он выдал 300 писем жителям из разных городов и попросил доставить их одному человеку из Бостона (США). Важным условиям было то, что люди могли передавать письма только своим знакомым, которые по их мнению могли знать целовека-цель. По результатам исследовани даже не смотря на то, что до места назначения дошли далеко не все письма, те, которым это удалось, прошли в среднем через цепь из 5-6 человек.

Данные наблюдения кажутся очень полезными для нас. Ведь если нам удастся воссоздать граф, который с хорошей точностью моделируют общественные сети, мы сможем из любой вершины графа добираться до цели за сравнимо малое число посредников. Причём для этого нам даже не придётся использовать сложные алгоритмы поиска. Вспомним эксперимент Милгрема, в нём каждый человек не пытался искать оптимальный путь от него до цели, он лишь отправлял письма тем знакомым, которые казались ближе к месту назначения. Формально - это простой жадный поиск, который в эксперименте Милгрема прошёлся всего лишь по 5-6 промежуточным значениям и, что не мало важно, достиг цели.

2.2.2 Случайные графы Erdős–Rényi

Попробуем предположить, что общественные сети можно представить, как случайный граф $G(n,p) = V, E$ состоящий из n вершин, с вероятностью проведения ребра p. Логично предположить, что $np = const = \lambda$. Можно объяснить это так: структура графа не должна зависить от кол-ва вершин в нём. Он всегда должен выглядеть примерно одинаково ($deg(v) \approx$

const). Ну или ещё одно описание: кол-во моих друзей не увеличится, если население планеты увеличится в несколько раз. Обоснуем это формально:

$$P(deg(v) = k) = \binom{n}{k} p^k (1-p)^{1-k} \Rightarrow deg(v) \sim Bin(p) \tag{1}$$

Так как $np = const = \lambda : \mathbb{E}[deg(v)] = np = \lambda$

Так мы описали граф, изучением которого занимались два великих Венгерских математика Paul Erdős и Alfréd Rényi. (Причём λ обычно выбирается много меньше чем n, ведь наш круг общения никчёмно мал по сравнению с 8 млрд. людей планете)

Разультаты, к которым пршли данные математики показали, что случайные графы имеют сравнительно малый средний диаметр графа $d(G)$. Где

$$d(G) = \frac{1}{|V|} \sum_{u,v \in V} d(u,v) \tag{2}$$

$$d(u,v) = \min_{k \in \mathbb{R}} \{\exists a_i,\ i = \overline{1k} : a_1 = u, a_k = v, (a_i, a_{i+1}) \in E\} \tag{3}$$

Казалось бы, малый диаметр - всё, что нам нужно. И это было бы так, если бы мы не использовали жадный алгоритм для поиска. Случайные рёбра в графе помогают нам легко и быстро приблизиться к месту назначения, однако отсутствие локальных рёбер приводит к большому кол-ву локальных минимумов, что ведёт за собой большую погрешность полученном результате, пример:

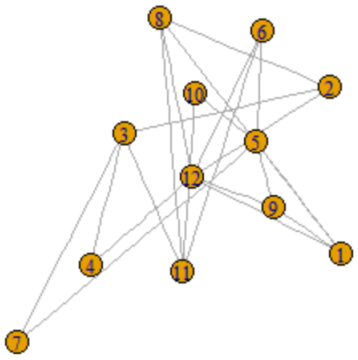

Рис. 1: Случайный граф

Взглянув на рисунок 1, можно заметить, что если мы хотим добраться до вершины 3, используя жадный поиск, то это нам удастся только из непосредственных соседей вершины номер 3. Это говорит о том, что такой граф хорош в навигации на больших масштабах, но плох при локальном поиске. Что не состыкуется с теорией 6 рукопожатий

Ошибка в наших рассуждениях была в том, что мы предположили, что связи между людьми абсолютно случайны, но ведь это не так. Милгрэм, описывая результаты своего

эксперимента, делал акцент на то, что общественные графы состоят из кластеров. Это можно описать так: Вероятность знакомства с человеком тем больше, чем мы ближе; А также, вероятность, того, что люди дружат прямо пропорциональна кол-ву их общих друзей. Наличие данной концепции образовало бы больше локальных рёбер, что решило бы проблему локальных минимумов. Формально это можно описать, через коэффицент кластеризации вершины:

$$cc(v) = \frac{\#\{(x,y) \in E,\ x,y \in V : (x,v),(y,v) \in E\}}{\#\{(x,y),\ x,y \in V : (x,v),(y,v) \in E\}}$$

По аналогии коэффицент кластеризации графа:

$$cc(G) = \frac{1}{|V|} \sum_{v \in V} cc(v)$$

Чем он больше, тем сильнее кластеризован наш граф.

