|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***Алгоритм поиска кратчайшего безопасного пути на электронной карте местности***

Студент \_\_ИУ7-84\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Д.И. Монахов**\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**М. Ю. Барышникова\_

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**Ю.В. Строганов**\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

Введение

Проблема определения маршрута была актуальной еще в древности и является актуальной по сей день. Раньше человеку приходилось использовать карту для поиска оптимального по времени пути, чтобы достичь желаемого пункта назначения. В наши дни мало что изменилось, нам по-прежнему нужно перемещаться и очевидное желание сэкономить время и топливо никуда не исчезло. К тому же начинают появляться роботизированные средства передвижения, такие как беспилотные автомобили, беспилотные летательные аппараты (БПЛА). В военной промышленности также производятся крылатые ракеты с рельефометрическим наведением, которое использует карты для прокладывания маршрута полета таким образом, чтобы не попасть под огонь враждебных средств противоракетной обороны. Согласно проведенным экспериментам, данный тип наведения показывает отличные результаты в преодолении средств защиты и позволяет ракете обладать высокими показателями живучести и помехозащищенности [[1](https://babel.hathitrust.org/cgi/pt?id=mdp.39015013772390&view=1up&seq=349)]. Наведение по рельефу в комплексе с инерциальными средствами навигации позволяет аппарату быть практически невосприимчивым к средствам радиоэлектронной борьбы, т.к. исключает использование радиосвязи в навигации. Автоматизация процесса прокладывания маршрута произошла и в гражданской сфере: практически на каждом мобильном компьютере (смартфоне) имеются программно-аппаратные средства для отображения картографических данных, геопозиционирования пользователя и помощи в прокладывании маршрута. Также технология используется в бизнесе. Самый яркий пример – это логистика. Оптимизация пути является основной задачей логистики, от успешности выполнения которой зависит прибыль компании. Стоимость маршрута может зависеть от множества параметров, таких как: время, расстояние, расход топлива и т.д. А сами эти параметры могут зависеть от характеристики ландшафта, по которому прокладывается путь. [[2](https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=82FD69BD60193407E0159A0844367A21?doi=10.1.1.164.8916&rep=rep1&type=pdf)]

Все рассмотренные выше приложения требуют программных средств поиска кратчайшего пути на электронной карте местности, который удовлетворяет некоторым заданным пользователем требованиям. Например, гражданин хочет избежать использования нерегулируемых перекрестков, логистическая компания не готова оплачивать платные дороги, а крылатая ракета не должна попасть в зону покрытия вражеской противоракетной обороны. Исходя из этого, данная работа предлагает рассмотреть применение алгоритмов поиска кратчайшего пути с возможностью исключения неблагоприятных участков пути. Под безопасностью пути будет пониматься оценка пользователя программного продукта о допустимости включения участка в конечный план маршрута.

И так, целью работы является модификация алгоритма поиска кратчайшего пути для решения проблемы прокладывания маршрута на карте местности с учетом требований пользователя по исключению специфических участков.

Для достижения поставленной цели необходимо решить следующие задачи:

* исследовательская (выбор существующего алгоритма поиска кратчайшего пути)
* конструкторская (модификация выбранного алгоритма для решения проблемы)
* технологическая (выбор средства отображения картографических данных, выбор среды и языка программирования, описание основных моментов разработки).

Аналитический раздел

Рассмотрим существующие алгоритмы поиска кратчайшего пути. Все приведенные ниже алгоритмы основываются на представлении пространства в виде графа, в котором вершины являются точками пространства, а рёбра отображают наличие возможности переместится из одной точки в другую.

Сеточная карта (дискретное рабочее поле)

Для начала следует рассмотреть алгоритмы, работающие с сеточным представлением карты местности.

Карту можно представить в виде сетки (или таблицы), состоящей из проходимых и непроходимых ячеек. Каждая ячейка соответствует некоторой области в реальном мире и имеет две координаты (x, y), совпадающие с положением ячейки в таблице.

Рассмотрим построение такой сетки на примере фрагмента карты. Непроходимыми будем считать те ячейки, которые более чем на 50% заполнены какими-либо объектами реального мира. Эти ячейки закрасим красным цветом, а затем выделим получившуюся таблицу как отельное изображение (рис 1).



