Задача коммивояжёра

Задача коммивояжёра (англ. *Travelling salesman problem*, сокращённо *TSP*) — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов

Точно неизвестно, когда проблему коммивояжера исследовали впервые. Однако, известна изданная в 1832 году книга с названием «Коммивояжёр — как он должен вести себя и что должен делать для того, чтобы доставлять товар и иметь успех в своих делах — советы старого курьера», в которой описана проблема, но математический аппарат для её решения не применяется. Зато в ней предложены примеры маршрутов для некоторых регионов Германии и Швейцарии.

Ранним вариантом задачи может рассматриваться англ. *Icosian Game* Уильяма Гамильтона 19 века. Первые упоминания в качестве математической задачи на

оптимизацию принадлежат Карлу Менгеру (нем. *Karl Menger*), который сформулировал её на математическом коллоквиуме в 1930 году так:

> *Мы называем проблемой посыльного (поскольку этот вопрос возникает у каждого почтальона, в частности, её решают многие путешественники) задачу найти кратчайший путь между конечным множеством мест, расстояние между которыми известно.*

Вскоре появилось известное сейчас название *задача странствующего торговца* , которую предложил Хасслер Уитни (англ. *Hassler Whitney*) из Принстонского университета.

Вместе с простотой определения и сравнительной простотой нахождения хороших решений задача коммивояжёра отличается тем, что нахождение действительно оптимального пути является достаточно сложной задачей. Учитывая эти свойства, начиная со второй половины XX века исследование задачи коммивояжёра имеет не столько практический смысл, сколько теоретический в качестве модели для разработки новых алгоритмов оптимизации.

Постановка задачи

В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро. В тех случаях, когда между отдельными городами не существует сообщения, этого можно достичь путём ввода рёбер с максимальной длиной. Из-за большой длины такое ребро никогда не попадет к оптимальному маршруту, если он существует.

Таким образом, решение задачи коммивояжёра — это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

В зависимости от того, какой критерий выгодности маршрута сопоставляется величине ребер, различают различные варианты задачи, важнейшими из которых являются симметричная и метрическая задачи.

Асимметричная и симметричная задачи

В общем случае, асимметричная задача коммивояжёра отличается тем, что она моделируется ориентированным графом. Таким образом, следует также учитывать, в каком направлении находятся ребра.

В случае симметричной задачи все пары ребер между одними и теми же вершинами имеют одинаковую длину. Симметричная задача моделируется неориентированным графом. В симметричном случае количество возможных маршрутов вдвое меньше асимметричного случая.

Метрическая задача

Симметричную задачу коммивояжёра называют *метрической*, если относительно длин ребер выполняется неравенство треугольника. Условно говоря, в таких задачах обходные пути длиннее прямых, то есть, ребро от вершины i до вершины j никогда не бывает длиннее пути через промежуточную вершину k:

```
c[i][j] <= c[i][k] + c[k][j]
```

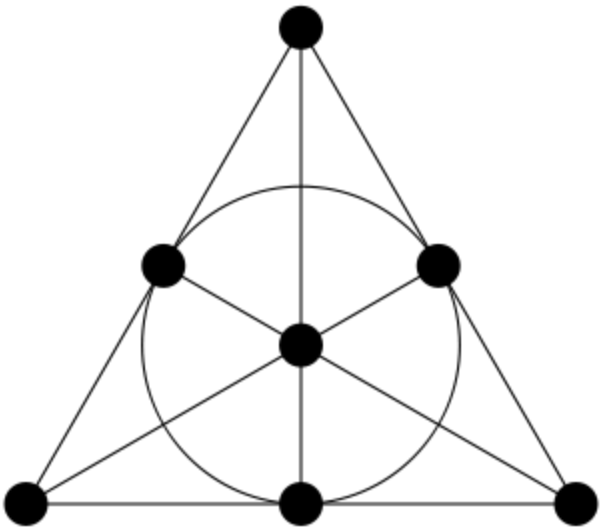
Такое свойство длины ребер определяет измеримое пространство на множестве ребер и меру расстояния, удовлетворяющую интуитивному пониманию расстояния.

Алгоритмы решения

- Полный перебор
- Жадные алгоритмы
- Метод минимального остовного дерева
- Метод ветвей и границ
- Муравьиный алгоритм
- Генетический алгоритм

Жадный алгоритм

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум.



Принцип жадного выбора

Говорят, что к оптимизационной задаче применим **принцип жадного выбора**, если последовательность локально оптимальных выборов даёт глобально оптимальное решение. В типичном случае доказательство оптимальности следует такой схеме:

1. Доказывается, что жадный выбор на первом шаге не закрывает пути к оптимальному решению: для всякого решения есть другое, согласованное с жадным выбором и не хуже первого.
2. Показывается, что подзадача, возникающая после жадного выбора на первом шаге, аналогична исходной.
3. Рассуждение завершается по индукции.

Размен монеток

Задача. Монетная система некоторого государства состоит из монет достоинством a = 1 < a2 <...< a(n). Требуется выдать сумму S наименьшим возможным количеством монет.

