Оглавление

Введение

Основной целью ознакомительной практики 4-го семестра, входящей в учебный план подготовки бакалавров по направлению 01.03.04 — Прикладная математика, является знакомство с особенностями осуществления деятельности в рамках выбранного направления подготовки и получение навыков применения теоретических знаний в практической деятельности.

В рамках освоенного курса «Введение в информационные технологии» изучены основные возможности языка программирования C++ и сформированы базовые умения в области программирования на C++. Задачей практики является закрепление соответствующих знаний и умений и овладение навыками разработки программ на языке C++, реализующих заданные алгоритмы. На протяжении курса «Информационные технологии профессиональной деятельности» был получен опыт работы в системе компьютерной верстки TEX. Этот опыт оказался полезным при написании отчета по ознакомительной практике.

Формулировка индивидуального задания

Задача коммивояжера

Рассмотрим граф, т.е. набор вершин и ребер. Каждое ребро является характеризуется весом – положительным числом – стоимостью движения по нему, другими словами граф является взвешенным. Найдем такой обход графа, который будет включать ровно один раз каждую его вершину. Такой обход называется гамильтоновым циклом, а граф содержащий такой цикл - гамельтоновым графом. Задача коммивояжера заключается в том, чтобы найти гамильтоновый цикл минимальной стоимости, т.е. чтобы сумма весов ребер была наименьшей.

Необходимо выполнить следующее:

1. Решить задачу «полным перебором», находя все возможные гамельтоновы циклы.
2. Решить задачу с помощью алгоритма имитации отжига.

1. Метод полного перебора

1.1. Описание метода полного перебора

При реализации полного перебора рассматривались в качестве потенциальных путей все возможные перестановки из вершин. В начале выбирается такая перестановка, чтобы номера вершин в матрице смежности были в порядке возрастания. Например, если имеется 5 вершин, то такой перестановкой будет (1 2 3 4 5). Вычисляется ее вес, путем суммирования весов ребер. Далее перебираем все остальные перестановки, считаем для каждой вес, выбираем перестановку с наименьшим весом. Такая перестановка и будет решением.

1.2. Особенности реализации на языке C++

Граф представим как матрицу смежности `std::vector<vector<double>> matrix`. Для получения следующей перестановки использовалась функция `std::next_permutation(iterator.begin(), iterator.end())` [1]. Данная функция возвращает следующую перестановку, согласно лексикографическому порядку, и `false`, если такой не нашлось. Соответственно, пока данная функция не вернет `false`, считаем вес перестановки и сравниваем его с наименьшим из ранее полученных. Таким образом, мы переберем все гамльтоновы циклы и выберем из них тот, который имеет наименьший вес.

Временная сложность алгоритма $o(n!)$, так как имеем $n!$ перестановок и функция `std::next_permutation` имеет константную сложность.

1.3. Особенности реализации в Wolfram Mathematica

Для получения всех перестановок использовалась функция `Permutations[]`. Данная функция принимает на вход начальную перестановку, а возвращает все возможные перестановки. Далее действуем согласно общему алгоритму.

2. Алгоритм имитации отжига

2.1. Описание алгоритма имитации отжига

Алгоритм имитации отжига (*англ.* Simulated annealing) основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую, и что процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте [3].

Перейдем к описанию самого алгоритма. Выберем перестановку, состоящую из номеров вершин в матрице смежности, расставленных в произвольном порядке. Например, если имеется 5 вершин, то такой перестановкой может быть (3 5 1 2 4). Зададим константы `TempMax`, `TempMin`, `CycleRate`, `CoolingRate`. Они обозначают начальную и предельно допустимую температуру, максимальное количество итераций цикла и коэффициент понижения температуры. Далее рассмотрим цикл, в котором мы будем получать следующую перестановку, считать ее вес, проверять является ли она минимальной, уменьшать температуру. Условием выхода из цикла будет превышение количество итераций или температура ниже допустимой. Теперь про каждый этап подробнее:

- Выбираем следующий гамельтоновый цикл путем инвертирования пути между случайными двумя городами. Например имеем перестановку (3 5 1 2 4), двумя случайными числами будут 5 и 4. Тогда получаем (3 5 2 1 5).
- Вес перестановки считаем - это сумма весов ребер, как и в прошлом методе.

- Условие минимальности перестановки - наименьший вес. Если вес гамельтоного цикла больше или равен весу текущего минимального цикла, то необходимо вычислить вероятность по формуле

$$P = e^{\left(-\frac{\Delta E}{T}\right)}, \tag{1}$$

где ΔE является разницей длин нового и текущего маршрутов, а T — температура в данный момент времени. Далее мы сравниваем полученную вероятность со случайном числом на интервале $(0; 1)$, если вероятность больше или равна этому числу, то принимаем текущий цикл за минимальный. Вычисление погрешности было введено для того, чтобы снизить вероятность остановки перемещений при нахождении локальных минимумов.

- Температуру уменьшаем по формуле

$$T = T \cdot \frac{CoolingRate}{i},$$

где i — номер итерации.

Алгоритм имитации отжига не гарантирует нахождения минимума функции, однако позволяет значительно выиграть во времени исполнения, ведь ожидаемая временная сложность алгоритма $o(n^5)$ [5].

На рис. 1 приведена блок-схема данного алгоритма [4].