Рисунок 1. Пример создания сеточной карты

Такое представление карты уже достаточно легко анализировать. Очевидно, что в данном примере использовалась слишком низкая частота дискретизации пространства, в конечном программном продукте следует повысить точность.

**Алгоритм Ли**

Существуют алгоритмы, которые работают с форматом входных данных в виде дискретного рабочего поля. Самый известный такой алгоритм – это алгоритм Ли. Рассмотрим принцип работы данного алгоритма. Он основан на алгоритме поиска в ширину и его сильной стороной является гарантия нахождения кратчайшего пути между двумя точками, если такой путь существует.

Входные данные: дискретное рабочее поле с помеченными непроходимыми ячейками, пара точек (старт и финиш).

Выходные данные: последовательность ячеек, представляющая кратчайший путь из точки старта в точку финиша, если такой путь существует.

Работа алгоритма делится на три этапа: инициализацию, распространение волны и восстановление пути.

1) Инициализация: построение множества ячеек, запоминание стартовой и финишной ячейки.

2) Распространение волны: порождается шаг в соседнюю ячейку с проверкой проходимости и принадлежности к уже рассмотренным ячейкам.

3) Соседняя ячейка помечается как рассмотренная и получает в соответствие число, равное количеству шагов от стартовой ячейки.

4) Из каждой рассмотренной на данном шаге ячейки порождается новая волна, которая трактует данную ячейку как стартовую.

5) При достижении конечной ячейки проводится восстановление пути. Начиная с конечной ячейки ищется соседняя ячейка, имеющая в соответствие число на единицу меньшее чем у текущей. Последовательность этих ячеек и становится результатом работы алгоритма (после инверсии составленного списка).

Ниже приведен псевдокод алгоритма:

// Инициализация

Пометить стартовую ячейку

d := 0

// Распространение волны

ЦИКЛ

ДЛЯ каждой ячейки loc, помеченной числом d

пометить все соседние свободные непомеченные ячейки числом d + 1

КЦ

d := d + 1

ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны)

// Восстановление пути

ЕСЛИ финишная ячейка помечена

ТО

перейти в финишную ячейку

ЦИКЛ

выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке

перейти в выбранную ячейку и добавить её к пути

ПОКА текущая ячейка — не стартовая

ВОЗВРАТ путь найден

ИНАЧЕ

ВОЗВРАТ путь не найден

Следует заметить, что соседство ячеек может быть двух вариантов: в смысле окрестности Мура и окрестности фон Неймана. В первом случае, в отличие от второго, рассматриваются не только ближайшие вертикальные и горизонтальные ячейки, но также диагональные. Принцип работы алгоритма никак не зависит от определения соседства.

Однако у алгоритма Ли есть недостаток - данный алгоритм является очень требовательным ко времени и памяти. Временная сложность и требование к памяти для сетки с размерами M x N составляет O(M x N), т.е. с увеличением размеров исследуемой области требования алгоритма растут в квадратической зависимости.

Способы сократить время выполнения:

1) выбирать в качестве начальной точки ту, которая находится дальше от центра

2) распространять волну и из начальной точки и из конечной

3) исследовать как можно меньшую прямоугольную область, включающую входные точки

Можно сказать, что данный алгоритм слабо подходит к поставленной задаче. Не смотря на гарантированность результата, он является слишком медленным и требовательным к памяти. Следует рассмотреть другие алгоритмы поиска кратчайшего пути, которые будут более эффективны по этим параметрам [[3](http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf)].

Преобразование в граф

Теперь рассмотрим способ превращения карты местности в графовое представление. Реальный мир является не дискретным пространством, что исключает возможность бесконечно точного хранения его реплики в памяти компьютера. Однако для решения задач прокладывания маршрутов это не понадобится. Достаточно разбить карту на равномерную сетку, где каждая проходимая ячейка представляется вершиной графа, а непроходимые участки (например, стены) удаляют соответствующие рёбра между вершинами (см. рис 1). Данный подход является расширением дискретного рабочего поля, который был описан выше.



Рисунок 2. Графовое представление пространства

Такое представление пространства позволит применить классические алгоритмы обхода графов, которые и будут рассмотрены ниже.

Поиск в ширину

Поиск в ширину – один из простейших алгоритмов обхода графа. Он является базой для многих продвинутых алгоритмов поиска пути. Например, алгоритм Прима, алгоритм Дейкстры и алгоритм Ли используют те же принципы, что и поиск в ширину.