2.2.3 Тесные графы

В рассуждениях выше мы пришли к выводам, что графы в реальном мире обладают двумя основными свойствами: малым диаметром и большим коэффицентом кластеризации. В 1998г, наблюдая за реальными сетями, к тем же самым выводам пришли два американских математика: Дункан Уоттс и Стивен Строгац. Они выделили такие графы в отдельную группу "Тесные графы". формально их можно записать так:

$$\Big(G(U,V) - \text{Тесный граф}\Big) \Leftrightarrow \begin{cases} d(G) \approx 0 \\ cc(G) \approx 1 \end{cases}$$

Давайте докажем, что в случайный граф не подходит под это определение. Докажема, что коэффицент кластеризации стремится к 0 с ростом n:
Для начала посчитаем $\mathbb{E}[cc(v)]$. Обозначим $N = \binom{deg(v)}{t},\ t \leq deg(v)$. Тогда:

$$P(N * cc(v) = t) = \binom{N}{t} p^t (1-p)^{N-t} \Rightarrow N * cc(v) \sim Bin(p)$$

Тогда $\mathbb{E}[N * cc(v)] = Np \Rightarrow \mathbb{E}[cc(v)] = p : \forall N \in \mathbb{N}$ Заметим, что $cc(v)$ хоть и распределены одинаково, однако не независимы. Я утверждаю, что мы этим можем пренебречь, так как $k \ll n$. Данная гипотеза будет обоснована позднее.
Если считать, что $cc(v)$ - i.i.d, то можно использовать УЗБЧ:

$$cc(G) = \overline{cc(v)} \xrightarrow{a.s.} \mathbb{E}[cc(v)] = p = \frac{\lambda}{n} \to 0,\ n \to \infty$$

Теперь сформулируем гипотезу о независимости:

$$H_0 : cc(u_i) - iid,\ \forall u \in V : k \ll n$$

$$H_1 : alternative$$

Хочется ещё подметить, что тесные графы встречаются очень часто в природе. Нейронные сети в нашем мозге, карты дорог, пищевые цепочки. Их формирование кажется впоне логичным. Мозгу необходимо, чтобы сигнал быстро перемещался между нейронами, для быстрой обработки информации и реакции. Можно утверждать, что другие типы графов в данных примерах просто не прошли естественный отбор.

2.3 Подход Пономаренко

В 2012 году математики А.А.Пономаренко, Ю.А.Мальков, А.А.Логвинов и В.В.Крылов опубликовали статью, в которой предложили свой подход к созданию графов со свойствами тесного мира. А также предоставили оптимизацию для алгоритма жадного поиска. Так как сама модель очень сильно опирается на этот алгоритм, предлагаю начать изучение именно с него.

2.3.1 Жадный поиск

Данный алгоритм предназначен для эффективного поиска вершины в графе. Идейно он очень прост. На каждой итерации цикла, мы смотрим все соседние вершины и идём к той, что ближе всего к цели. Если таких вершин нет, то мы нашли то, что искали.

- Преимуществом данного алгоритма является его быстродействие по сравнению с другими методами.
- Недостаток же кроется в его неточности. Может существовать такая вершина отличная от нашей, соседи которой будут дальше от цели, чем она сама. Такие вершины, как уже упоминалось, принято называть локальными минимумами

Блок схему алгоритма можно увидеть на рис 2.

