**Министерство образования и науки Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная Школа Информационных Технологий и Робототехники

Направление подготовки 09.03.02 Информационные системы и технологии

Отделение Информационных Технологий

**УЧЕБНО-ИСЛЕДОВАТЕЛЬСКАЯ РАБОТА СТУДЕНТА**

|  |
| --- |
| **Тема работы** |
| Поиск кратчайшего пути на графе дорог |

Студенты

|  |  |  |  |
| --- | --- | --- | --- |
| **Группа** | **ФИО** | **Подпись** | **Дата** |
| 8И6Б | П. В. Моисеев |  |  |
| 8И6Б | И. А. Кудрявцева |  |  |

Руководитель

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Должность** | **ФИО** | **Ученая степень, звание** | **Оценка** | **Подпись** | **Дата** |
| Доцент ОИТ | Р. В. Ковин | к.т.н. |  |  |  |

Томск – 2019 г.

**Оглавление**

[Введение 3](#_Toc535013679)

[Цель и задачи 4](#_Toc535013680)

[1. Задача поиска кратчайшего пути 5](#_Toc535013681)

[1.1. Классические алгоритмы поиска кратчайшего пути 5](#_Toc535013682)

[1.1.1. Алгоритм Дейкстры 5](#_Toc535013683)

[1.1.2. A\* – модификация алгоритма Дейкстры 5](#_Toc535013684)

[1.1.3. Двунаправленный поиск 7](#_Toc535013685)

[1.2. Иерархические подходы 9](#_Toc535013686)

[1.2.1. Использование иерархии дорог 9](#_Toc535013687)

[1.2.2. Построение иерархии графа 10](#_Toc535013688)

[1.2.3. Сепараторы 11](#_Toc535013689)

[1.2.4. Grid Based Transit Nodes 11](#_Toc535013690)

[1.2.5. Таблицы расстояний 11](#_Toc535013691)

[1.2.6. Reach 12](#_Toc535013692)

[1.3. Целеустремленные алгоритмы 13](#_Toc535013693)

[1.3.1. Arc-Flags 13](#_Toc535013694)

[1.3.2. ALT 13](#_Toc535013695)

[2. Реализация алгоритмов поиска кратчайшего пути 15](#_Toc535013696)

[2.1. Визуализация графа 15](#_Toc535013697)

[2.2. Реализация алгоритма Дейкстры 15](#_Toc535013698)

[2.3. Реализация алгоритма А\* 16](#_Toc535013699)

[2.4. Сравнение работы алгоритмов 17](#_Toc535013700)

[Заключение 20](#_Toc535013701)

[Список литературы 21](#_Toc535013702)

# Введение

Задача поиска кратчайшего пути в последнее время получила широкое распространение, благодаря своему применению для решения множества других задач.

Она широко используется при определении наименьшего расстояния в сети дорог.

Сеть дорог можно представить в виде графа с положительными весами. Вершины являются дорожными развязками, а ребра дорогами, которые их соединяют. Веса рёбер могут соответствовать протяжённости данного участка, времени необходимому для его преодоления или стоимости путешествия по нему. Ориентированные ребра можно использовать для представления односторонних улиц. В таком графе можно ввести характеристику, которая указывает на то, что одни дороги важнее других для длительных путешествий (например, автомагистрали).

Задача о кратчайшем пути является одной из важнейших классических задач теории графов. На сегодняшний день известно множество алгоритмов для ее решения.

В данной работе будет представлен обзор, сравнение и реализация основных алгоритмов решения данной задачи.

# Цель и задачи

Основной целью проекта является изучение алгоритмов поиска кратчайшего пути на графе дорог.

Практическим результатом проекта является программная реализация алгоритмов Дейкстры и А\* на языке программирования C#.