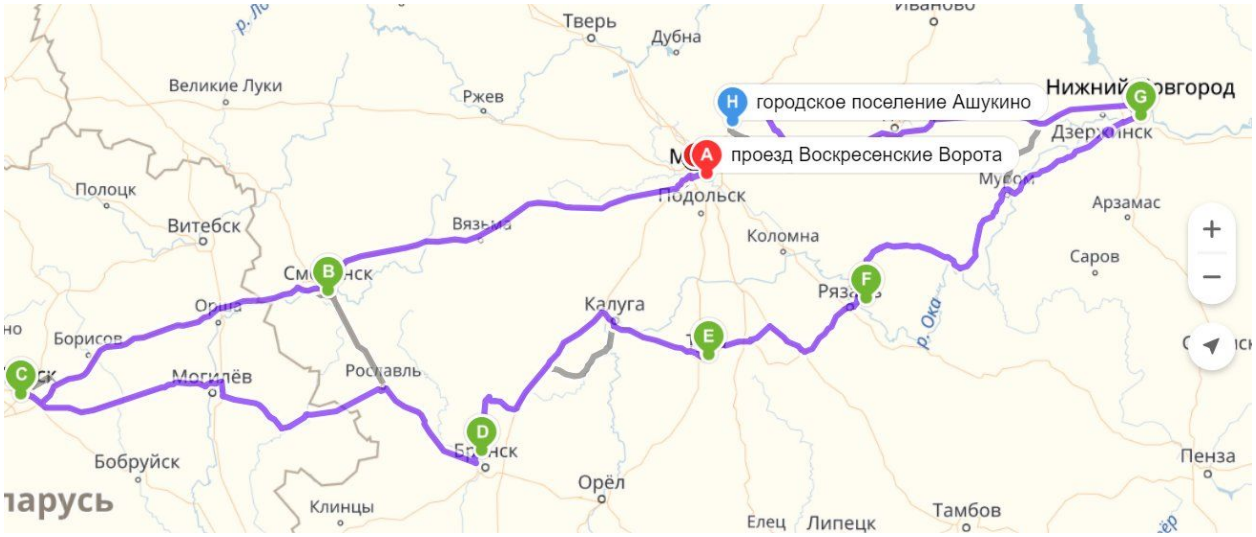
Жадный алгоритм решения этой задачи таков. Берётся наибольшее возможное количество монет достоинства a(n): x(n}= S/a(n). Таким же образом получаем, сколько нужно монет меньшего номинала, и т. д.

Для данной задачи жадный алгоритм не всегда даёт оптимальное решение, а только для некоторых, называемых *каноническими*, монетных систем, вроде используемых в США (1, 5,

10, 25 центов). Неканонические системы таким свойством не обладают. Так, например, сумму в 24 копейки монетами в 1, 5 и 7 коп. жадный алгоритм разменивает так: 7 коп. — 3 шт., 1 коп. — 3 шт., в то время как правильное решение — 7 коп. — 2 шт., 5 коп. — 2 шт.

Метод ближайшего соседа

> Пункты обхода плана последовательно включаются в маршрут, причем каждый очередной включаемый пункт должен быть ближайшим к последнему выбранному пункту среди всех остальных, ещё не включенных в состав маршрута.



Псевдокод

```
DSF(v)
посетили[v] = истина;
финиш = НетНеПосещенныхВершин();
путь.Добавить(v);
для u из v в {порядке возрастания весов ребер}
                если финиш то
                        если u == старт то
                                вернуть истина
```

иначе
если не посетили[u] **то**
если DFS(u) **то**
вернуть истина
посетили[v] = ложь
путь.УдалитьПоследний()
вернуть ложь

Метод ветвей и границ

Метод ветвей и границ (англ. *branch and bound*) — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Общая идея метода может быть описана на примере поиска минимума функции f(x) на множестве допустимых значений переменной x. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура *ветвления* состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое *деревом поиска* или *деревом ветвей и границ*. *Узлами* этого дерева являются построенные подобласти (подмножества множества значений переменной x.

В основе метода лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения

(*правило отсева*). Любой узел дерева поиска, нижняя граница которого больше значения **m**, может быть исключён из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм (последовательность действий):

1) Построение матрицы с исходными данными.
2) Нахождение минимума по строкам.
3) Редукция строк.
4) Нахождение минимума по столбцам.
5) Редукция столбцов.
6) Вычисление оценок нулевых клеток.
7) Редукция матрицы.
8) Если полный путь еще не найден, переходим к пункту 2, если найден к пункту 9.
9) Вычисление итоговой длины пути и построение маршрута. Более подробно эти

Итак, методика решения задачи коммивояжера:

1. Построение матрицы с исходными данными Сначала необходимо длины дорог соединяющих города представить в виде следующей таблицы:

Город	1	2	3	4
1	M	5	11	9
2	10	M	8	7
3	7	14	M	8
4	12	6	15	M

В нашем примере у нас 4 города и в таблице указано расстояние от каждого города к 3-м другим, в зависимости от направления движения (т.к. некоторые ж/д пути могут быть с односторонним движением и т.д.). Расстояние от города к этому же городу обозначено буквой М. Также используется знак бесконечности. Это сделано для того, чтобы данный отрезок путь был условно принят за бесконечно длинный. Тогда не будет смысла выбрать движение от 1-ого города к 1-му, от 2-ого ко 2-му, и т.п. в качестве отрезка маршрута.

2. Нахождение минимума по строкам Находим минимальное значение в каждой строке (di) и выписываем его в отдельный столбец.

Город	1	2	3	4	di
1	M	5	11	9	5
2	10	M	8	7	7
3	7	14	M	8	7
4	12	6	15	M	6

3. Редукция строк Производим редукцию строк – из каждого элемента в строке вычитаем соответствующее значение найденного минимума (di). В итоге в каждой строке будет хотя бы одна нулевая клетка.

Город	1	2	3	4	di
1	M	0	6	4	5
2	3	M	1	0	7
3	0	7	M	1	7
4	6	0	9	M	6

4. Нахождение минимума по столбцам Далее находим минимальные значения в каждом столбце (dj). Эти минимумы выписываем в отдельную строку.

Город	1	2	3	4	di
1	M	0	6	4	5
2	3	M	1	0	7
3	0	7	M	1	7
4	6	0	9	M	6
dj	0	0	1	0	

5. Редукция столбцов Вычитаем из каждого элемента матрицы соответствующее ему dj. В итоге в каждом столбце будет хотя бы одна нулевая клетка.

Город	1	2	3	4	di
1	M	0	5	4	5
2	3	M	0	0	7
3	0	7	M	1	7
4	6	0	8	M	6
dj	0	0	1	0	

6. Вычисление оценок нулевых клеток Для каждой нулевой клетки получившейся преобразованной матрицы находим «оценку». Ею будет сумма минимального элемента по строке и минимального элемента по столбцу, в которых размещена данная нулевая клетка. Сама она при этом не учитывается. Найденные ранее di и dj не учитываются. Полученную оценку записываем рядом с нулем, в скобках. И так по всем нулевым клеткам:

Город	1	2	3	4
1	M	0 (4)	5	4
2	3	M	0	0
3	0	7	M	1
4	6	0	8	M

7. Редукция матрицы Выбираем нулевую клетку с наибольшей оценкой. Заменяем ее на «М». Мы нашли один из отрезков пути. Выписываем его (от какого города к какому движемся, в нашем примере от 4-ого к 2-му). Ту строку и тот столбец, где

образовалось две «М» полностью вычеркиваем. В клетку, соответствующую обратному пути, ставим еще одну букву «М» (т.к. мы уже не будем возвращаться обратно).

