ПОИСК КРАТЧАЙШИХ ПУТЕЙ В ТРАНСПОРТНЫХ СЕТЯХ

CALCULATION OF THE SHORTEST PATHSIN TRANSPORT NETWORKS

УДК 519.173.5

Копылов Максим Дмитриевич, студент кафедры Вычислительной техники, Санкт-Петербургский государственный электротехнический университет, г. Санкт-Петербург.

Хохлов Кирилл Андреевич, студент кафедры Вычислительной техники, Санкт-Петербургский государственный электротехнический университет, г. Санкт-Петербург

Kopylov M.D. maksim.kopylov@gmail.com

Khokhlov K.A. kkhohlow@gmail.com

Аннотация

При работе с современными логистическими задачами часто возникает необходимость поиска кратчайшего пути. Данная статья анализирует различные алгоритмы поиска кратчайшего пути в транспортных сетях, а также их практическое применение. Был разработан инструмент Pathfinder, который позволяет искать кратчайший путь в графе с помощью различных алгоритмов. С помощью Pathfinder были исследованы алгоритмы на графах.

Annotation

When dealing with modern logistics tasks, it is often necessary to find the shortest path. This article analyzes algorithms for finding the shortest path in transport networks, as well as their practical application. The Pathfinder tool was developed, which allows finding the shortest path in a graph using various algorithms. Algorithms on graphs were investigated using Pathfinder.

Ключевые слова: Транспортные сети, логистика, алгоритмы на графах, алгоритм Дейкстры, оптимизация, таблицы путей, A*, ALT

Keywords: Transport networks, logistics, graph algorithms, Dijkstra's algorithm, optimization, path tables, A *, ALT

Введение

Статья рассматривает несколько способов решения задачи поиска кратчайшего пути (Shortest-Paths, SP) в контексте транспортных сетей.

Суть задачи SP состоит в том, чтобы найти кратчайший путь в графе между двумя его вершинам s и t (начальная и конечная точка соответственно).

В настоящее время, с развитием мобильных технологий, все острее стоит потребность в решении задачи SP с помощью мобильных устройств.

Частично решить проблему можно переносом вычислений на удаленные сервера, но в таком случае появляется другая проблема: в рамках огромного количества запросов на поиск кратчайшего пути необходимо обеспечить минимально возможное время ответа системы в условии относительно ограниченных вычислительных ресурсов.

Для решения такой задачи существует большое количество алгоритмов, которые решают эту задачу с разной степенью эффективности. Определяя способ решения необходимо соблюдать баланс между скоростью работы алгоритма и ресурсами, которые он потребляет в процессе работы.

Классическим решением задачи SP является алгоритм Дейкстры, которому в худшем случае, даже с различными оптимизациями, для поиска кратчайшего пути необходимо обойти весь граф.

Развитием алгоритма Дейкстры можно назвать алгоритм A*, который добавляет в изначальный алгоритм потенциальную функцию, с помощью которой оценивает снизу кратчайший путь. Оба алгоритма находят точное решение задачи SP за полиномиальное время.

В условии, когда временные ограничения поиска решения на графе большой размерности должны укладываться в миллисекунды, применяются различные оптимизации. Большинство из них основаны на двухэтапном решении задачи.

На первом этапе, называемом предобработкой, происходит извлечение из графа исходного графа информации, которая может помочь в ускорении работы классических алгоритмов. А на втором этапе происходит непосредственно сам запуск алгоритма, который будет искать решение на основе информации, полученной на этапе предобработки.

Существует достаточно много алгоритмов, в основе которых лежит двухэтапная структура: Ark-Flags, SHARK, ALT, и т.д.

Постановка задачи

Задача о кратчайшем пути (англ. Shortest-Paths, SP) - одна из основных задач маршрутизации. Данная задача относится к теории графов. Задача о кратчайшем пути применяется в следующих областях: моделирование трафика, анализ сетевых структур, множество различных логистических задач.

Целью задачи является поиск минимального расстояния между двумя заданными вершинами ориентированного графа U и V. Поиск такого пути производится минимизацией весов ребер, пролегающих между вершинами.