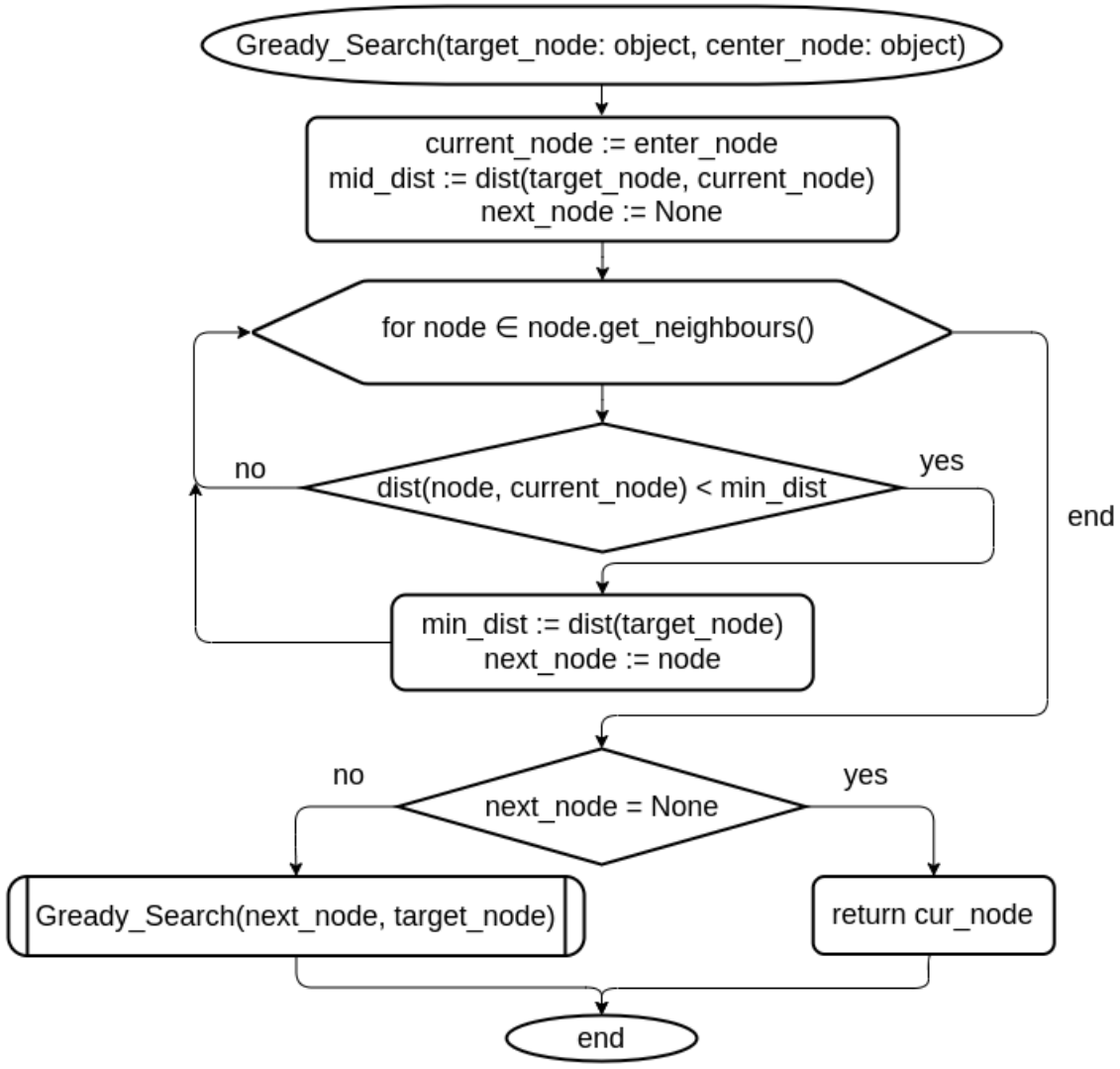

Рис. 2: Блок схема жадного поиска

Хочется подметить, что этот алгоритм относится к классу рандомизированных алгоритмов, так как сам поиск мы начинаем в случайной вершине. На практике такой подход показывает результат куда лучше, чем если бы стартовая вершина была детерминированной.