Для достижения цели работы поставим перед собой следующие задачи:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задача | Сроки | Ответственные |
| 1 | Поиск информации по теме проекта | 6.10.18-7.10.18 | Кудрявцева |
| 2 | Изучение известных алгоритмов | 8.10.18-1.11.18 | Кудрявцева, Моисеев |
| 3 | Реализация визуализации графа | 2.11.18-8.11.18 | Кудрявцева |
| 4 | Разработка алгоритма Дейкстры | 8.11.18-1.12.18 | Моисеев |
| 5 | Разработка алгоритма А\* | 2.12.18-25.12.18 | Кудрявцева |
| 6 | Сравнение работы разработанных алгоритмов | 25.12.18-28.12.18 | Моисеев |
| 7 | Выполнение отчета по работе | 9.01.19-11.01.19 | Кудрявцева, Моисеев |

# Задача поиска кратчайшего пути

Задача о кратчайшем пути — задача поиска самого короткого пути между двумя вершинами на графе, в которой минимизируется сумма весов рёбер, составляющих путь.

В данном разделе рассматриваются некоторые подходы и алгоритмы для решения задачи поиска кратчайшего пути на графе.

# Классические алгоритмы поиска кратчайшего пути

## Алгоритм Дейкстры

Алгоритм Дейкстры (также называемый поиском с равномерной стоимостью) позволяет нам задавать приоритеты исследования путей. Вместо равномерного исследования всех возможных путей он отдаёт предпочтение путям с низкой стоимостью. Мы можем задать уменьшенные затраты, чтобы алгоритм двигался по дорогам, повышенную стоимость, чтобы избегать лесов и многое другое.

Проблема в том, что алгоритм Дейкстры не использует никакой информации о свойствах графа и искомого маршрута и при его работе (распространении т.н. «волны») периметр этой «волны» движется от искомой точки во всех направлениях без какой-либо дискриминации.

Скорость работы алгоритма Дейкстры сильно зависит от скорости операций с приоритетной очередью.

## A\* – модификация алгоритма Дейкстры

A\* — это модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам, A\* находит путь к одной точке. Он отдаёт приоритет путям, которые ведут ближе к цели.

В A\* стоимость вершины при помещении её в приоритетную очередь не просто равна пройденному пути, а включает еще и оценку пути оставшегося.

При этом, если ребро ведет нас по направлению к финишу, оценка для него будет уменьшаться и компенсировать пройденный путь.

Для ребер, ведущих от цели, эта оценка станет расти, в результате такие точки попадут в очередь с большим весом и вполне вероятно, что до них дело так и не дойдёт.

A\* относится к так называемым информированным алгоритмам т.к. в эвристике мы используем предположение, что движение в направлении цели скорее приведёт нас к успеху.

Есть одна важная проблема с оценкой стоимости. В жизни геометрически близкие объекты могут быть достаточно удаленными с точки зрения дорожного графа. Например, находиться на разных сторонах реки, горной гряды, моря. В таком случае оценка, основанная на геометрической близости, начнёт усугублять ситуацию, упорно направляя «волну» в неправильном направлении, что хорошо видно на рисунке 1, взятом из источника [1].

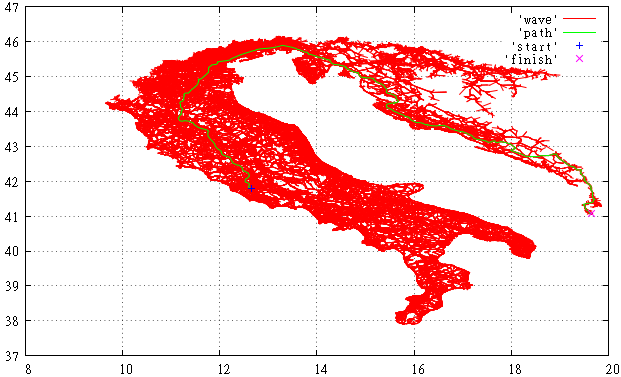


Рисунок 1 – просмотренная алгоритмом A\* часть графа OSM[[1]](#footnote-1) для поиска пути из Италии в Албанию

Впрочем, это всё равно лучше, чем алгоритм Дейкстры (рисунок 2) [1]. Хорошо видно, как заполнив всю Италию, «волна» начала переливаться через край, набрала скорость и быстро достигла цели.