Получив на вход граф и начальный узел, поиск в ширину систематично обойдет грани графа, чтобы найти все узлы, достижимые из начального. алгоритм вычислят расстояние (минимальное количество граней) из начального узла в каждый достижимый узел и создает дерево, содержащее все достижимые узлы.

Пусть задан невзвешенный ориентированный граф *G*=(*V*,*E*), в котором выделена исходная вершина. Требуется найти длину кратчайшего пути (если таковой имеется) от одной заданной вершины до другой. Частным случаем указанного графа является невзвешенный неориентированный граф, т.е. граф, в котором для каждого ребра найдется обратное, соединяющее те же вершины в другом направлении.

Для алгоритма нам потребуются очередь и множество посещенных вершин *was*, которые изначально содержат одну вершину *s*. На каждом шагу алгоритм берет из начала очереди вершину *v* и добавляет все не посещённые смежные с *v* вершины в *was* и в конец очереди. Если очередь пуста, то алгоритм завершает работу.

Временная сложность алгоритма может быть выражена как O(|E| + |V|), где |E| - количество граней, а |V| - количество узлов графа. При этом значение O(|E| + |V|) может варьироваться между O(|V|) и O(|V|2) в зависимости от строения графа. [[4](https://www.researchgate.net/publication/282488307_Pathfinding_Algorithm_Efficiency_Analysis_in_2D_Grid)]

Ниже приведен псевдокод алгоритма

1. Поместить исходную вершину в очередь

2. Извлечь вершину x из очереди и пометить как посещенный

а. Если узел x является целевым, то завершить поиск как успешный

б. Иначе, добавить в очередь все неотмеченные вершины, которые связаны с x и не находятся в очереди

3. Если очередь пуста, то завершить поиск как неудачный

4. Перейти в п. 2

**Алгоритм Дейкстры**

Алгоритм Дейкстры работает с проблемами поиска кратчайшего пути в направленных взвешенных графах G = (V, E) с неотрицательными весами граней. Алгоритм создает массив узлов S, чьи веса кратчайшего пути из начального узла уже были определены. Алгоритм последовательно выбирает узлы с наименьшим ожидаемым расстоянием, не входящие в S и «релаксирует» все исходящие из этих узлов грани. Алгоритм Дейкстры называют «жадным» алгоритмом, т.к. он всегда выбирает ближайших узел, который следует добавить в S. Худшее время работы алгоритма оценивается как O(|E| + |V|log|V|)|), где |E| - количество граней, а |V| - количество узлов графа.

Данный алгоритм не очень удобно использовать при рассматриваемом способе представления пространства. Все проходимые ячейки пространства должны соединяться гранями с одинаковым весом, т.к. располагаются на одинаковом расстоянии. В случае, если все веса графа равны, то алгоритм Дейкстры перестает чем-либо отличаться от алгоритма поиска в ширину.

**Выводы по алгоритмам семейства поиска в ширину**

Данные алгоритмы являются слишком медлительными для использования в решении поставленной задачи. Основным недостатком алгоритмов семейства поиска в ширину является стремление обойти все узлы графа, не учитывая взаимного расположения начального и конечного узлов. Теоретическая оценка сложности показывает квадратическую зависимость от размеров сетки пространства. Рассмотрим пример выполнения экспериментальной программы.

Для эксперимента использовалась модельная карта местности с одной горизонтальной преградой между начальной и конечной точкой.



Рисунок 3. Пример работы алгоритмов семейства поиска в ширину

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен алгоритм Дейкстры, на правой – алгоритм Ли. Оба алгоритм нашли кратчайший путь (выделен красным). Однако видно, что алгоритмы действовали явно не оптимально – в обоих случаях подавляющее большинство ячеек было посещено. Можно сделать вывод, что на реальных данных алгоритмы будут показывать неудовлетворительную производительность.

**Поиск по первому наилучшему совпадению**

Поиск «лучший - первый» - алгоритм поиска, исследующий граф путём расширения наиболее перспективных узлов, выбираемых в соответствии с указанным правилом. Данное семейство алгоритмов вводит новое понятие – эвристическая функция оценки. Эта функция представляет собой правило, по которому оценивается перспективность очередного узла n. Чаще всего данной оценкой является расстояние в декартовой системе координат, т.е. наиболее перспективным является наиболее близкий к цели узел.