Математики из нижнего новгорода, упомянутые выше, предложили способ, как можно улучшить данный алгоритм, уменьшив вероятность попадания в локальные минимумы. Его блок схему вы можете найти на рис 3. Идея заключается в том, чтобы запустить жадный поиск несколько раз от произвольных вершин и выбрать ту, что окажется ближе. Данный подход рационален, так как мы подразумеваем, что работаем в "Тесном"графе, а значит, поиск не будет занимать много времени.

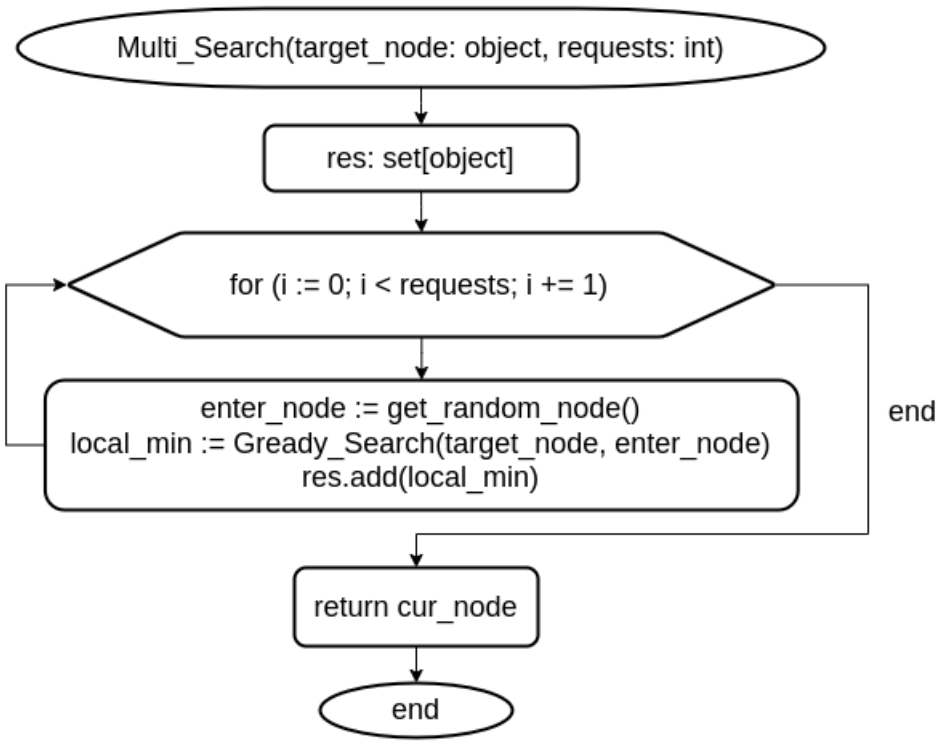

Рис. 3: Блок схема многократного жадного поиска

2.3.2 Построение структуры

Для построение структуры нам необходимо взять пустой граф и добавлять в него вершины поэлементно, опираясь на следующий алгоритм: Перед добавлением новой вершины v в граф, используя жадный поиск, мы ищем её потенциальное окружение (самые близкие к ней вершины, присутствующие в графе на данный момент). Выбрав из них только k штук (здесь k - параметр, который можно варьировать) связываем их с целевой вершиной v. Блок схему данного Алгоритма можно увидеть на рис 4.

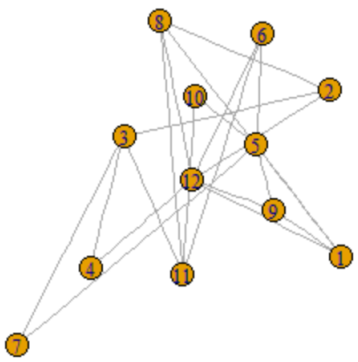

Рис. 4: Блок схема многократного жадного поиска

Хочется подметить несколько важных аспектов:

- Граф сильно кластеризован по определению, так как мы связываем вершины только с их ближайшими соседями
- Во время создания структуры, данные должны приходить случайно! Игнорирование этого требования может привести к увеличению среднего диаметра

Поподробнее остановимся на втором пункте.
Рассмотрим граф, изображённый на рис 5:

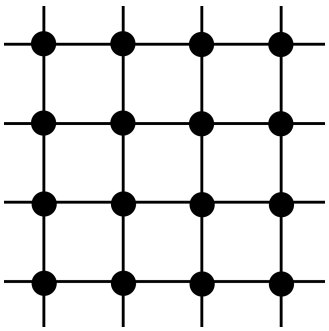

Рис. 5: Пример

Заметим, что даже такой крохотный пример уже имеет средний диаметр = 2.375. Если мы увеличим сетку в 16 раз, то получим средний диаметр = 9.5. То есть мы пришли к тому, что этот граф на 256 вершинах будет иметь средний диаметр больше, чем реальный граф с 8 млрд вершинами. Но это ещё не всё, заметим, что коэфицент класстеризации тут вовсе равен нулю. Так что, этот пример не просто слабо похож на тесный граф, а является его полной противоположностью.

Для понимания, как случайность спасает нас в данной ситуации, разделим построение графа на 2 части:

- В самом начале формирование графа, вершин в нём будет немного. Случайный поток данных приведёт к тому, что узлы будут достаточно далеко друг от друга (вероятность обратного пренебрежимо мала). Поэтому Ближайшие соседи вершины v, которые будут найдены посредством жадного алгоритма, будут хоть и самыми близкими из текущих, однако всё ещё будут находиться на достаточно большом расстоянии к v.
- С течением времени, ситуация начнёт меняться, ведь вершин будет становиться всё больше, а значит, и расположены они будут куда плотнее. Вершины добавленные

под конец формирования графа, будут иметь соседей, которые будут к ним очень близки не только в сравнении с другими, но и по абсолютной величине.

Другими словами, в в самом начале формирования графа образуются длинные связи, которые помогают быстро перемещаться по графу, а ближе к концу - короткие, которые спасают от локальных минимумов.

2.4 Подход Клайнберга

Следующий подход предложил Американский математик Джон Клайнберг. Перед началом разбора, мне бы хотелось сделать некоторую поправку. Джон Клайнберг в своих статьях нигде не упоминает о коэффициенте класстеризации. Вместо этого он говорит только о наличии длинных и коротких рёбер. Это немного расширяет класс "Тесных"графов, однако идейно ничего не меняет.

Изучим заново граф, который изображён на Рис. 5. Он состоит только из коротких связей, а значит, тесным графом не является. Есть разные способы решить эту проблему, давайте рассмотрим предложение Джона Клайнберга, так как его метод формирования длинных рёбер очень естественно вписывается в окружающий нас мир. Идея проста: чем человек дальше от меня, тем меньше вероятность нашего знакомства. Формально это можно записать так:

$$P(u \to v) = \frac{1}{d(u,v)^r}\frac{1}{C}, \text{ где } C = \sum_{u \neq z \in V} \frac{1}{d(u,z)^r} \tag{4}$$

Во время своего исследования, он пришёл к выводу, что наилучший результат достигается при r = d, где d - размерность пространства. Причём кол-во длинных рёбер должно быть примерно равно $q = \log n$. Так как: TODO:

Воспользуемся данной техникой для построения графа в общем случае: Для построения структуры по методу Клайнберго необходимо для каждой вершины v пройтись по всему графу, выбрать ближайших соседей, связать их с v и посчитать константу $C = \sum_{v \neq z \in V} \frac{1}{d(v,z)^r}$ на будущее. После завершения, останется только выбрать $q = \log n$ длинных связей, опираясь на распределение выше (4), и связать их с v.

Однако хочу заметить, что данный алгоритм можно немного упростить, если заметить, что в построение коротких рёбер здесь нет необходимости. Обосновать это очень просто, близкие связи будут образовываться сами собой, так как вероятность появления коротких рёбер будет значительно выше, чем длинных. Блок схема алгоритма изображена на рис. 6,

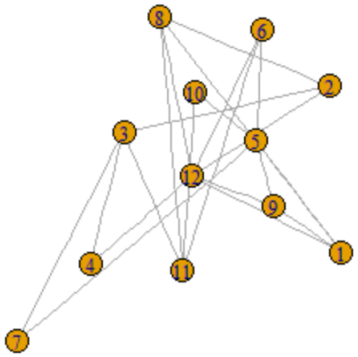

Рис. 6: Блок схема построение графа по методу Джона Клайнберга

2.5 Предложени по модификации

Оба подхода хорошо себя показывают в проблеме поиска ближайшего соседа. Однако Хочется предложить модификацию того алгоритма, который был предложен Нижегородскими математиками.