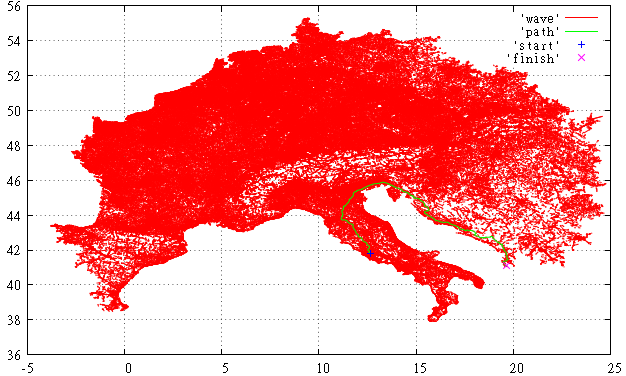


Рисунок 2 – просмотренная алгоритмом Дейкстры часть графа OSM для поиска пути из Италии в Албанию

## Двунаправленный поиск

Можно пустить две A\* волны навстречу друг другу. Это называется двунаправленным А\* (рисунок 4) и на первый взгляд кажется очень привлекательной идеей. В самом деле, при хорошей транспортной связности “волна” представляет собой эллипсоид, две малых волны, пущенные навстречу, заметут меньшую площадь по сравнению с одной большой. С другой стороны, возникает проблема обнаружения встречи «волн», точек в их периметрах может быть довольно много и проверять на каждом шагу наличие ребра в чужом периметре не так уж и дешево.

На практике двунаправленный поиск алгоритмом Дейкстры (рисунок 3) быстрее обычного алгоритма Дейкстры примерно в два раза [2].

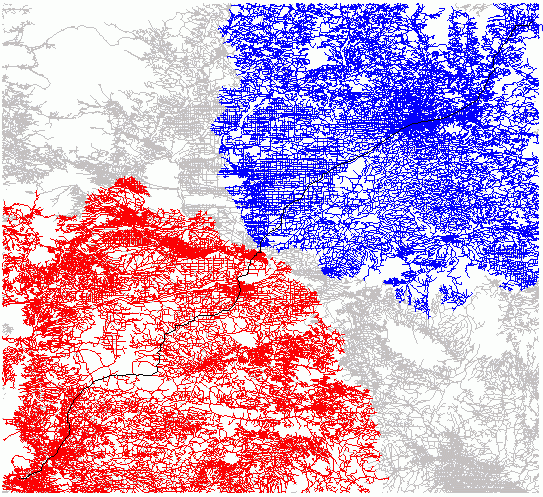


Рисунок 3 – Двунаправленный алгоритм Дейкстры

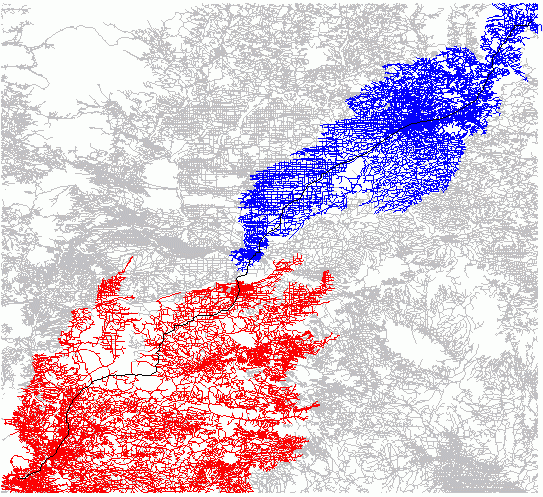


Рисунок 4 – Двунаправленный алгоритм А\*

Двунаправленный поиск имеет свои особенности. Он хорошо работает с алгоритмом Дейкстры, но с А\* возникает трудность такого рода: он усиливает его действие. «Там, где А\* искал хорошо, он будет искать просто отлично, там, где А\* искал плохо, он начинает искать совершенно отвратительно» [3].