Существует множество способов решения задачи о кратчайшем пути. Наиболее часто используется алгоритм Дейкстры. На практике алгоритм Дейкстры не оптимален, поэтому существуют различные модификации алгоритма Дейкстры, целью которых служит сокращение времени, затрачиваемого на поиск.

Одной из подобных модификаций является алгоритм A* (A star). Данный алгоритм работает быстрее, т.к. в A* применяется потенциальная функция, определенная на множестве вершин графа. Существует также улучшение алгоритма A*, алгоритм ALT (A* with Landmark & Triangle). В алгоритме ALT потенциальная функция может быть задана набором ориентиров, подмножеством множества вершин графа G. В таком случае перемещениям ориентиров соответствуют различные потенциальные функции. Выбор

оптимального набора ориентиров осуществляется в конечном множестве вариантов и является NP-трудной задачей.

Современные высоконагруженные системы ставят высокие критерии эффективности и применения алгоритмов. Современный сервер, такой, например, как Яндекс.Карты принимает миллионы запросов ежедневно, поэтому эффективный, быстрый и незатратный по ресурсам поиск маршрутов крайне необходим [6]. В статье кратко описывается каждый из популярных алгоритмов и анализируется его практическое применение с помощью Pathfinder. Далее производится оценка алгоритма и перспектив его применения.

Обзор применяемых алгоритмов

Т.к. задача поиска кратчайшего пути является задачей теории графов, которую необходимо применить к транспортным сетям, которые в свою очередь являются совокупностью всех видов транспорта; улицы и дороги, линии внеуличного транспорта (трамвай, метро), а также маршруты общественного транспорта, но с другой стороны транспортную сеть можно представить в виде ориентированного взвешенного (каждое ребро графа имеет вес) графа G = <V, E>, где E - ребра графа (дороги), а V - узлами (перекрестки), вес ребра задается некоторой функцией Weight(e) на каждом ребре e ∈ E, которая сопоставляет ребру некоторое строго положительное число. Таким образом задача поиска пути в транспортной сети сводится к уже известной задаче поиска пути в графе.

Далее в статье будут рассмотрены некоторые алгоритмы, которые решают поставленную задачу, также будет приведен пример программы, которая реализует эти алгоритмы, на основе которой будет проведено их сравнение.

Алгоритм Дейкстры

Алгори́тм Де́йкстры (англ. Dijkstra’s algorithm) — алгоритм на графах, который был изобретен Эдсгером Дейкстрой в 1959 году. Данный алгоритм находит кратчайшие пути из любой данной вершины графа до всех других вершин этого графа.

Сложность алгоритма Дейкстры зависит от следующих факторов:

- Способ нахождения очередной вершины V графа
- Способ хранения множества непосещенных вершин в графе
- Способ обновления существующих меток, соответствующих ребрам графа

Пусть дан граф G, в котором N вершин соединены с помощью M ребер.

Очевидно, что при полном переборе всех вершин сложность алгоритма составляет O(n^2): основной цикл выполняется не более N раз, побочные вложенные циклы также выполняются не более N раз.

Введем улучшение: запустим алгоритм одновременно из начальной вершины в конечную и из конечной в начальную. Такой подход не испортит время работы в худшем случае. Рассмотрим реализацию данного улучшения:

Создадим две приоритетных очереди вершин графа. На одной очереди будем искать путь из S в T, а на другой наоборот - из T в S. Как только любая вершина Z окажется удалена из обеих очередей - кратчайший путь найден и алгоритм завершит работу. При практическом применении данное улучшение ускоряет скорость работы алгоритма Дейкстры примерно в два раза. [3]

Далее в статье будут рассмотрены другие улучшения алгоритма Дейкстры, использующие эвристические функции.

Алгоритм A*

Основной возможностью для улучшения алгоритма Дейкстры является оптимизация выбора следующей вершины, которую обработает алгоритм. Для алгоритма A* порядок обхода вершин определяется эвристической функцией “Расстояние + стоимость”. Обозначим такую функцию f(x).