Город	1	2	3	4
1	M	0 (4)	5	4
2	3	M	0 (5)	0 (1)
3	0 (4)	7	M	1
4	6	0 (6)	8	M

8. Если полный путь еще не найден, переходим к пункту 2, если найден к пункту 9 Если мы еще не нашли все отрезки пути, то возвращаемся ко 2-му пункту и вновь ищем минимумы по строкам и столбцам, проводим их редукцию, считаем оценки нулевых клеток и т.д. Если все отрезки пути найдены (или найдены еще не все отрезки, но оставшаяся часть пути очевидна) – переходим к пункту 9

Город	1	2	3	4
1	M	0 (4)	5	4
2	3	M	0 (5)	0 (1)
3	0 (4)	7	M	1
4	6	0 (6)	8	M

. 9. Вычисление итоговой длины пути и построение маршрута Найдя все отрезки пути, остается только соединить их между собой и рассчитать общую длину пути (стоимость поездки по этому маршруту, затраченное время и т.д.). Длины дорог соединяющих города берем из самой первой таблицы с исходными данными.

Город	1	2	3	4
1	M	0 (4)	5	4
2	3	M	0 (5)	M
3	0 (4)	7	M	1
4	6	M	8	M

В нашем примере маршрут получился следующий: 4 → 2 → 3 → 1 → 4. Общая длина пути: L = 30.

Муравьиный алгоритм

Муравьиный алгоритм (*алгоритм оптимизации подражанием муравьиной колонии*, англ. *ant colony optimization*, ACO) — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания и представляет собой метаэвристическую оптимизацию. Первая версия алгоритма, предложенная доктором наук Марко Дориго в 1992 году, была направлена на поиск оптимального пути в графе.

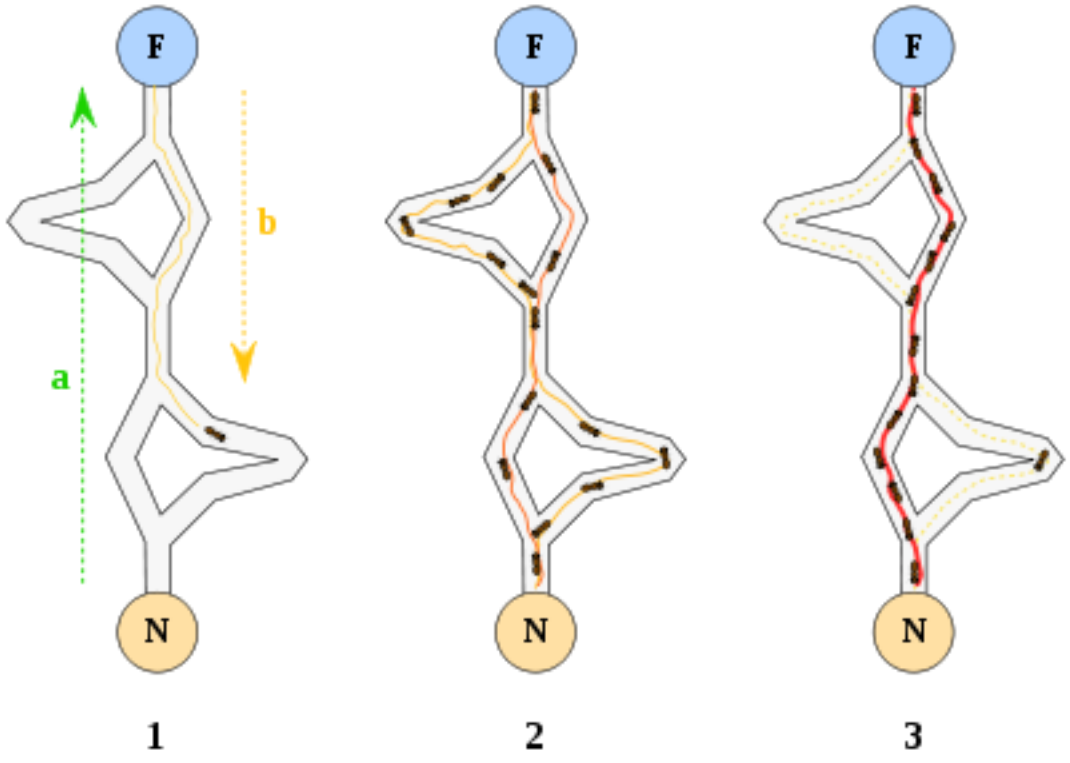
Краткое изложение

В реальном мире муравьи (первоначально) ходят в случайном порядке и по нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания.

Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым и как следствие, плотность феромонов остаётся высокой.

Испарение феромонов также имеет свойство избежания стремления к локально-оптимальному решению. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.

Пример



Оригинальная идея исходит от наблюдения за муравьями в процессе поиска кратчайшего пути от колонии до источника питания.

1. Первый муравей находит источник пищи (F) любым способом (a), а затем возвращается к гнезду (N), оставив за собой тропу из феромонов (b).
2. Затем муравьи выбирают один из четырёх возможных путей, затем укрепляют его и делают привлекательным.

3. Муравьи выбирают кратчайший маршрут, так как феромоны с более длинных путей быстрее испаряются.

Среди экспериментов по выбору между двумя путями неравной длины, ведущих от колонии к источнику питания, биологи заметили, что, как правило, муравьи используют кратчайший маршрут..

Рёбра:

Муравей будет двигаться от узла i к узлу j с вероятностью:

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}$$, где:

$\tau_{i,j}$ — это количество феромонов на ребре i, j;

α — параметр, контролирующий влияние $\tau_{i,j}$;

$\eta_{i,j}$ привлекательность ребра i, j (начальное значение, как правило, $1/d_{i,j}$, где d расстояние);

β — параметр, контролирующий влияние $\eta_{i,j}$.