# Иерархические подходы

## Использование иерархии дорог

Некоторые коммерческие системы, например, StreetMap USA (рисунок 5), используют для роутинга тот факт, что хорошо спланированная дорожная сеть имеет двухуровневую природу — есть сеть местных дорог и (значительно меньшая) сеть шоссе для поездок на дальние расстояния [1]. Представляется естественным использовать этот факт. Вводятся шлюзы (transit nodes) — вершины, на которых происходит переход из одного уровня в другой. Нахождение “достаточно длинного” пути сводится к нахождению путей от:

* исходной точки до нескольких ближайших шлюзов;
* нескольких ближайших шлюзов до финальной точки;
* кратчайшего маршрута от любого из начальных шлюзов до любого из финальных, это, конечно, делается за один сеанс.

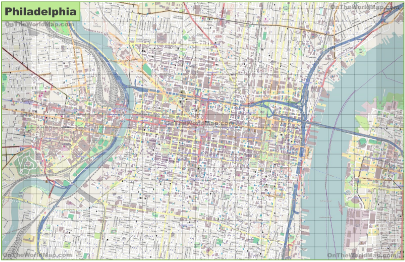


Рисунок 5 – Карта StreetMap

Минусы такого подхода:

* Не везде дорожная сеть хорошо спланирована, кое-где она выросла стихийно, следовательно, исходный посыл не работает.
* Сеть верхнего уровня должна быть связной и выверенной. Из OSM, например, невозможно получить такую сеть (лёгким движением руки) просто отфильтровав дороги по классам.
* Институт шлюзов также требует много ручной работы.

## Построение иерархии графа

Если нет возможности выверять граф, остаётся возможность построить иерархию автоматически.

Так или иначе, используется идея, что хоть граф и не выверен, тем не менее, атрибуты рёбер позволяют строить маршруты приемлемого качества. Но в силу размеров графа, построение тем же A\* в рабочем режиме непозволительно дорого.

Например, это может выглядеть так:

* на этапе предрасчета выбирается множество (пусть даже случайных) вершин;
* для пар пространственно удаленных вершин обычным A\* строятся кратчайшие маршруты;
* на основе построенных маршрутов ведётся статистика пройденных рёбер;
* после того, как набралось достаточное количество данных, посещенные рёбра объявляются следующим уровнем иерархии;
* последовательно идущие рёбра без ветвлений сливаются в «макрорёбра» с сохранением стоимости проезда;

Построенный граф может быть снова подвержен описанной процедуре, таким образом строится необходимое число иерархий.

Маршрутизация в таком иерархическом графе осуществляется двунаправленным поиском (A\* или алгоритмом Дейкстры).

## Сепараторы

Основной идеей метода является попытка разделения графа на компоненты путём удаления небольшой части ребер — сепараторов. Эти сепараторы и предварительно вычисленные пути между ними образуют следующую иерархию. Утверждается [4], что, затратив времени и пространства на диске для предварительного расчета, можно выполнять запросы за .

## Grid Based Transit Nodes

Это в общем та же идея с сепараторами, но в целях масштабирования и простоты граф делится на фрагменты решеткой или квадродеревом, ребра, которые пересекают границы фрагментов становятся транзитными. Понятно, что цена этому — эффективность. В плюсах — высокая автоматизация и как следствие отсутствие человеческого фактора.

## Таблицы расстояний

На вышестоящих уровнях иерархии в процессе поиска пути не ищутся, а стоимость подсчитывается на основе предвычисленных таблиц расстояний между транзитными нодами [4]. Когда маршрут определен, пути восстанавливаются локальным поиском (рисунок 6).