Данная функция получается путем суммирования двух других функций: функции стоимости достижения конечной вершины из рассматриваемой (данная функция является эвристической) и функции достижения рассматриваемой вершины из начальной (данная функция не обязательно должна быть эвристической).

Т.О. f(x) = g(x) + h (x), где g(x) - путь из начальной в текущую, а h(x) - из текущей в конечную вершины.

На функцию h(x) накладывается ограничение: данная функция не должна не должна переоценивать расстояния к целевой вершине. В качестве примера h(x) можно использовать функцию, возвращающую расстояние между вершинами по прямой. Очевидно, что такая функция никогда не переоценит расстояние из-за правила треугольника.

В качестве функции h(x) статье была взята функция евклидова расстояния:

$$h(v) = \sqrt{(v.x - goal.x)^2 + (v.y - goal.y)^2}$$

Как и алгоритм Дейкстры, A* пошагово просматривает все пути от начальной вершины к конечной и завершается после нахождения минимального пути. A* является информированным алгоритмом, поэтому сначала будут обработаны пути, которые являются наиболее “похожи” на минимальный путь. A* отличается от жадного алгоритма т.к. g(x) - стоимость пути от начальной вершины, а не от предыдущей.

Свойства алгоритма:

A* является полным, т.е. существующее решение всегда будет найдено.

A* оптимально эффективен для заданной эвристики h. Т.е. A* гарантированно исследует минимально возможное количество узлов по сравнению с любым другим алгоритмом. [1][2][5]

Недостатком A* является неоптимальность на некоторых типах графов: например, можно построить граф, где сначала необходимо сделать длинный шаг от конечной точки, и только потом двигаться в ее направлении. В таком случае A* потеряет время на обработку лишних вершин в начале поиска.

Двунаправленный A*

По аналогии с алгоритмом Дейкстры, существует двунаправленная версия A*, использующая две потенциальные функции. В случае использования двунаправленного A*, достигается примерно двукратное ускорение работы в сравнении с односторонним A*.

Рассмотренные выше алгоритмы анализируют граф по ходу работы, т.е. без предобработки. В случае множественного вызова A* для разных пар вершин имеет смысл провести некоторую оптимизацию перед вызовом алгоритмов. Вариация алгоритма с предобработкой представлена в следующем разделе.

Алгоритм ALT

Известно, что алгоритм A* находит точное решение задачи нахождения кратчайшего пути для графа G = (V, E), если потенциальная функция является преемственной и допустимой. При |V| = n время работы алгоритма A* составляет O(n^2). Однако на практике за счет использования нижних оценок пути A* способен значительно уменьшить время нахождения кратчайшего пути по сравнению с алгоритмом Дейкстры. Чем лучше (больше по значению) нижние оценки, тем быстрее работает алгоритм A*.

Рассмотрим алгоритм, состоящий из двух этапов: подготовки и запроса, работающий на основе алгоритма A* и использующий иную потенциальную функцию.

Подготовка:

Необходимо взять некоторое количество вершин, как ориентиры;

Для каждой выбранной вершины необходимо посчитать и запомнить кратчайшие пути до остальных.

Запрос:

Запускается алгоритм A*;

Если ребро является частью кратчайшего пути между текущей вершиной и ориентиром, полученным на этапе подготовки, то в первую очередь предпринимается попытка пойти по нему.

Для нижних оценок пути используется неравенство треугольника. Пусть A - ориентир, тогда:

- $dist(v,w) \geqslant dist(A,w) - dist(A,v)$
- $dist(v,w) \geqslant dist(v,A) - dist(w,A)$
- $dist(v,w) \geqslant \max\{dist(A,w) - dist(A,v), dist(v,A) - dist(w,A)\}$

Таким образом, вычисленное значение $\boldsymbol{\pi}L(v)$ определяет потенциал текущей вершины v как наибольшую нижнюю оценку кратчайшего пути от этой вершины до целевой вершины t.

Потенциальная функция, удовлетворяющая условию $dist(v, t) \geqslant \boldsymbol{\pi}L(v) \geqslant 0$, называется допустимой. Такая функция не переоценивает значение кратчайшего пути. Неравенство треугольника и приведенные формулы гарантируют допустимость функции.