Есть гипотеза, что кол-во длинных рёбер, которые образовываются посредством данного алгоритма недостаточно для эффективной навигации по графу. Кроме того, нужно следить за тем, чтобы данные шли случайно, что тоже иногда может быть не очень удобно.

Для решения этой проблемы, предлагаю объединить данный подход с классическим методом построения тесного графа Уоттса–Строгаца. Идея заключается в следующем:

1) Жадным поиском ищем окружение вершины v, которую хотим добавить

2) Выбираем k(параметр) ближайших вершин.

3) Связываем вершину v лишь с некоторыми из выбранных вершин. Вероятность ребра теперь будет являться отдельным параметром и задаваться заранее

4) Каждое непроведённое ребро будет компенсироваться другим. Это новое ребро будет связывать нашу вершину v с другой абсолютно случайно выбранной вершиной в графе.

В практической части, я собираюсь проверить эффективность данного подхода в сравнении с другими методами. А также подобрать параметр p, при котром модель будет давать наилучшие результаты.

3 Практическая часть

3.1 Описание абстракций

Для начала, я бы хотел описать, какие классы были реализованы, каким образом и для чего они преднозначены.

3.1.1 Класс Graph

Данный граф является основным. Он реализует все самые необходимые инструменты для взаиможействия с графами, такие как: добавление вершины в граф, добавление ребра, очистка графа и т.д.

Данная абстракция была реализована на основе хэш таблицы (std::unordered_map). Такой контейнер был выбран не случайно, так как он осуществляет очень быстрый доступ к произвольным данным (Добавление, поиск, удаление за O(1)).

Этот класс является шаблонным и может хранить в себе объекты любой природы. Главное, чтобы они удовлетворяли двум требованиям:

- Наличие перегрузки оператора равенства
- Наличие функции хэширования (для std::unordered_map)

Объявление класса:

```
template<typename TObject, typename HashFunc>
class Graph {
    std::unordered_map<TObject, std::vector<TObject>, HashFunc> graph;
    std::vector<TObject> nodes;
    size_t size;
public:
    Graph() = default;
    Graph(const Graph& g) = delete;
    Graph(Graph&& g) = delete;

    void add_node(const TObject& p);
    void add_edge(const TObject& p1, const TObject& p2);
    void delete_edge(const TObject& p1, const TObject& p2);
```

```
        void clear();
        std::size_t get_size() const;
        bool consists_node(const TObject& p) const;
        bool consists_edge(const TObject& p1, const TObject& p2) const;
        const std::vector<TObject>& get_neighbours(const TObject& p) const;
        const TObject& get_random_node() const;
        friend std::ostream& operator<< <>(std::ostream& out, const Graph& graph);
        ~Graph() = default;
    };
```

Хочу отдельное внимание обратить на метод get_random_node. Его необходимость станет очевидна вдальнейшем, пока мне бы хотелось рассказать о дилеме, с которой я столкнулся во время её написания. Для реализации этого метода, мы должны иметь возможность выбирать из хэш таблицы любой ключ случайно, однако это возможно сделать только за линейное время, из-за отсутствия в unordered_map обращения по индексу. Для решения этой проблемы мною было принято решение создать отдельный вектор из объектов, так как в нём, используя случайные индексы, мы без проблем сможем выбрать абсолютно любой объект за O(1). Это привело к следующим последствиям:

- Увеличение кол-во памяти необходимое для хранения структуры
- Ухудшение асимптотической скорости удаление выбранных элементов (до линейной)

Однако все эти недостатки никак негативно не повлияли конкретно в решении задач текущего проекта

3.1.2 Классы Point и Point3D

Данные абстракции в моём проекте используются, как объекты, вершины графа. Они являются больше примером для наглядной демонстрации работы других классов.

4 Заключение