Обновление феромонов

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j},$$

где:

$\tau_{i,j}$ — количество феромона на дуге i, j,

ρ — скорость испарения феромона,

$\Delta\tau_{i,j}$ — количество отложенного феромона, обычно определяется как:

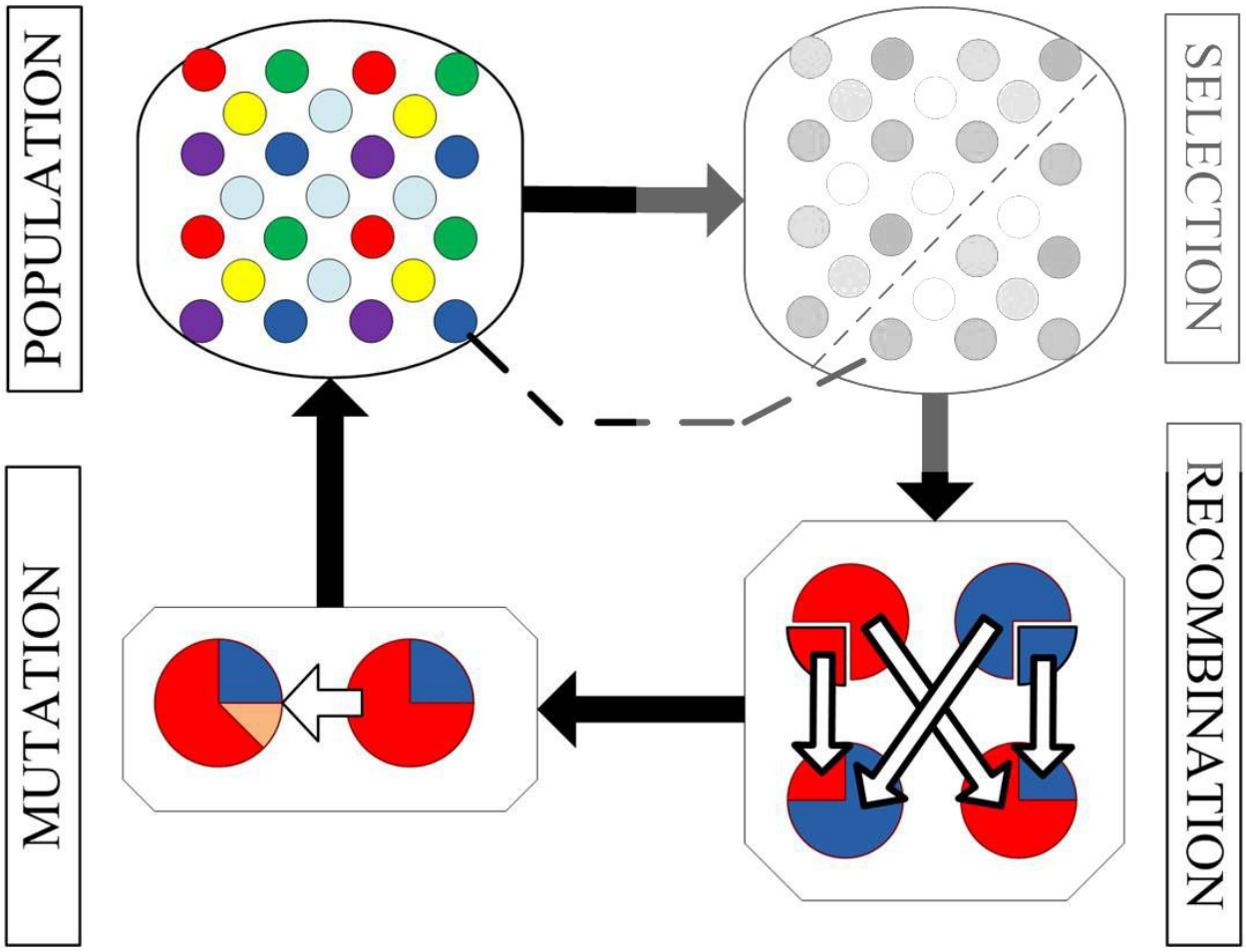
$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k & \text{if ant } k \text{ travels on edge } i,j \\ 0 & \text{otherwise} \end{cases},$$

где L_k — стоимость k-го пути муравья (обычно длина).

Генетический алгоритм

Генети́ческий алгори́тм (англ. *genetic algorithm*) — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием

механизмов, аналогичных естественному отбору в природе. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.



Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задать целевую функцию (приспособленности) для особей популяции
2. Создать начальную популяцию

- (Начало цикла)
    1. Размножение (скрещивание)
    2. Мутирование
    3. Вычислить значение целевой функции для всех особей
    4. Формирование нового поколения (селекция)

5. Если выполняются условия остановки, то (конец цикла), иначе (начало цикла).

Модель популяции

Популяция – это список маршрутов.

В популяцию можно добавлять дочерний маршрут, или заменять им, плохой (слишком длинный) маршрут популяции. Можно сделать процедуру чистки популяции, удаляющую время от времени из популяции плохие маршруты и т. д.

Перед созданием начальной популяции для каждого города определяется список городов-соседей, то есть городов, наиболее близко расположенных к текущему городу. Города в списке упорядочены по возрастанию расстояний городов до текущего города.

Начальная популяция формируется следующим образом:

1. Получить случайное число из диапазона [0, количествоГородов – 1] и интерпретировать его как номер текущего города
2. Получить случайное число из диапазона [0, 99], и если оно меньше константно заданной вероятности, то получить следующий город маршрута по принципу жадного алгоритма, то есть взять его из списка городов-соседей (берется первый свободный город списка). Если свободных, то есть не включенных в маршрут городов в списке соседей нет, то случайным образом берется свободный город из общего списка городов.
   В противном случае в маршрут добавляется свободный город, случайно выбранный из общего списка городов.
3. Передвинуть, сохраняя последовательность, в каждом маршруте города так, чтобы на первом месте оказался ранее заданный город - стартовая точка.
4. Добавить в конец каждого маршрута его первый город - стартовая точка.

Сформированная по приведенной схеме модель популяции позволяет довольно просто реализовать операции скрещивания и мутации.