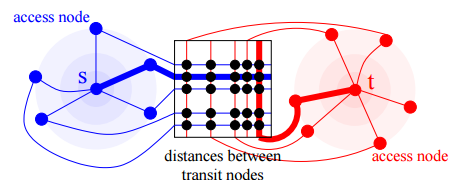


Рисунок 6 – Восстановление пути по таблице расстояний

## Reach

Замечено, что при построении длинных оптимальных маршрутов, «локальные» рёбра посещаются только в самом начале или конце маршрута. Следовательно, построив некоторое количество «обучающих» длинных маршрутов, можно понять, насколько близко расположено то или иное ребро к любому из концов маршрута.

Для некоторого обучающего маршрута P(s…u.v...t) вводится показатель reach — минимальное расстояние до концов reach(uv) = min(dist(s…..u), dist(v…..t)), где dist() – расстояние между двумя вершинами. На всём обучающем множестве reach(uv) — максимальное значение на всех маршрутах, где встретилось ребро (uv) [5].

При «боевом» поиске мы вдали от старта и финиша просто будем стараться избегать рёбер с маленьким значением reach (рисунок 7).

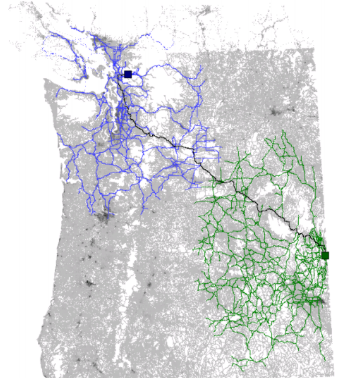


Рисунок 7 – Алгоритм Reach

Идея метода очень красивая, вопросы вызывает лишь качество обучающей выборки, её достаточность и ресурсы, потраченные на обучение [6].

# Целеустремленные алгоритмы

## Arc-Flags

Граф делится на фрагменты. Проводится обучение построением кратчайших маршрутов между предопределенными точками. При построении обучающего пути, для каждого ребра сохраняется факт, что через него проходит кратчайший маршрут в клетку финальной точки [7].

Таким образом для каждого ребра мы храним маску флагов, в какие фрагменты графа можно через это ребро добраться кратчайшим путём.

Специфические недостатки этого метода видны невооруженным взглядом:

* количество фрагментов графа не может быть велико, 8000 фрагментов (что не так уж и много) дадут нам (предположительно несжимаемый) килобайт на каждое ребро;
* надо быть очень осторожным с нарезкой фрагментов, внутри фрагмента граф должен быть связным.

## ALT

Из всех вершин выбирается небольшое количество landmarks: λ. В изначальном варианте для каждой вершины предварительно рассчитывались стоимости до каждого λ. Это требовало колоссального количества дополнительной памяти и в дальнейшем требования смягчились и вершины стали группировать.

Поиск в ALT (рисунок 8) осуществляется как в A\*, но оценка оставшегося пути делается на основе рассчитанных стоимостей [1]. Пусть мы рассматриваем ребро (u,v) на пути к целевой вершине t. Для каждой λ в соответствии с неравенством треугольника мы имеем оценку оставшейся части пути (через λ): dist(λ, t) − dist(λ, v) ≤ dist(v, t) и dist(v, λ) − dist(t, λ) ≤ dist(v, t), где dist() - расстояние между двумя вершинами. Минимум для всех λ и даст искомую оценку [8].

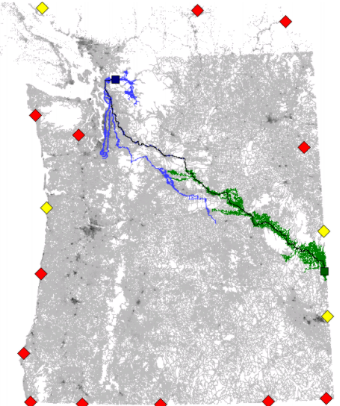


Рисунок 8 – Алгоритм ALT

# Реализация алгоритмов поиска кратчайшего пути

Для реализации были выбраны алгоритм Дейкстры и алгоритм А\*, так как они являются классическими алгоритмами, на которых основано большинство современных алгоритмов поиска кратчайшего пути.