Рассмотрим простейшую реализацию алгоритма «лучший – первый».

1) Создать список открытых узлов OPEN

2) Добавить в OPEN начальный узел

3) Если очередь пуста, то перейти к 9

4) Удалить лучший узел N из OPEN

5) Если N является конечным, то произвести восстановление пути и вернуть путь

6) Создать список соседних узлов для N

7) Оценить каждый соседний узел, добавить их в OPEN и пометить N как родительский

8) Перейти к 3

9) Закончить

Эта реализация алгоритма имеет недостаток – она может не найти путь между двумя узлами, даже если он существует. Например, алгоритм может зациклиться, если зайдет в тупик (ситуация, когда потомок узла является его родителем).

Добавление списка уже посещенных узлов позволит избежать зацикливания в тупиках. [5]

Проведем эксперимент и сравним жадную версию алгоритма «лучший – первый» с алгоритмом, основанным на поиске в ширину. Для эксперимента использовалась модельная карта местности с одной горизонтальной преградой между начальной и конечной точкой.



Рисунок 4. Сравнение жадного алгоритма «лучший-первый» и алгоритма Ли

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен жадный алгоритм «лучший-первый», на правой – алгоритм Ли. Оба алгоритм нашли кратчайший путь (выделен красным). Видно, что алгоритм на левой стороне посетил значительно меньшее количество ячеек. Это связано с тем, что данный алгоритм использует дополнительную информацию о положении цели и учитывает ее с помощью эвристической функции.

Рассмотрим карту другого вида. Добавим на местность так называемую «ловушку» - область, в виде незамкнутой выпуклой фигуры. Ориентируем ее так, чтобы отверстие в стене находилось ближе к начальной точке. Результаты приведены на рисунке 5.



Рисунок 5. Ловушка для жадного алгоритма «лучший-первый»

Видно, что алгоритм Ли (справа) нашел кратчайший путь, а алгоритм «лучший-первый» дал неверный результат. Данный путь не является кратчайшим. Такой результат связан с тем, что алгоритм никак не учитывает фактическую длину пути, а полагается лишь на эвристическую функцию. Последняя заводит алгоритм в ловушку, из которой нужно искать выход.

Данный недостаток делает жадный алгоритм «лучший-первый» неподходящим для решения задачи кратчайшего пути. Не смотря на значительно более высокую эффективность по сравнению с поиском в ширину, отсутствие гарантии нахождения кратчайшего пути вынуждает отказаться от его применения.

**Алгоритм A\***

Ранее был рассмотрен алгоритм, использующий эвристическую функцию для предсказания наиболее перспективного направления обхода графа. Однако алгоритм не давал гарантии того, что результирующий путь будет кратчайшим. Алгоритм A\* призван устранить этот недостаток. Он основывается на том же принципе эвристики, комбинируя сильные стороны алгоритма «лучший-первый» и алгоритма Дейкстры.

Отличия от алгоритма «лучший-первый» заключается в специальной эвристической функции, которая учитывает не только приблизительное расстояние до цели, но и количество шагов до начальной точки.

Общий вид эвристической функции алгоритма A\*:

Здесь f(n) – эвристическая функция, g(n) – количество шагов от старта до узла n, h(n) – оценочное расстояние от узла n до целевого узла.

Алгоритм A\* гарантирует нахождение кратчайшего пути, если таковой существует. Данное свойство делает его подходящим для решения поставленной задачи. Для оценки эффективности проведем эксперимент.



Рисунок 6. Сравнение алгоритма A\* и алгоритма Ли

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен алгоритм A\*, на правой – алгоритм Ли. Оба алгоритм нашли кратчайший путь (выделен красным). Видно, что алгоритм A\* на левой стороне посетил значительно меньшее количество ячеек. Результат все же менее впечатляющий чем у алгоритма «лучший-первый», но такова цена гарантии нахождения кратчайшего пути.

Алгоритм HPA\*

Проблема поиска кратчайшего пути должна решаться в кратчайшее время (а в некоторых приложениях в реальном времени) в условиях ограниченной памяти и процессорного времени. Рассмотренный алгоритм A\* хоть и избегает рассматривания всех ячеек электронной карты, но все равно является достаточно медлительным на больших картах.