Общая идея алгоритма:

1. Сформировать начальную популяцию – список маршрутов, создаваемых на основе жадного алгоритма с некой наперед заданной вероятностью
2. Рассчитать для каждого маршрута его длину.
3. generation = 0.

4. Отобрать случайным образом из популяции population несколько маршрутов и поместить их в рабочую группу.
5. Упорядочить рабочую группу по возрастанию длины маршрута (например, пузырьковая сортировка).
6. Взять два первых (лучших) маршрута массива рабочей группы в качестве родительских и сформировать, используя скрещивание и мутацию, дочерний маршрут (вероятность мутации задается настроечным параметром).
7. Заменить в популяции худший маршрут рабочей группы на маршрут потомка.
8. Вычислить длину маршрута потомка и если она меньше длины лучшего маршрута, то признать потомка в качестве лучшего маршрута.
9. generation = generation + 1.
10. Если generation < maxGenerations, то перейти к п. 7 алгоритма, иначе завершить вычисления.

Вопросы?

Имеется много скептиков относительно целесообразности применения генетических алгоритмов. Например, Стивен С. Скиена, профессор кафедры вычислительной техники университета Стоуни—Брук, известный исследователь алгоритмов, лауреат премии института IEEE, пишет:

> *Я лично никогда не сталкивался ни с одной задачей, для решения которой генетические алгоритмы оказались бы самым подходящим средством. Более того, я никогда не встречал никаких результатов вычислений, полученных посредством генетических алгоритмов, которые производили бы на меня положительное впечатление.*

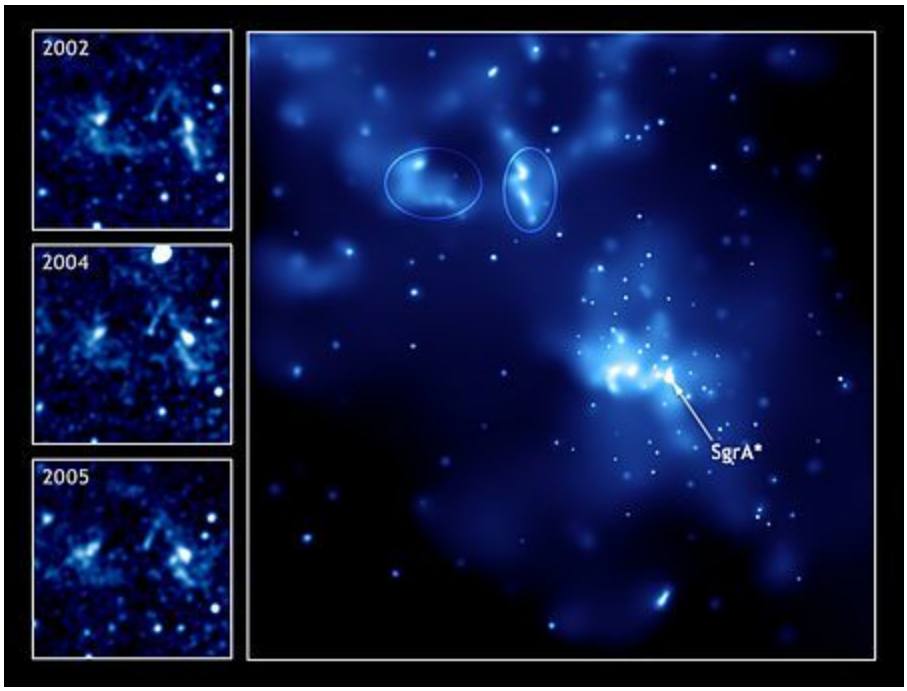
Алгоритм A*

Поиск A* (произносится «А звезда» или «А стар», от англ. *A star*) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Этот алгоритм был впервые описан в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем. Это по сути было расширение алгоритма Дейкстры, созданного в 1959 году. Новый алгоритм достигал более высокой производительности (по времени) с

помощью эвристики. В их работе он упоминается как «алгоритм A». Но так как он вычисляет лучший маршрут для заданной эвристики, он был назван A*.

Стрелец A* (лат. *Sagittarius A*, Sgr A**; произносится «Стрелец А со звёздочкой») — компактный радиоисточник, находящийся в центре Млечного Пути, входит в состав радиоисточника Стрелец А. Представляет собой высокоплотный объект, вероятно сверхмассивную чёрную дыру, окружённую горячим радиоизлучающим газовым облаком диаметром около 1,8 пк. Расстояние до радиоисточника составляет около 26 тыс. св. лет, масса центрального объекта — $4{,}31 \cdot 10^6$ $M_\odot$



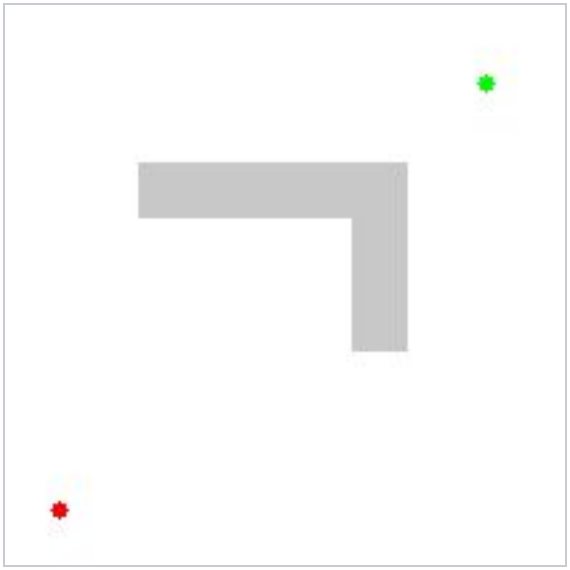
Идея алгоритма

Относится к классу информированных алгоритмов поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. Основа алгоритма - **эвристическая функци**я «расстояние + стоимость». Эта функция — сумма стоимости достижения рассматриваемой вершины (*A*) из начальной (стоимость уже пройденного пути), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной.

По сути, это жадный алгоритм, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе следующей вершины он учитывает, помимо прочего, *весь* пройденный до неё путь, а не только минимальное следующее ребро.