# Визуализация графа

Чтобы увидеть результаты работы алгоритмов и сравнить их было решено рисовать граф в разработанном нами во время прохождения дисциплины «Геоинформатика» приложении MiniGIS. Был создан класс Graph, описывающий структуру графа: список вершин (поле List<Point> points), список ребер (List<Edge> edges). Вершины представлены существующим классом Point, описывающем точечные объекты. Класс ребер Edge наследуется от класса Line, описывающего линейные объекты и имеет поле Weight, хранящее вес ребра.

# Реализация алгоритма Дейкстры

Алгоритм Дейкстры:

1. Заносим в приоритетную очередь стартовую точку с нулевой стоимостью.

2. Пока в очереди что-то есть и цель не достигнута — пусть вершина V, для каждого исходящего ребра (E), которое мы до сих пор не просматривали:

а) проверяем что это не искомое ребро, если да, то конец;

б) подсчитываем стоимость прохода E;

в) и заносим конечную вершину ребра E в очередь со стоимостью её достижения — стоимость достижения V + стоимость E.

Для проверки работоспособности алгоритма Дейкстры, будем осуществлять поиск кратчайшего пути на графе, состоящем из ста вершин, образующих сеть 10 на 10 и связанных с 4 соседними вершинами (рисунок 9).

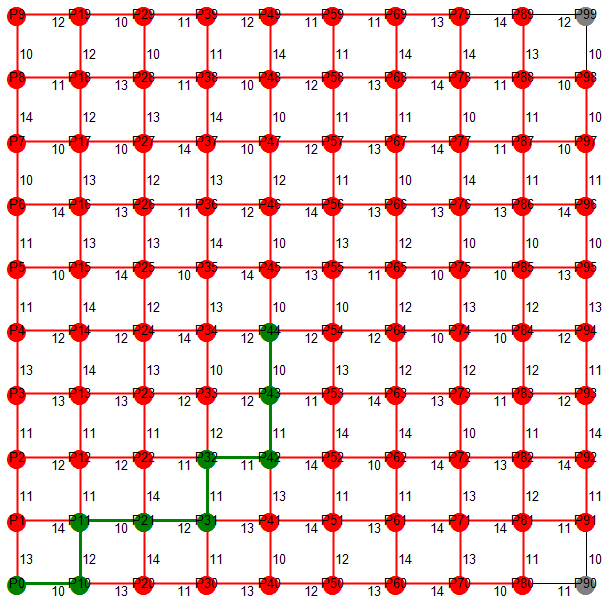


Рисунок 9 – Работа алгоритма Дейкстры

Зеленым цветом отмечен минимальный путь из вершины P44 в вершину P00, красным – просмотренные ребра и вершины.

# Реализация алгоритма А\*

Алгоритм А\* получен добавлением к алгоритму Дейкстры эвристической функции поиска геометрического расстояния до цели. Поиск пути будем осуществлять на том же графе (рисунок 10).