Алгоритм HPA\* (Hierarchical Path-Finding A\*) призван решить эту проблему. Он основывается на иерархическом подходе к разбиению пространства и дальнейшем применения обычного A\* не к изначальной проблеме и карте, а к небольшим участкам. Исходное дискретное пространство разбивается на взаимосвязанные кластеры ячеек. До первого запуска поиска пути вычисляются и запоминаются оптимальные пути внутри каждого кластера. Далее путь ищется по абстрактному графу, составленного из кластеров. После этого путь уточняется с помощью рассмотрения путей внутри задействованных кластеров.

Поиск пути состоит из трех этапов. Во-первых, алгоритм находит путь от начальной точки до границы кластера, который ее содержит. Во-вторых, по абстрактному графу ищется путь от начального кластера к конечному. В-третьих, путь завершается нахождением пути от границы конечного кластера к конечной точке. Абстрактный граф строится на основе информации, полученной из электронной карты местности.

Рассмотрим этап первоначальной обработки карты, который производится до первого запуска поиска пути. Карта разбивается на прямоугольные группы ячеек, называемые кластерами (рис. 7). Черные клетки соответствуют непроходимым участкам карты, белые, соответственно, являются проходимыми. Кластеры разделены красными линиями. Разбиение осуществляется без учета характеристик самого пространства, достаточно только задать желаемый размер кластера.

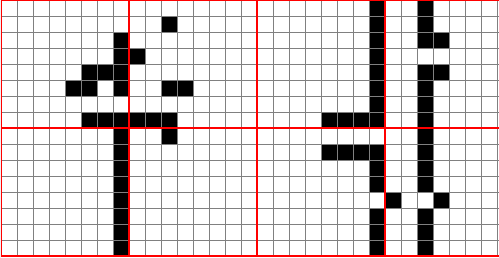


Рисунок 7. Разбиение на кластеры

Для каждой границы между кластерами находится массив входов, которые их соединяют. Под входом понимается наибольший свободный от препятствий сегмент, располагающийся вдоль общей границы двух соседних кластеров c1 и c2. Пусть имеются два смежных отрезка l1 и l2, принадлежащих кластерам c1 и с2 соответственно, и определяющих смежные грани кластеров. Для каждой клетки определим клетку symm(t) как клетку, симметричную клетке t по отношению к границе между кластерами. t и symm(t) прилегают друг к другу и никогда не принадлежат одному и тому же кластеру. Тогда вход e – это массив ячеек, удовлетворяющий следующим условиям:

* вход не выходит за границы кластеров:
* симметрия:
* вход не содержит непроходимых ячеек
* вход является максимально возможным

Результат работы этого процесса изображен на рисунке 8. Границы входов представлены синими кругами. Вход может быть представлен одной или двумя точками, в зависимости от длины отрезка.

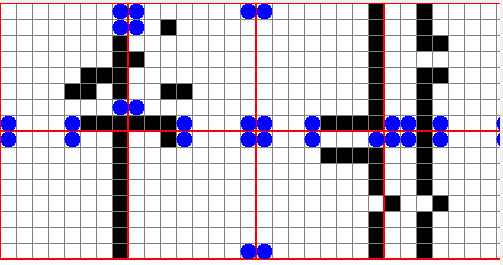


Рисунок 8. Границы входов

Далее строится взвешенный граф, узлами которого являются границы входов. Все грани узлов, представляющих симметричные входы, всегда имеют вес, равный 1. Для граней, соединяющих узлы одного кластера, веса вычисляются с помощью нахождения длины кратчайшего пути между ними. Эти пути можно сохранить для дальнейшего использования, однако в таком случае потребуется значительно больший объем памяти. Остальные узлы между собой не соединяются. Результат этого этапа можно увидеть на рис. 9. Зелеными линиями представлены грани полученного графа. Черными числами показаны веса граней.

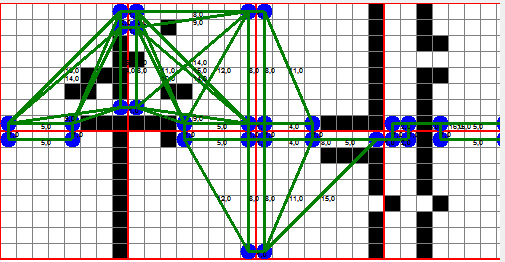


Рисунок 9. Построение графа

После того, как граф был построен и были посчитаны веса связей внутри кластеров, электронная карта готова к использованию алгоритмом HPA\*. Эта информация может быть вычислена заранее, сохранена вместе с файлами геометрии карты и загружена при запуске программы. Для динамически меняющихся карт, заранее полученная информация должна модифицироваться.