A* пошагово просматривает пути, ведущие от начальной вершины в конечную, выбирая на каждом шаге наилучший вариант с точки зрения эвристической функции .

На каждом шаге алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.



Пустые кружки в узлах принадлежат *открытому списку*, красные/зелёные относятся к *закрытому списку*

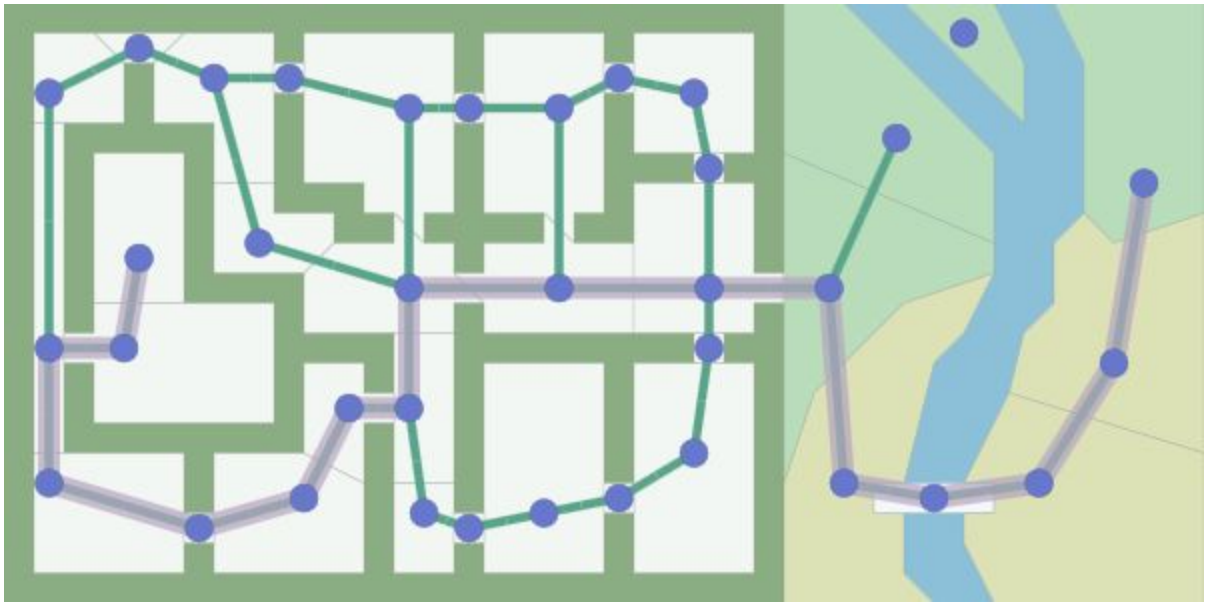
Псевдокод

```
A*(A, B):
     маршруты.Добавить(0,новый Маршрут(пусто, A))
     пока маршруты не пусто
         маршрут = маршруты.Минимальный()
         v = маршрут.вершина
         если v == B то
            вернуть маршрут
         посещали[v] = истина
```

```
        для u из v
            если не посещали[u] то
                нов = новый Маршрут(маршрут, u, вес(v,u))
                маршруты.добавить(маршрут.Эвристика(), маршрут)
    вернуть ложь

класс Маршрут {
    Маршрут родитель
    Вершина вершина
    double дистанция
    double оценка

    Маршрут(маршрут, точка):
        родитель = маршрут
        вершина = точка
        если родитель пусто то
            дистанция = 0
        иначе
            дистанция = родитель.дистанция+вес
        оценка = Расстояние(точка, B)

    Эвристика()
        вернуть дистанция + Коэфф*оценку
}
```

Обработка тупиков?

Есть идеи как улучшить?