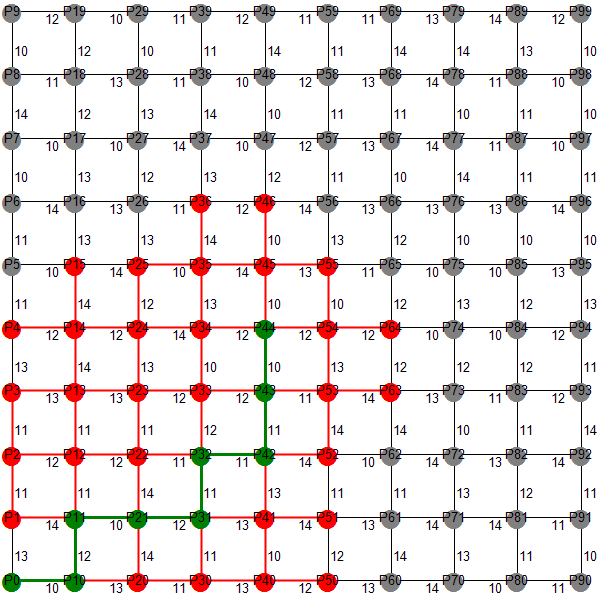


Рисунок 10 – Работа алгоритма А\*

# Сравнение работы алгоритмов

Путь, найденный с помощью алгоритма А\* (рисунок 10) совпадает с путем, найденным алгоритмом Дейкстры (рисунок 9). Но при этом у А\* есть видимое преимущество. Алгоритм А\* обходит меньше вершин и ребер, чем алгоритм Дейкстры, а, следовательно, работает быстрее. На данном графе алгоритм Дейкстры, чтобы найти кратчайший путь обошел 98 вершин и 175 ребер за 9 мс, а алгоритм А\* – 39 вершины и 57 ребер за 4 мс.

Тест проводился на компьютере с характеристиками: Операционная система Windows 10 64-bit, Процессор Intel Core i3-4000M 2,40 Ггц, Оперативная память 8 Гб.

При добавлении препятствия (рисунок 11) работа алгоритма Дейкстры остается прежней (пройдено 98 вершин и 170 ребер за 9 мс), а алгоритм А\* стал искать путь хуже (пройдено 72 вершины и 112 ребер за 5 мс).

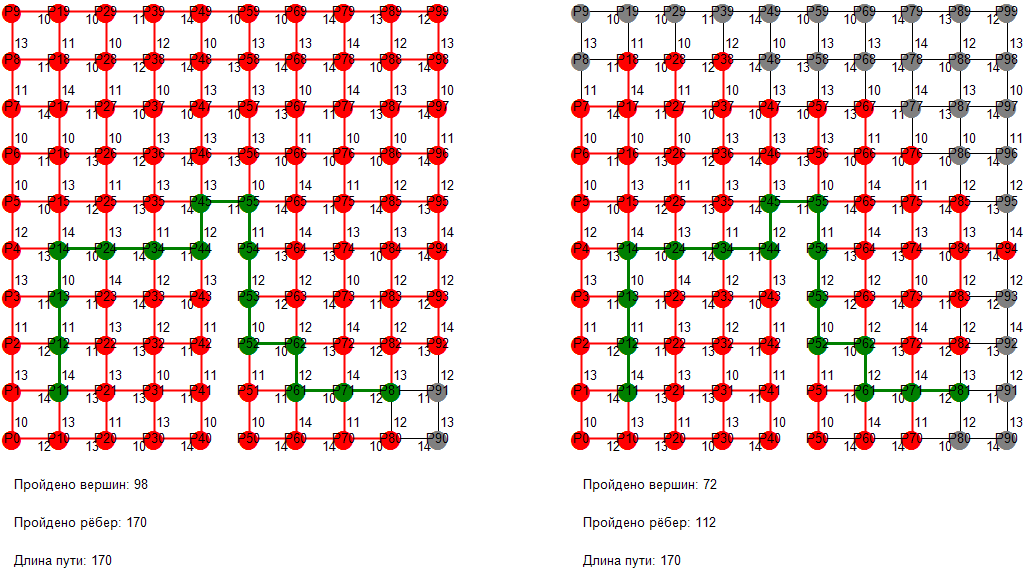


Рисунок 11 – Граф с препятствием

Также было проведено тестирование для графов разных размеров (таблица 1).

