ПОСТРОЕНИЕ ПРОСТРАНСТВЕННОГО ДОРОЖНОГО ГРАФА

М.А. Степанов [1], М.Е. Семёнов [2]
[1] МБОУ лицей при Томском политехническом университете, Россия, г. Томск, ул. А. Иванова, 4, 634028
[2] Томской политехнический университет, Россия, г. Томск, пр. Ленина, 30, 634050
E-mail: sme@tpu.ru

Непрерывное позиционирование и навигация человека на улице и в помещении с помощью мобильного приложения является актуальной задачей. В Томском политехническом университете (ТПУ) разработан веб-сервис для поиска и бронирования аудиторий университета (maps.tpu.ru). Одним из направлений развития данного веб-сервиса является создание навигации внутри учебных корпусов, а затем сквозной (бесшовной) навигации в кампусе университета. Цель данной работы – разработка алгоритма и прототипа программного обеспечения для построения дорожного графа с учетом дополнительных условий.

В качестве входных данных использованы поэтажные планы зданий, которые хранятся в векторном формате (файлы с расширением *.svg). Формально для i-го этажа, i = 1, 2, …, k известны координаты дверей $(x_{ij}, y_{ij}) \in \boldsymbol{R}$, j=1, 2, …, n во внутренние помещения здания. Требуется обеспечить навигацию пользователя между любыми двумя помещениями.

Для решения поставленной задачи предлагаем использовать графовую модель, в которой вершины – это точки плоскости (двери), а ребра – линии между парами вершин, вес ребра – евклидовое расстояние между смежными вершинами. Будем считать, что передвижение между внутренними помещениями осуществляется через систему связанных коридоров и лестниц по прямолинейным отрезкам на разных уровнях. Для учета этих условий введем *вершины ветвления*, а также припишем каждой вершине дополнительную характеристику – номер этажа $z_{ij} \in \boldsymbol{Z}$. Далее, применим к i-му графу, i = 1, 2, …, k алгоритм Прима и найдет минимальное остовное дерево. Затем полученные деревья объединим в дорожный граф через добавление ребер между вершинами ветвления. Окончательно, применим к дорожному графу алгоритм Дейкстры и найдем кратчайший простой путь для любой пары заданных пользователем вершин (рис. 1).

Предложенное решение задачи реализовано на языке Python, с использованием библиотеки matplotlib для визуализации результатов. Верификация найденных остовных деревьев показала, что не всегда полученное дерево удовлетворяет пространственным ограничениям. На данном этапе работы мы вынуждены дополнительно монтировать (rewiring) ребра. Для решения указанной проблемы требуется проведения дополнительных исследований, направленных на разработку алгоритма для задания координат точек ветвления.

Требуется для найти граф, который связывает все терминальные точки и имеет минимальный вес среди всех подобных подграфов, причём *вес графа* – сумма евклидовых расстояний между двумя точками, между которыми существует ребро дорожного графа.

Одно из возможных решений данной задачи – формирование неориентированного взвешенного (*дорожного*) графа G [1, 2]. Дорожный граф – сеть дорог, связывающая заданные точки пространства, а затем поиск минимального остовного дерева [3].

Таким образом, мы имеем множество терминальных точек с координатами (x_i, y_i, z_i), i=1, 2, …, n, между которыми можно определить евклидово расстояние.

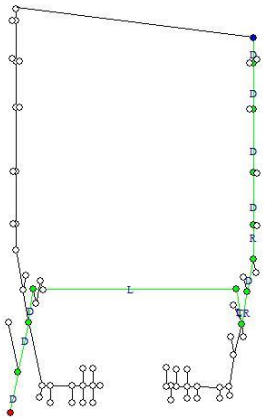

Рис. 1. Пример кратчайшего простого пути

При объедении следует учитывать наличие альтернативных маршрутов в здании, что приводит к появлению циклов в дорожном графе.

Для минимизации веса дорожного графа G допускается использовать дополнительные точки – *точки ветвления* [3, 4]. *Точки ветвления* – вершины дорожного графа, не являющиеся терминальными точками. В работах [1, 2] рассмотрены эвристические алгоритмы построения графа G, которые требуют явного указания точек ветвления пользователем.

В нашем алгоритме будем использовать метод *машинного обучения*. *Машинное обучение* – совокупность методов *искусственного интеллекта (ИИ),* которые направлены не на прямое решение задачи, а на обучение и поиск лучших из всевозможных решений.

Описание алгоритма и блок-схема

Для достижения поставленной цели мы предлагаем следующий алгоритм, основанный на методе *к*-средних соседей машинного обучения:

Шаг 1. Разобьем множество терминальных точек на k непересекающиеся подмножеств так, чтобы между любой парой точек (x_i, y_i) и (x_j, y_j) одного подмножества может быть найден путь с учетом пространственных ограничений (пересекать рёбра многоугольника P запрещено).

Шаг 2. Для каждого i-го подмножества точек найдем *центральную* точку C_i, $i = 1, 2, \ldots, k$, координаты которой определим, как среднеарифметическое значение всех координат точек данного подмножества.

Шаг 3. Найдём новые центры подмножеств C'_i следующим образом.