Далее проводится поиск пути по полученному графу. Первым этапом этого процесса является добавление стартового узла к границе кластера, который его содержит. Аналогичное действие производится с конечным узлом и конечным кластером. После добавления узлов ищется кратчайший путь по графу с использованием алгоритма A\*. Результатом работы алгоритма является последовательность узлов графа, составленного из проходов между кластерами. Последним этапом является восстановление кратчайших путей между узлами графа с использованием A\* внутри кластеров.

HPA\* позволяет добавлять новые уровни абстракции, объединяя кластеры в более крупные группы. Таким образом алгоритм становится масштабируемым на все более крупные электронные карты местности.

За счет значительного сокращения пространства поиска, алгоритм показывает временную эффективность, до 10 раз превосходящую A\*. На рисунке 10 приведены результаты сравнения алгоритмов A\* и HPA\*, проведенного исследователями University of Alberta. По оси абсцисс отложена длина кратчайшего пути, а по оси ординат – количество рассмотренных алгоритмами узлов и время выполнения [[6](6.%09http:/webdocs.cs.ualberta.ca/~kulchits/Jonathan_Testing/publications/ai_publications/jogd.pdf)].

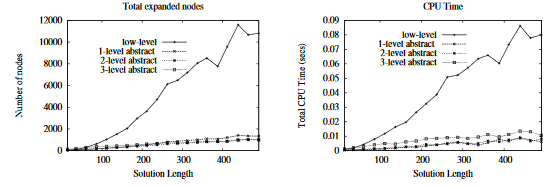


Рисунок 10. Сравнение HPA\* и A\*

Конструкторский раздел

Технология отображения картографических данных

Для реализации отображения электронной карты местности необходимо использовать специальный набор программных средств, который называется ГИС. ГИС (географическая информационная система) - Информационная система, обеспечивающая сбор, хранение, обработку, доступ, отображение и распространение пространственно-координированных данных (пространственных данных). ГИС содержит данные о пространственных объектах в форме их цифровых представлений (векторных, растровых, и иных) [[7](http://www.gisa.ru/13058.html)]. Включение такой системы в программный комплекс позволит работать с электронными картами местности, которые представляю собой модель реальной местности.

Формат картографических данных

В картографических системах данные чаще всего представляются в векторной форме. Это связано с тем, что при схематическом изображении пространства реальный внешний вид объекта не играет никакой роли, а значит можно хранить информацию только о его очертаниях. Также электронная карта должна быть легко масштабируема. Это условие накладывает ряд серьезных ограничений на растровую графику. В связи с этим растровые карты (например, снимки со спутника) должны иметь очень высокое разрешение, или набор слоев разного разрешения под разный масштаб. В связи с тем, что конечный продукт будет выполняться на персональных компьютерах, требования к производительности рендеринга достаточно высокие. Следовательно, предпочтительным является векторный формат карт.

Популярным форматом векторных карт является формат Shapefile. Этот формат описывает векторные элементы: точки, ломаные линии, многоугольники, которые в свою очередь могут представлять водоемы, здания, дороги и прочие объекты реального мира. Также Shapefile предоставляет возможность сопоставлять объектам различные значения (названия, температуру, ограничения скорости, количество населения и т.д.). Карты в формате Shapefile состоят из трёх обязательных файлов:

1) .shp – данные о геометрии объектов

2) .shx – данные индексации

3) .dbf – атрибуты объектов в формате dBase IV

Помимо этих файлов карте могут принадлежать несколько необзятельных дополнительных файлов:

1) .prj – описание проекции

2) .sbn и .sbx – пространственные индексы объектов

3) .fbn и .fbx - пространственные индексы объектов только для чтения

4) .ain и .aih – индексные файлы атрибутов таблиц

5) .ixs - индекс геокодирования наборов данных для чтения и записи

6) .mxs - индекс геокодирования наборов данных для чтения и записи (формат ODB)

7) .atx – индекс атрибута для .dbf файла

8) .shp.xml – метаданные в XML формате

9) .cpg – кодовая страница для определения кодировки символов .dbf файла

10) .qix - альтернативный пространственный индекс дерева квадрантов, используемый программами MapServer и GDAL / OGR

В файлах .shp, .shx и .dbf объекты в каждом файле соответствуют по позиции записи в файле, т.е. объекту из первой записи в .shp файле соответствуют атрибуты из первой запись в .dbf и т.д.

