ISSN 2225-9309 (Print)
2026. № 1 (35). С. 41–45

Молодежный
Вестник УГАТУ

http://journal.uust.ru

УДК 519.6

doi 10.54708/22259309_2026_13541

Гибридный алгоритм A*-Дейкстры для динамически изменяющихся графов

С. Д. ЛЕУШЕВ[1]

[1]pozzilivs@gmail.com

[1]ФГБОУ ВО «Уфимский университет науки и технологий» (УУНиТ)

Аннотация. В статье приведено описание работы гибридного алгоритма A*-Дейкстры. Показано, как можно реализовать алгоритм на языке программирования C++. Описано возможное применение этого алгоритма в программном обеспечении.

Ключевые слова: поиск пути; граф; алгоритм Дейкстры; A*; A со звездой; C++.

ВВЕДЕНИЕ

Задача о поиске кратчайшего пути в графе является достаточно популярной. Например, в любых картографических сервисах для построения маршрута не обойтись без алгоритма, который будет искать минимальный путь от точки к точке.

Алгоритм Дейкстры является одним из самых распространённых способов поиска пути в графе. Он используется при решении обширного списка задач. Например, алгоритм Дейкстры используется в социальных сетях для предложения списка возможных знакомых. Алгоритм находит предполагаемых «друзей» благодаря поиску кратчайшего пути между пользователями.

Алгоритм A* (A со звездой) также является достаточно популярным алгоритмом поиска пути в графе. Он быстрее Дейкстры за счёт эвристики – оценки «полезности» вершин графа. Так, A* можно использовать в телефонной сети, где каждая линия имеет свою пропускную способность. Можно представить город в виде графа, где вершины соответствуют коммутационным станциям, рёбра – линиям передач, а веса рёбер – пропускной способности. Тогда создание наилучшего сигнала между абонентами можно отнести к задачам по поиску кратчайшего пути в графе.

Для статического графа идеальным решением может послужить алгоритм A* за счёт своей малой скорости работы и высокой точности. Для динамического графа A* может не подойти, так как с изменением длины рёбер может неверно посчитаться эвристика. Тогда можно использовать алгоритм Дейкстры, но из-за достаточно большого времени работы он не является самым выгодным вариантом.

Тогда возникает проблема: как найти кратчайший путь в динамически изменяющемся графе.

ОПИСАНИЕ И РАЗРАБОТКА АЛГОРИТМА

Вариантом решения данной проблемы является создание гибридного алгоритма A*- Дейкстры. Суть этого алгоритма заключается в том, что все статические рёбра мы рассматриваем с помощью алгоритма A*, а динамические (то есть подвергаемые изменениям) рёбра – с помощью Дейкстры. Такой гибридный подход позволяет эффективно комбинировать преимущества обоих алгоритмов: скорость A* за счёт эвристического поиска на статических участках графа и гарантированную точность Дейкстры на динамически изменяемых рёбрах. Это особенно полезно в средах, где лишь часть графа подвержена частым изменениям, так как минимизирует перерасчёт маршрутов и сохраняет высокую производительность.

Таким образом, предложенный гибридный алгоритм начинает работу как стандартный A*, используя эвристическую функцию (например, манхэттенское расстояние) для эффективного поиска пути в статической части графа. Однако при обнаружении динамического ребра алгоритм переключается в режим Дейкстры, временно отказываясь от эвристической оценки. При просчёте динамического ребра алгоритм обратно переключается на эвристический подсчёт оценки.

Этот подход имеет существенный недостаток: при обработке динамических рёбер отсутствие эвристического направления приводит к исследованию большинства вариантов в зоне изменений.

Рассмотрим особенности реализации алгоритма на языке программирования C++. Для хранения графа будем использовать класс Graph, включающий в себя методы для работы с ним (рис. 1). Для хранения вершин используем класс Point (рис. 2).

```
class Graph {
public:
  void AddEdge(Point a, Point b, bool is_dynamic = false);

  int GetSize();

  const std::unordered_map<Point, double, Point::Hash>& GetNeighbors(
      const Point& node);

  void UpdateEdge(Point from, Point to, double new_weight);

  bool IsDynamicEdge(const Point& from, const Point& to) const;

  bool VertexInDynamicEdge(const Point& point) const;
private:
  // Хранение рёбер: {from: {to: weight}}
  std::unordered_map<Point, std::unordered_map<Point, double, Point::Hash>,
                     Point::Hash>
      edges_;
  int vertices_count_ = 0;

  std::unordered_set<Point, Point::Hash> dynamic_vertices_;

  // Хеш-функция для пары точек
  struct PairHash {
    size_t operator()(const std::pair<Point, Point>& p) const {
      return Point::Hash()(p.first) ^ Point::Hash()(p.second);
    }
  };

  std::unordered_set<std::pair<Point, Point>, PairHash> dynamic_edges_;
};
```