Для каждого центра найдём множество точек H', принадлежащих хотя бы одному из рёбер многоугольника P, таких что расстояние от всех точек множества H'_i до C'_i меньше, чем расстояние от этих точек H'_i до любых других центров C'_i. Координаты i-го центра будут равны среднеарифметическому значению координат соответствующего подмножества H'_i.

Шаг 4. Повторим шаг 3 до тех пор, пока центральные точки предыдущей итерации не будут совпадать с центральными точками текущей итерации. Полученные центральные точки добавим в множество H'.

Шаг 5. Повторим шаги 2-4 до тех пор, пока не объединим центральные точки в дорожный граф G с учетом пространственных ограничений.

Используемые материалы, методы и оборудование

Разработка программного обеспечения осуществлена в среде *Python* IDLE 3.7. Данная среда разработки свободно распространяется на сайте создателей языка – python.org.

Для визуализации результатов расчетов использована библиотека *matplotlib*.

Форматом входных данных будем использовать формат векторной графики *.svg, используемый на сайте (maps.tpu.ru).

Для поиска путей в построенном графе используется алгоритм Дейкстры [8]. Для его реализации инициализируем массив $\{b_i\}$ записей, содержащих информацию о i-ой вершине дорожного графа. Будем находить кратчайший путь до b_x вершины дорожного графа. Рассмотрим поочередно все вершины b_i, в порядке возрастания их расстояния до точки b_x (b_x рассматривать н будем). Для каждой вершины b_i рассмотрим все вершины b_j. Если путь от b_j через $b_{i\ до}$ b_x короче, чем прямой путь из b_j в b_x, то запишем этот путь в массив. В итоге у нас будем массив, содержащий кратчайшие пути b_x до любой другой вершины.

Поиск пересечения путей осуществляется поиском точки пересечения двух прямых (содержащих отрезки ребра многоугольника P и ребра графа между проверяемыми точками).

Предложенный алгоритм протестирован на модельных выпуклых и невыпуклых полигонах, имеющих различную конфигурацию: I, Г, Т, Н и для различного количества $n \geq 6$ областей.

Заключение

Были рассмотрены различные инструменты для разработки программного обеспечения и выбран наиболее оптимальный вариант.

Провели анализ литературы по теме построения дорожного графа и работы с ним.

На основе метода машинного обучения был разработан алгоритм, способный справиться с поставленной задачей.

Провели аналитический расчёт сложности алгоритма, построили график прироста сложности от увеличения сложности помещения (количества коридоров).

Были проведены тесты, определяющие скорость и точность выполнения задачи.

Программа прошла некоторые простые тесты, чем показала перспективу её дальнейшего использования. Выполнение программы на приведённый выше тестах занимает менее 2 секунд при запущенной библиотеке matplotlib. В лучшем случае алгоритм имеет сложность выполнения O ((N + 20 * n) * K), где N – количество рёбер дорожного графа, K – Количество рёбер многоугольника P, n (обычно не более 3) - количество итераций циклов по нахождению циклов. На постановку одной точки ветвления или ребра у студентов, помогающих с проектом maps.tpu.ru, уходит, в среднем, около 20 секунд (учитывая время на проверку и переходы между этажами) значит, для 128 аудиторий и 127 рёбер уйдет не менее полутора часов. На сайте ТПУ указано 46 корпусов. Следовательно, программа поможет сэкономить, как минимум 69 рабочих часов.

Выводы и предложения по внедрению результатов

В качестве развития данной работы мы планируем построить дорожный граф в рамках замкнутой многоэтажной системы коридоров некоторого здания и разработать алгоритм для нахождения множества субоптимальных маршрутов. При этом в будущем мы не будем ограничиваться наличием только одного входа в аудиторию.

Также планируется перевести входные данные в *.svg формат и провести тестирование программы на всех имеющихся корпусах ТПУ.

Благодаря выбранному языку Python, существует возможность внедрения кода на сайт ТПУ, для дальнейшего его использования учащимися и преподавателями.

Список использованных источников

1. Щеголева Л.В., Воронов Р.В. Построение дорожного графа для маршрутизации мобильного робота в замкнутой системе коридоров // Инженерный вестник Дона. – 2015. – № 3. ivdon.ru/ru/magazine/archive/n3y2015/3168
2. Шакуров А. М. Web-приложение для поиска кратчайшего маршрута внутри здания: бакалаврская работа. ТПУ. –2017. – 85 с.
3. Хайнеман Д., Поояис Г., Сеяков С. Алгоритмы. Справочник с примерами на C, C++, Java и Python. М.: Издательство “Диалектика”. – 2017. – 432 с.
4. Лотарев Д.Т., Уздемир А.П. Преобразование задачи Штейнера на евклидовой плоскости к задаче Штейнера на графе // Автоматика и телемеханика. – 2005. – В. 10. – с. 80–92.
5. Статистический сервис Яндекс.Радар // https://radar.yandex.ru/top_list?type=service&isSearch=true&row_id=yandex-ru-maps&offset=1
6. Блог разработчиков яндекса //https://yandex.ru/company/technologies/routes
7. Онлайн журнал FrequentFlyers.ru // http://www.frequentflyers.ru/ 2016/03/23/svo_app/
8. Поляков К.Ю. Динамические структуры данных в языке Си // 2009 с. 35