Неравенство треугольника и приведенные формулы также гарантируют свойство преемственности:

$dist(s, v) + \boldsymbol{\pi}L(v) \geqslant \boldsymbol{\pi}L(s)$ - это своеобразная форма записи неравенства треугольника для стартовой вершины s, текущей вершины v и целевой вершины t через потенциальные функции.

Справедливость условия вытекает из цепочки неравенств:

$$d(s, v) + \boldsymbol{\pi}L(v) \geq \boldsymbol{\pi}L(s)$$

$$d(s, v) + [d(l, t) – d(l, v)] \geqslant [d(l, t) - d(l, s)]$$

$$d(s, v) + d(l, t) - d(l, t) + d(l, s) \geqslant d(l, v)$$

$$d(s, v) + d(l, s) \geqslant d(l, v)$$

Аналогично доказывается условие для ориентированного графа путем рассмотрения различных вариантов направленности дуг, образующих анализируемый треугольник. [4]

Проблемы выбора ориентиров

Вершины-ориентиры необходимо выбирать таким образом, чтобы максимизировать число острых углов между потенциальными начальными точками, ориентирами и конечными точками.

Хорошим ориентиром можно считать точку, которая находится "до" s, в таком случае не будет общих ребер на кратчайшем пути, или "за" t, что обеспечит острый угол и, соответственно, меньшее отклонение от предварительно вычисленного кратчайшего пути от искомого (рис. 1). Хорошими будут

ориентиры, которые будут удовлетворять этим условиям в большинстве запросов.



Рисунок 1.

Таким образом, напрашивается вывод о том, что необходимо выбирать ориентирами точки, которые находятся на краях транспортной сети.

Случайный выбор

Набор узлов-ориентиров выбирается случайно из всех доступных вершин графа.

Плоскостный

Карта делится на несколько секторов одинаковой площади и ориентиром выбираются самые удаленные точки каждого из секторов. Такой способ подходит, если используется граф относительно правильной формы. На практике обычно используют оптимизированную версию

Избирательный

Ориентиры добавляются на граф по одному. Первый ориентир определяется случайным, в чем и заключается недостаток метода, в то время как последующие выбираются с оглядкой на то, какие вершины графа плохо покрыты текущим набором ориентиров.

Поиск максимального покрытия

Избирательный способ можно улучшить путем увеличения в несколько раз количества отбираемых ориентиров, с последующим отсеиванием лишних, что минимизирует время последующих запросов. В этом случае задача выбора ориентиров становится NP-полной.

Такой вариант является наилучшим, но у него наиболее медленное время предобработки. [4]

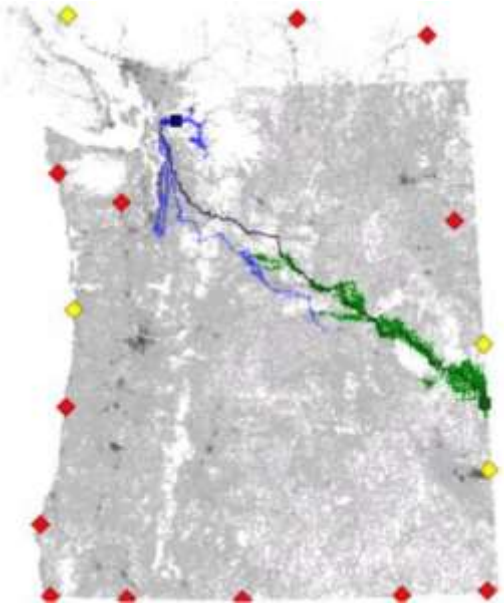


Рисунок 2. Иллюстрация работы алгоритма ALT.

Проектирование Pathfinder, работа программы

Для анализа алгоритмов был разработан инструмент Pathfinder. Pathfinder является клиент-серверным приложением для анализа алгоритмов поиска путей в компьютерных сетях. Данные для программы берутся из OpenStreetMap и преобразовываются в модель данных, пригодную для обработки программой.