Рис. 1. Класс Graph

```
class Point {
public:
  int x = 0;
  int y = 0;
  bool in_dynamic_edge = false;
  Point() = default;
  Point(int x, int y) : x(x), y(y) {};

  bool operator==(const Point& other) const {
    return x == other.x && y == other.y;
  }

  bool operator!=(const Point& other) const {
    return x != other.x || y != other.y;
  }

  bool operator<(const Point& other) const {
    return x < other.x && y < other.y;
  }

  // Для использования Point в unordered_map
  struct Hash {
    size_t operator()(const Point& p) const {
      return std::hash<int>()(p.x) ^ (std::hash<int>()(p.y) << 1);
    }
  };
};
```

Рис. 2. Класс Point

Сам гибридный алгоритм будет принимать на вход в качестве аргументов ссылку на граф, а также ссылки на начальную и конечную точки. Функция будет рассматривать наиболее пригодные для конечного пути вершины. Для каждой выбранной вершины алгоритм рассматривает рёбра, исходящие из этой вершины. При рассмотрении ребра алгоритм уточняет, какое оно: статическое или динамическое. Если ребро оказывается статическим, то в хранилище приоритетных вершин мы кладём эвристическое расстояние до соединённой ребром вершины. Иначе если ребро динамическое, то в хранилище мы кладём только расстояние до соединённой вершины. Такой способ соотношения путей позволяет нам давать более высокий приоритет динамическим рёбрам, чтобы рассматривать их в первую очередь. Возвращать функция будет построенный путь от стартовой до конечной вершины (рис. 3).

```
std::vector<Point> HybridAStarDijkstra(Graph& graph, const Point& start,
                                       const Point& goal) {
  // Приоритетная очередь: пары (f_value, точка)
  std::priority_queue<std::pair<double, Point>, std::vector<std::pair<double, Point>>,
                      std::greater<std::pair<double, Point>>>
      open_set;
  open_set.push({0, start});

  std::unordered_map<Point, Point, Point::Hash> came_from;
  std::unordered_map<Point, double, Point::Hash> g_values;
  g_values[start] = 0;

  while (!open_set.empty()) {
    double value = open_set.top().first;
    auto current = open_set.top().second;
    open_set.pop();

    current.in_dynamic_edge = graph.VertexInDynamicEdge(current);

    if (current == goal) {
      // Восстановление пути
      std::vector<Point> path;
      Point node = goal;
      while (node != start || came_from.count(node)) {
        path.push_back(node);
        node = came_from[node];
      }
      path.push_back(start);
      std::reverse(path.begin(), path.end());

      return path;
    }

    for (const auto& neighbor_pair : graph.GetNeighbors(current)) {
      Point neighbor = neighbor_pair.first;
      double edge_cost = neighbor_pair.second;
      double new_g = g_values[current] + edge_cost;

      if (!g_values.count(neighbor) || new_g < g_values[neighbor]) {
        came_from[neighbor] = current;
        g_values[neighbor] = new_g;

        double f_value;
        if (graph.IsDynamicEdge(current, neighbor)) {
          // Режим Дейкстры (h=0)
          f_value = new_g;
        } else {
          // Режим A* с эвристикой
          f_value = new_g + Heuristic(neighbor, goal);
        }

        open_set.push({f_value, neighbor});
      }
    }
  }
  return {};  // Путь не найден
}
```

Рис. 3. Реализация алгоритма

ЭФФЕКТИВНОСТЬ РАБОТЫ ПРИЛОЖЕНИЯ

Было проведено множество тестов на графах со 100,000 (ста тысячами) рёбер (рис. 4) и 10,000,000 (десятью миллионами) рёбер (рис. 5) и сравнение их результатов с алгоритмом Дейкстры. Было выявлено, что гибридный алгоритм демонстрирует высокую эффективность. Расстояния, найденные гибридным методом, отличаются от значений, полученных с помощью алгоритма Дейкстры, не более чем на 7 %. При этом среднее время работы сократилось в 3 раза, что делает гибридный алгоритм особенно полезным для обработки больших графов.

```
Гибридный алгоритм:     (Длина пути - 1865.44575) Время работы: 93
Дейкстра:       (Длина пути - 1865.44575) Время работы: 456
Обновление рёбер...
Гибридный алгоритм:     (Длина пути - 1802.90818) Время работы: 92
Дейкстра:       (Длина пути - 1802.90818) Время работы: 479
```

Рис. 4. Время и точность работы гибридного алгоритма и алгоритма Дейкстры на графе на 100,000 рёбер

```
Гибридный алгоритм:     (Длина пути - 934.19008) Время работы: 1822
Дейкстра:       (Длина пути - 922.38251) Время работы: 25756
Обновление рёбер...
Гибридный алгоритм:     (Длина пути - 908.19008) Время работы: 1365
Дейкстра:       (Длина пути - 908.19008) Время работы: 25848
```

Рис. 5. Время и точность работы гибридного алгоритма и алгоритма Дейкстры на графе на 10,000,000 рёбер

Эти результаты показывают, что предложенный метод успешно сочетает в себе скорость и относительную точность (что делает алгоритм невыгодным для задач, требующих точный ответ), предлагая практическое преимущество для задач, требующих быстрой обработки данных без значительной потери качества.