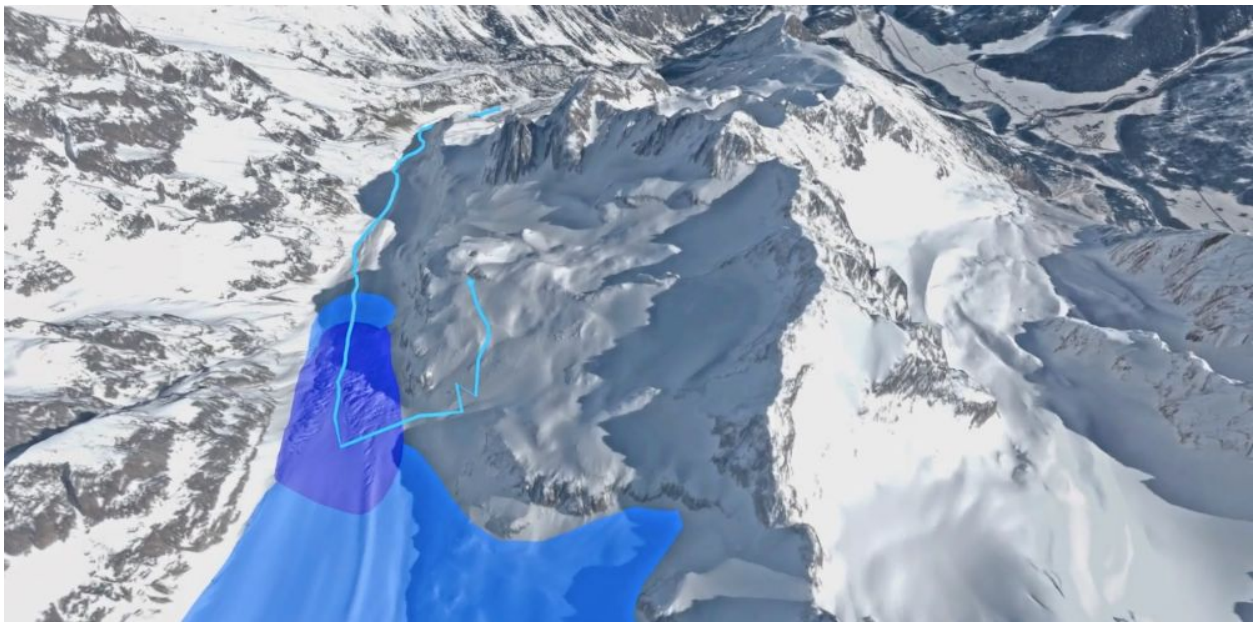


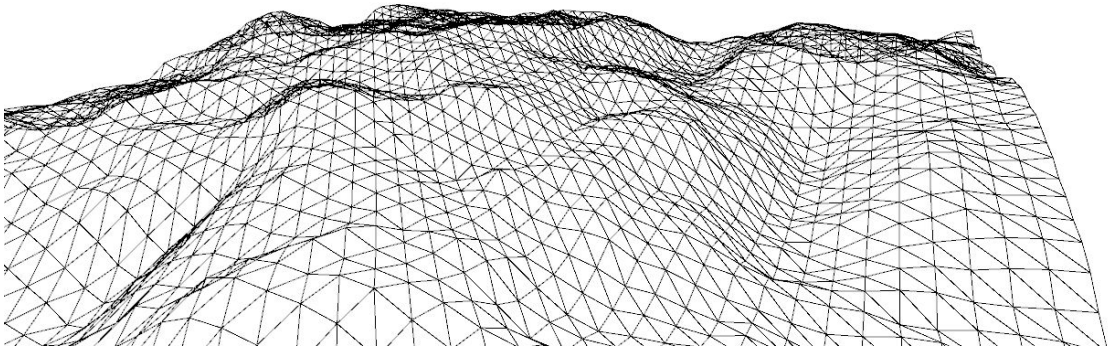
```
A*(A, B):
      маршруты.Добавить(0,новый Маршрут(пусто, A))
      пока маршруты не пусто
          маршрут = маршруты.Минимальный()
          v = маршрут.вершина
          если v == B то
                вернуть маршрут
          посещали[v] = маршрут.дистанция
          тупик = истина
          для u из v
              были = посещали.Существует(u)
              если были и посещали[u] > 0 и
                   посещали[u] > маршрут.дистанция+вес(v,u) или
                   не были то
                  нов = новый Маршрут(маршрут, u, вес(v,u))
                  маршруты.добавить(маршрут.Эвристика(), маршрут)
                  посещали.Удалить(u)
                  тупик = ложь
          если тупик то
            посещали[v] = -1
      вернуть ложь
```

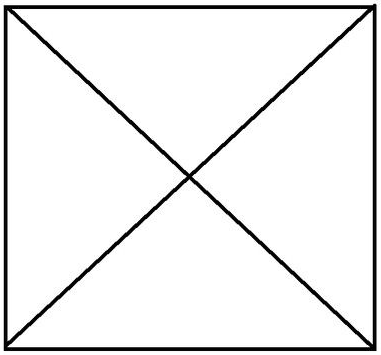
Вопросы ?

Алгоритмы в реальных проектах

Построение оптимальное линии спуска для фрирайдера по рельефу. Использовался алгоритм A*







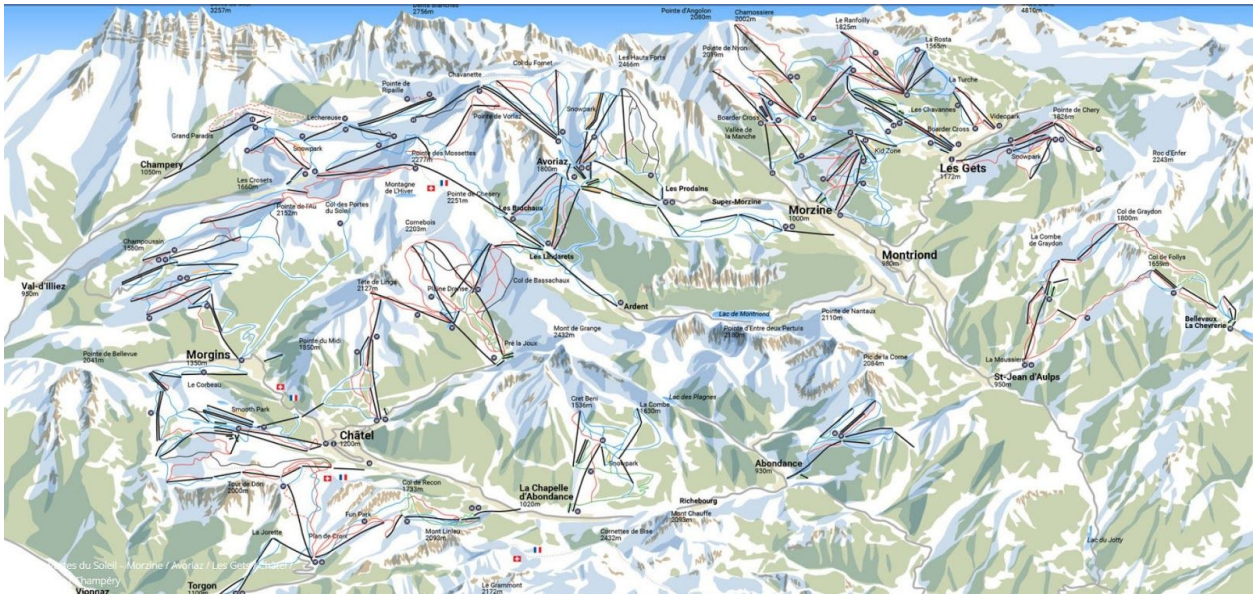
Выбор шага сетки

Использовалась референтная функция, подготавливающая данные. Измерялось время выполнения, и по пропорции вычислялся шаг

шаг = эталонный_шаг*замер / эталонный_замер
шаг += (эталонный_шаг-шаг)/2

Ограничения

1) Ехать только вниз, или по горизонтали
2) Езда вверх имела очень высокий коэффициент в эвристике
3) Уклон меньше заданного в настройках



Навигация по горнолыжному курорту

1) Кратчайший маршрут точка-точка
2) Аналог задачи коммивояжера Ски Сафари- совершить круговой маршрут с возвратом в стартовую точку, потратив на это заданное время, с учетом условий:
   - заданный уровень катания, определяет цвет трасс
   - минимизация повторов трасс (учетом предыдущих дней)
   - заехать пообедать в заданное время
   - учет реальной скорости лыжника (по истории),
   - учет скорости подъемников,
   - учет очередей на подъемники
   - учет времени работы подъемников

Задача 1 решалась алгоритмом Дейкстры.
Реальный объем данных несколько тысяч ребер - считается мгновенно.