Рисунок 3. Интерфейс клиента программы Pathfinder

Интерфейс приложения

При запуске приложения пользователь должен выбрать город, транспортная сеть которого будет использоваться в работе. Затем необходимо установить две точки, между которыми необходимо построить кратчайший маршрут.

Кнопка Find the path

запускает алгоритм, выбираемый через выпадающее меню в верхней части интерфейса.

Кнопка swap points

меняет местами точки A и B на графе. Это может повлиять на итоговый путь в ориентированных графах.

Кнопка clear points

удаляет точки с графа.

Кнопки + и -

отвечают за масштаб графа.

Для того, чтобы найти кратчайший путь между двумя точками, необходимо установить эти точке на карте, затем нажать кнопку “Find the path”, которая запустит алгоритм. Затем, после того как алгоритм закончит работу, в верхней части экрана появится модальное окно, в котором будет написано время, за которое алгоритм нашел кратчайший путь.



Рисунок 4. Пример карты и итогового пути.

Анализ алгоритмов с помощью Pathfinder

Для исследования был выбран город Санкт-Петербург. Были взяты 3 пары точек на разном расстоянии - 35км, 20км и 7.5км. Далее были произведены проверки различных алгоритмов Дейкстры и их улучшений.

Результаты тестирования на расстоянии 35км (рис. 5) приведены в таблице 1:



Рисунок 5. Маршрут для точек, удаленных на 35 км.

Алгоритм	Время работы, сек
Алгоритмы Дейкстры	0.75
Алгоритм Дейкстры с ранней остановкой	0.635
Двунаправленный алгоритм Дейкстры	0.238
A*	0.372
Двунаправленный A*	0.444

ALT	0.093
Двунаправленный ALT	0.031

Таблица 1. Результаты тестирования на 35км.

Результаты тестирования на расстоянии 20км (рис. 6) приведены в таблице 2:



Рисунок 6. Маршрут для точек, удаленных на 20 км.

Алгоритм	Время работы, сек
Алгоритмы Дейкстры	0.768
Алгоритм Дейкстры с ранней остановкой	0.48
Двунаправленный алгоритм Дейкстры	0.236
A*	0.15
Двунаправленный A*	0.144
ALT	0.041

Двунаправленный ALT	0.028

Таблица 2. Результаты тестирования на 20км.

Результаты тестирования на расстоянии 7.5км (рис. 7) приведены в таблице 3:

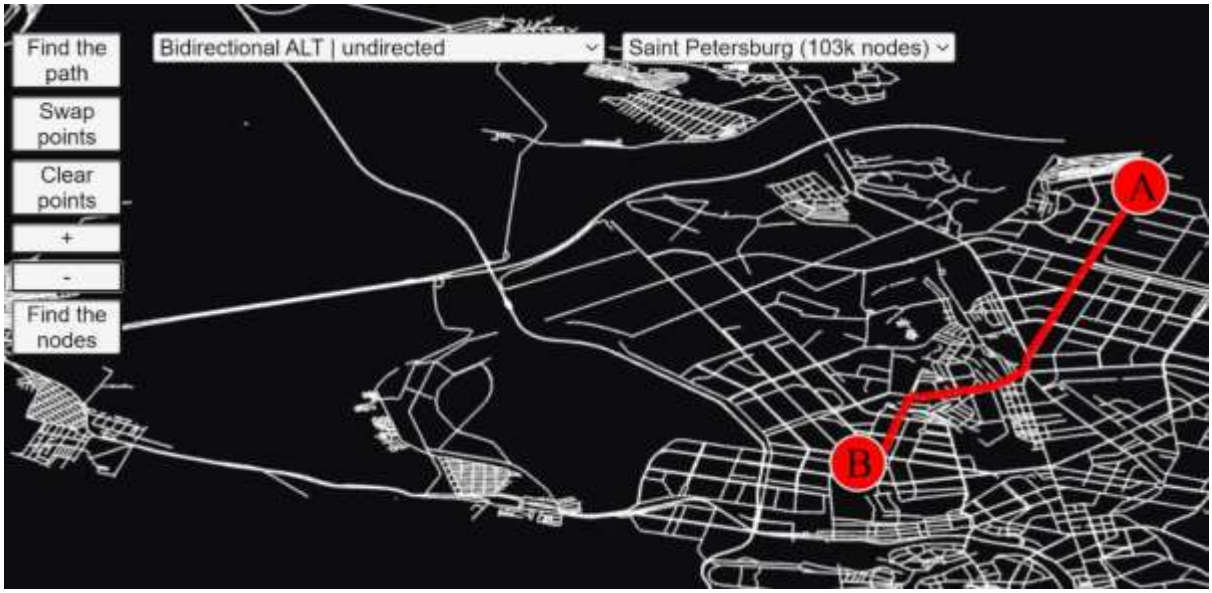


Рисунок 7. Маршрут для точек, удаленных на 35 км.

Алгоритм	Время работы, сек
Алгоритмы Дейкстры	0.750
Алгоритм Дейкстры с ранней остановкой	0.076
Двунаправленный алгоритм Дейкстры	0.049
A*	0.018
Двунаправленный A*	0.027
ALT	0.012
Двунаправленный ALT	0.02

Таблица 3. Результаты тестирования на 7.5км.

Экспериментальные данные показывают, что, как и ожидалось, алгоритм Дейкстры значительно менее эффективен, чем другие алгоритмы. Также можно заметить, что в условиях города двунаправленные алгоритмы часто работают медленнее однонаправленных из-за неоптимального строения графа.

Заключение

Еще из теоретических сведений было очевидно, что различные неинформированные варианты алгоритма Дейкстры не подходят для обработки больших дорожных сетей. На практике результат оказался тот же. Накладные расходы на использование таких алгоритмов слишком высоки, чтобы использовать на больших графах, которые могут насчитывают сотни тысяч вершин.

A* и его двусторонняя модификация достаточно неплохие для дорожных сетей, в частности для графа сети до 100 тысяч вершин. Для бóльших графов область работы алгоритма также начинает увеличиваться экспоненциально.

ALT - самый эффективный алгоритм по времени обработки запроса, так как является информированным. Однако и время предобработки не столь велико: для дорожной сети Санкт-Петербурга - 1 минута 29 секунд, для дорожной сети Лондона - 5 минут 47 секунд.

Литература

1. Амит Пател. Записки про A*. [http://theory.stanford.edu/~amitp/GameProgramming/#pathfinding].
2. Дэниел Деллинг, Питер Сандерс, Доминик Шультес и Доротея Вагнер, Highway Hierarchies Star⋆. [http://algo2.iti.kit.edu/schultes/hwy/hhStarSubmit.pdf].
3. Эдсгер Вибе Дейкстра, A note on two problems in connexion with graphs. [http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf].
4. Фабиан Фукс, О предварительной обработке ALT-алгоритма. [http://www.fabianfuchs.com/fabianfuchs_ALT.pdf].

5. Рина Дехтер, Джудеа Пирл, Обобщенные стратегии поиска лучшего первого и оптимальность A *. [https://dl.acm.org/doi/10.1145/3828.3830].
6. Дана С. Нау, Экспертные компьютерные системы. [https://www.cs.umd.edu/~nau/papers/nau1983expert.pdf].

Literature

1. Amit Patel, Amit’s A* pages. [http://theory.stanford.edu/~amitp/GameProgramming/#pathfinding]
2. Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner, Highway Hierarchies Star⋆ [http://algo2.iti.kit.edu/schultes/hwy/hhStarSubmit.pdf]
3. E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, [http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf]
4. Fabian Fuchs, On preprocessing the ALT-Algorithm [https://i11www.iti.kit.edu/extra/publications/bdd-afdg-09.pdf]
5. Rina Dechter , Judea Pearl, Generalized best-first search strategies and the optimality of A* [https://dl.acm.org/doi/10.1145/3828.3830]
6. Dana S. Nau, Expert computer systems. [https://www.cs.umd.edu/~nau/papers/nau1983expert.pdf]