Таблица 1 – Результаты тестирования алгоритмов для различных графов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Граф | Количество пройденных вершин | Количество пройденных ребер | Время работы алгоритма, мс |
| Дейсктра | G(100,180) | 98 | 175 | 9 |
| А\* | G(100,180) | 39 | 57 | 4 |
| Дейсктра | G(100,175)  с препятствием | 98 | 170 | 9 |
| А\* | G(100,175)  с препятствием | 72 | 112 | 5 |
| Дейсктра | G(400,760) | 400 | 760 | 16 |
| А\* | G(400,760) | 192 | 339 | 8 |
| Дейсктра | G(900,1740) | 900 | 1740 | 69 |
| А\* | G(900,1740) | 384 | 705 | 26 |

По результатам тестирования построены графики зависимости сложности алгоритмов от количества вершин (рисунки 12, 13 и 14).

Рисунок 12 – Зависимость количества пройденных вершин от количества вершин в графе

Рисунок 13 – Зависимость количества пройденных ребер от количества вершин в графе

Рисунок 14 – Зависимость времени выполнения от количества вершин в графе

# Заключение

Что касается теоретической части работы, мы видим два основных направления, в которых идёт развитие:

* Иерархии. Позволяют весьма эффективно строить пути на больших расстояниях в структурированных графах. Но на маленьких расстояниях дешевле пользоваться обычным A\* или Дейкстры. Следовательно, существует “серая” область, где оба алгоритма работают посредственно;
* Использование внутренней природы графа для принятия решения о направлении движения к цели. Идея выглядит многообещающей, но вызывает много вопросов. Основная проблема — человеческий фактор. Что lanmark’и, что фрагменты Arc-Flags требуют участия эксперта, если пустить их определение на самотёк, легко можно получить не то, что нужно. Кроме того, требуется (нелинейное от размера графа) количество дополнительной памяти.

В практической части работы было реализовано два основополагающих алгоритма. Их сравнение показало, что алгоритм А\* эффективней (по времени выполнения и количеству пройденных ребер и вершин) алгоритма Дейкстры более чем в 2 раза. Сложность алгоритма Дейкстры меньше зависит от структуры графа, чем сложность алгоритма А\*, что следует из тестирования на графах с препятствиями и без. Также, проведя тестирования на графах различных размеров, можно прийти к выводу, что алгоритм Дейкстры имеет квадратичную сложность, а сложность алгоритма А\* приближена к линейной, что делает его использование более выгодным.

# Список литературы

1. Муратшин Борис. M\* — алгоритм поиска кратчайшего пути, через весь мир, на смартфоне. Блог компании 2GIS. [Электронный ресурс] URL: https://habr.com/company/2gis/blog/326638/ (дата обращения: 08.10.18);
2. Кононенко В. А. Поиск кратчайших путей на графе дорог: презентация проекта / СПбАУ, 2011 г. http://mit.spbau.ru/files/kononenko.pdf;
3. Муратшин Борис. Знакомьтесь, М\*. [Электронный ресурс] URL: https://dump-conf.ru/person/204/ (дата обращения: 08.10.18);
4. Sanders Peter, Schultes Dominik. Robust, Almost Constant Time Shortest-Path Queries in Road Networks . – Karlsruhe, Universität Karlsruhe: http://www.dis.uniroma1.it/challenge9/papers/sanders.pdf;
5. Gutman R. J. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, 2004, January. https://dblp.org/rec/bib/conf/alenex/Gutman04;
6. Goldberg A. V. Reach for A∗: an Efficient Point-to-Point Shortest Path Algorithm. http://www.cs.princeton.edu/courses/archive/spr09/cos423/Lectures/ reach-mit.pdf;
7. E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In Proceedings of the 4th Workshop on Experimental Algorithms (WEA’05), Lecture Notes in Computer Science, pages 126–138. Springer, 2005;
8. Goldberg, A.V., Werneck, R.F.: Computing Point-to-Point Shortest Paths from External Memory. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX 2005), pp. 26–40. SIAM (2005).

1. OpenStreetMap (OSM) — некоммерческий веб-картографический проект по созданию силами сообщества участников — пользователей Интернета подробной свободной и бесплатной географической карты мира. [↑](#footnote-ref-1)