Задача 2. решалась комбинацией
- Поиск в глубину с ограничением
- Жадный алгоритм
- Эвристика для ребра e = e1 * e2 * e3 ...
- Эвристика для маршрута e = e(ребро1) + e(ребро2) +..

Алгоритм Флойда-Уоршалла

http://journal-s.org/index.php/vmno/article/view/5638

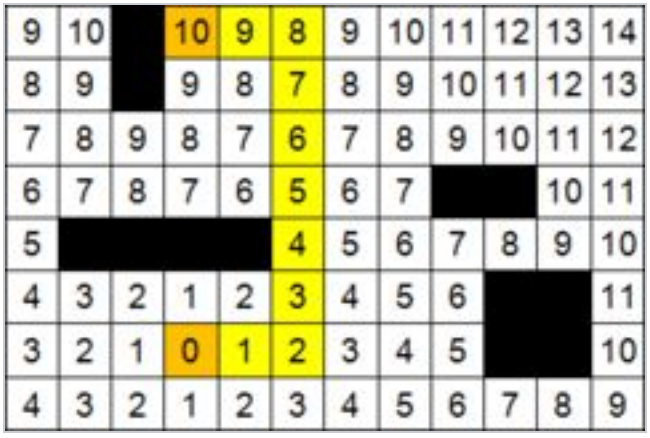
Применение алгоритма Флойда-Уоршелла для оптимизации работы персонала предприятий связи по обслуживанию сети

Алгоритм Ли

Алгори́тм волново́й трассиро́вки (**волновой алгоритм**, **алгоритм Ли**) — алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

В основном используется при компьютерной трассировке (разводке) печатных плат, соединительных проводников на поверхности микросхем. Другое применение волнового алгоритма — поиск кратчайшего расстояния на карте в компьютерных стратегических играх.

Волновой алгоритм в контексте поиска пути в лабиринте был предложен Э. Ф. Муром. Ли независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году.



Алгоритм работает на *дискретном рабочем поле* (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные (квадратные) ячейки. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник).

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: **инициализацию**, **распространение волны** и **восстановление пути**.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченная числом шагов от стартовой ячейки, становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько.

```
Ли(A, B)
F[A.x, A.y] = 0;
d = 0
цикл
     итерация = ложь
     для i=0 до N
          для j=0 до M
               если F[i,j] == d то
                    итерация = СделатьШаг(i,j)
     d++;
пока F[B.x,B.y] == 0 и итерация


СделатьШаг(x, y):
     если (x > 0 и F[x-1,y] == пустая_ячейка то
          F[x-1,y] = d
     если y > 0 и F[x,y-1] == пустая_ячейка то
          F[x,y-1] = d
     если x < N-1 и F[x+1,y] == пустая_ячейка то
          F[x-1,y] = d
     если y < M-1 и F[x,y+1] == пустая_ячейка то
          F[x,y+1] = d
```

Когда нужен `итерация = ложь ?`

Как будут обрабатываться препятствия?

Как не создавать все поле?

```
ВосстановлениеПути(A,B)
     если F[B.x, B.y] != пустая_ячейка то
          текущая = B
          цикл
             путь.добавить(текущая)
             текущая =
                НайтиСоседа(текущая, F[текущая.x, текущая.y]-1);
          пока текущая != A
          вернуть путь
     иначе
          вернуть пусто
```

Вопросы?

Алгоритм Джонсона

Алгоритм Джонсона — позволяет найти кратчайшие пути между всеми парами вершин взвешенного ориентированного графа. Данный алгоритм работает, если в графе содержатся рёбра с положительным или отрицательным весом, но отсутствуют циклы с отрицательным весом. Назван в честь Д. Б. Джонсона, опубликовавшего алгоритм в 1977 году.

Алгоритм Дейкстры очень удобен для получения всех кратчайших путей в графе, т.к. для n вершин он выполняется за: O(V E + V 2 lg V) – почти квадратичное время. Для его использования нужно предварительно избавиться от дуг отрицательного веса.

Теорема: используя функцию h : V → R, можно “перевзвесить” каждую дугу (u, v) ∈ E задав wh(u, v) = w(u, v) + h(u) − h(v). Тогда для любых двух вершин все пути между ними будут “перевзвешены” на одинаковое значение.

Пусть $p = v_1 \to v_2 \to \ldots \to v_k$ – путь в графе G. Тогда:

$$w_h(p) = \sum_{i=1}^{k-1} w_h(v_i, v_{i+1}) =$$

$$= \sum_{i=1}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) =$$

$$= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) =$$

$$= w(p) + \underbrace{h(v_1) - (v_k)}_{\text{константа} \geqslant 0}$$

Следствие:

$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$

Если найти функцию h : V → R, такую, что wh(u, v) > 0 для всех (u, v) ∈ E, то можно выполнить алгоритм Дейкстры для каждой вершины графа с измененными весами. Функция должна сделать веса всех дуг неотрицательными.
Можно заметить, что:

wh(u, v) > 0 ⇐⇒ h(v) − h(u) <= w(u, v)

Эта задача разностных ограничений (частный случай задачи линейного программирования), которую можно решить с помощью алгоритма Беллмана-Форда.

Схема алгоритма Джонсона:
1. С помощью алгоритма Беллмана-Форда найти функцию $h : V \to R$ такую, что $wh(u,v) > 0$ для всех $(u,v) \in E$. Это можно сделать, решив задачу разностных ограничений $h(v) - h(u) <= w(u,v)$. **Время: O(V E)**
2. Выполнить алгоритм Дейкстры, используя функцию wh для каждой вершины $u \in V$, чтобы вычислить $\delta h(u,v)$ для всех $v \in V$.**Время: O(V E + V 2 lgV)**
3. Для всех $(u,v) \in V \times V$ вычислить $\delta(u,v) = \delta h(u,v) - h(u) + h(v)$ **Время: O(V 2)**
Тогда общее время работы алгоритма Джонсона: **O(V E + V 2 lgV)** – практически квадратичное.
Алгоритм работает только для графов, в которых нет циклов отрицательного веса.