Неточность гибридного алгоритма связана с тем, что возможен вариант уменьшения длины ребра по сравнению с его минимально допустимой длиной (например, расстояние от точки до точки – 2, а длина ребра – 1). Из-за этого возникает неточность при вычислении эвристики, т. к. A* не учитывает фактические длины рёбер. Поэтому некоторые динамические рёбра, которые должны быть включены в путь, могут быть проигнорированы из-за уменьшения их по-настоящему возможного минимального пути.

ЗАКЛЮЧЕНИЕ

В данной статье представлен алгоритм поиска наикратчайшего пути в динамически изменяющихся графах, основанный на гибридизации алгоритмов A* и Дейкстры. Полученный метод позволяет эффективно обрабатывать изменения длин рёбер в графе, совмещая преимущества эвристического поиска A* и гарантированную точность Дейкстры.

Реализация алгоритма на языке C++ продемонстрировала его высокую производительность при работе с динамическими графами по сравнению с алгоритмом Дейкстры. Реализация может быть улучшена использованием более оптимизированных структур данных для хранения графа, поиска оптимальных рёбер и расстояний до них.

Дальнейшие преобразования алгоритма могут происходить с внедрением кэширования статических участков для ещё большего уменьшения времени работы или просмотра всех динамических рёбер для улучшения точности. Также оптимальным направлением является внедрение многопоточности и распараллеливание вычислений для ускорения работы алгоритма.

Разработанный алгоритм может быть применён в различных областях, включая маршрутизацию в транспортных сетях, создание и обновление путей в робототехнике и моделирование потоков данных. Его эффективность делает его перспективным решением для

задач, требующих быстрой обработки динамически изменяющихся графов и не требующих высокой точности.

СПИСОК ЛИТЕРАТУРЫ

1. **Дейкстра, Э.** A note on two problems in connexion with graphs // Numer-ische mathematik. – 1959. – Т. 1. – №. – 1. С. 269-271.

2. **Кудияров, А. В.** Алгоритм Дейкстры и его применение для решения оптимизационных задач / А. В. Кудияров // Достижение национальных целей устойчивого развития страны как условие повышения качества жизни населения. — Калуга: ИП Карпов А.Н., 2023. – С. 43-48.

3. **Поречный, С. С., Житникова, Н. И., Шерыхалина, Н. М.**, **Ураков, А. Р.** Дискретная математика / С. С. Поречный, Н. И. Житникова, Н. М. Шерыхалина, А. Р. Ураков – Уфа: РИК УГАТУ, 2019. – 400 с.

4. **Рафгарден, Т.** Совершенный алгоритм. Графовые алгоритмы и структуры данных. – СПб.: Питер, 2019. – 256 с.

5. **Шварц, Р.** Дискретная математика для программистов. – М.: ДМК Пресс, 2009.

6. **Шукуров, А. Э.** Алгоритм Дейкстры: оптимальный поиск кратчайшего пути в графах с весами ребер / А. Э. Шукуров // Наука и молодёжь: новые идеи и решения: Мат-лы XVII Международной научно-практической конференции молодых исследователей, Волгоград, 30–31 марта 2023 года. – Волгоград: Волгоградский государственный аграрный университет, 2023. – С. 295-297.

7. **Wayahdi, M. R. Greedy**, A-Star, and Dijkstra's Algorithms in Finding Shortest Path / M. R. Wayahdi, S. H. N. Ginting, D. Syahputra // International Journal of Advances in Data and Information Systems. – 2021. – Т. 2. – №. 1. – С. 45-52.

ОБ АВТОРАХ

ЛЕУШЕВ Сергей Дмитриевич, студент ПРО ИИМРТ УУНиТ.

METADATA

Title: Hybrid A*-Dijkstra algorithm for dynamically changing graphs.

Authors: S. D. Leushev

Affiliation:

Ufa University of Science and Technology (UUST), Russia.

Email: pozzilivs @gmail.com

Language: Russian.

Source: Molodezhnyj Vestnik UGATU (scientific journal of Ufa University of Science and Technology), no. 1 (35), pp. 41-45, 2026. ISSN 2225-9309 (Print).

Abstract: The article describes how the A*-Dijkstra hybrid algorithm works. It is shown how the algorithm can be implemented in the C++ programming language. The possible application of this algorithm in software is described.

Key words: pathfinding, graph, Dijkstra's algorithm, A*, A with a star, C++.

About authors:

LEUSHEV Sergey Dmitrievich, student PRO IIMRT UUST.