Рассмотрим формат векторной геометрии в shapefile. Двоичный файл .shp состоит из заголовка фиксированной длины, за которым следуют записи переменной длины. Каждая запись состоит из заголовка записи и содержимого. Упорядочивание двухмерных координат точек предполагает декартову систему координат с использованием порядка (X, Y) или (Easting Northing). Этот порядок согласуется с географическими координатами, в которых порядок аналогичен (долгота широта). Геометрические объекты также могут поддерживать трёхмерные и четырехмерные координаты.

Заголовок файла занимает 100 Байт и содержит 17 полей:

1) код– всегда равен 0x0000270a

2) длина файла в 16-битных словах

3) версия

4) типы геометрических объектов

5) минимальная прямоугольная граница

6) диапазон Z-координат

7) диапазон M-координат

Заголовок записи занимает 8 байт:

1) номер записи

2) длина записи в 16-битных словах

Содержимое записи:

1) тип геометрического объекта

2) список точек объекта

Shapefile поддерживает следующие основные геометрические примитивы:

1) точка – точка с координатами X, Y

2) поли-линия – ломана линия, состоящая из набора отрезков

3) многоугольник

4) мульти-точка – объект, состоящий из нескольких точек

5) трехмерная точка – точка с координатами X, Y, Z

6) трехмерная поли-линия

7) трехмерный многоугольник

8) трехмерная мульти-точка

Существует ряд других поддерживаемых примитивов, но они не представляют интереса для данного проекта.

Индексный файл .shx содержит такой же 100-байтовый заголовок, что и .shp файл. Содержимое индексного файла состоит из последовательности 8-байтовых записей из двух полей:

1) Смещение записи в 16-битных словах

2) Длина зписи в 16-битных словах

Используя индексный файл, можно легко искать записи, двигаясь не по самому файлу с геометрией, а по набору записей фиксированного размера. Таким образом, поиск возможно проводить с конца файла.

Файл атрибутов .dbf содержит атрибуты объектов. Он использует формат dBase IV. Данные представляются в виде таблицы, где заголовки столбцов содержат имена атрибутов, а номер строки соответствует номеру записи объекта в .shp файле. [[8](https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf)]

Выбор платформы для отображения картографических данных

Наиболее удобной формой использования этого набора является подключаемая библиотека, поэтому были рассмотрены наиболее популярные ГИС-библиотеки. Требованием к библиотеке была поддержка платформы .NET Framework и языка программирования C#.

Выбор пал на библиотеку MapAround. Это платформа для решения прикладных задач обработки, хранения и отображения пространственных данных. Поддерживает формат Shapefile и построение интерфейса WindowsForms [[9](http://www.maparound.ru/)]. Платформа и ее исходный код распространяются под лицензией, совместимой с GNU GPL License V3, что позволяет использовать ее в приложениях с открытым исходным кодом. [[10](http://www.maparound.ru/Download.aspx)]

Применение алгоритмов поиска кратчайшего пути к электронной карте местности

Список литературы

1. Prepared Statement of Capt. Walter M. Locke, U.S. Navy, Tomahawk Cruise Missile Project Manager. Hearings before the Subcommittee on Research and Development of the Committee on Armed Services, United States Senate, March 31, 1977. — Washington, D.C.: U.S. Government Printing Office, 1977. — 6454 p.
2. Engineering Route Planning Algorithms. Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Universit ̈at Karlsruhe (TH), 76128 Karlsruhe, Germany
3. <http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf>
4. Zarembo, Imants & Kodors, Sergejs. (2015). Pathfinding Algorithm Efficiency Analysis in 2D Grid. Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference. 2. 46. 10.17770/etr2013vol2.868. Страница 47
5. *Pearl J.* Heuristics: Intelligent Search Strategies for Computer Problem Solving. — Addison-Wesley, 1984. — С. 48.
6. http://webdocs.cs.ualberta.ca/~kulchits/Jonathan\_Testing/publications/ai\_publications/jogd.pdf
7. <http://www.gisa.ru/13058.html>
8. <https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>
9. http://www.maparound.ru
10. http://www.maparound.ru/Download.aspx