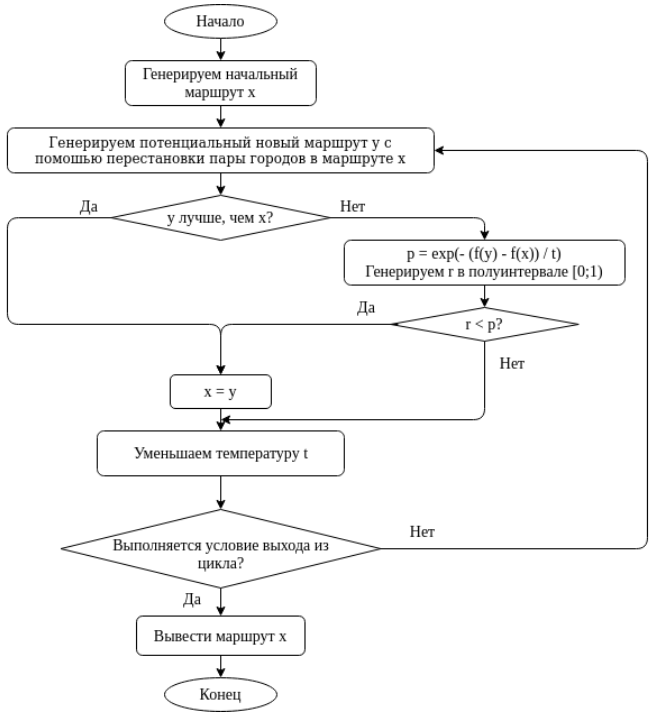

Рис. 1. Блок-схема алгоритма имитации отжига

2.2. Особенности реализации на языке C++

Граф представим как матрицу смежности `std::vector<vector<double>> matrix`. Начальную температуру зададим как максимальное значение `double`. Рассмотрим особенности реализации:

- Начальную перестановку получаем заполнением `std::vector<int> vertexes` числами от 0 включительно до размера матрицы смежности не включительно. Далее перемешаем элементы контейнера с помошью `std::shuffle(vertexes.begin(), vertexes.end(), hashFunction())`.
- `GetWeight(vertexes, matrix)` — это функция для подсчета суммарной длины маршрута. Суммирует вес ребер для перестановки vertexes и возвращает его.
- `FlipVertex(vertexes, firstN, secondN)` — данная функция инвертирует перестановку vertexes с элемента firstN по элемент secondN.
- `GetVertexes(vertexes, matrix)` — данная функция возвращает новый маршрут. Внутри нее вызывается функция `FlipVertex(vertexes, firstN, secondN)`, где $0 \leq$ `firstN` $<$ `secondN` $<$ `vertexes.size()` и эти числа являются случайными.
- `GetProbability()` — это функция, которая считает вероятность по формуле (1).

Для уменьшения вероятности ошибки алгоритм запускается несколько раз и качестве ответа выбирается перестановка с наименьшим весом.

3. Примеры решения модельных задач

Приведенные ниже примеры позволяют сравнить работу двух алгоритмов на тестах с различными входными данными.

В файле с входными данными в первой строке записано количество городов `n`, и следующие `n` строк занимает матрица смежности.

Пример 1. Рассматривается тест, состоящий из 4 городов, которые необходимо обойти. Это простейший случай задачи коммивояжера. На данном тесте можно проиллюстрировать формат входных и выходных данных и наглядно показать выбранный маршрут на графе.

```
Входные данные:
4
0 1 4 6
1 0 5 2
4 5 0 3
6 2 3 0
```

На рис. 2 изображен взвешенный граф. Синим цветом отмечен маршрут коммивояжера с минимальной длиной.

В таблице 1 приведено сравнение результатов работы программ, реализованных методом полного перебора и методом

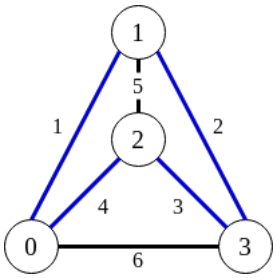

Рис. 2. Иллюстрация примера 1

имитации отжига. Можно заметить, что в обоих случаях минимальный маршрут определен правильно за почти одно и то же время.

Таблица 1. Пример работы алгоритмов для 4 городов

	Маршрут	Длина пути	Относительная погрешность, %	Затраченное время, с
Метод перебора	1 3 4 2 1	10	0,0	0,004
Метод имитации отжига при `coolingRate` = 0.1, `repeatRate` = 1	1 3 4 2 1	10	0,0	0,003

Пример 2. В тесте содержится 11 городов — это максимальное количество городов, которое может быть обработано методом полного перебора.

В этом примере становится видно как изменяется ответ и время исполнения метода отжига при изменении параметров `coolingRate` и `repeatRate`. В таблице 2 приведены средние значения длин найденных путей для разных значений параметра `coolingRate`, они были получены на основе 1000 измерений. Можно заметить, что при увеличении `coolingRate` при маленьком `repeatRate` относительная погрешность уменьшается, а при при увеличении `repeatRate` и уменьшении `coolingRate` наблюдается обратная тенденция. Кроме того, константа `coolingRate` почти не влияет на время исполнения программы, в отличии от `repeatRate`.

Также этот пример позволяет сравнить время, затраченное на решение задачи двумя разными методами. Для работы метода полного перебора требуется в несколько раз больше времени, чем для работы алгоритма имитации отжига.

Таблица 2. Пример работы алгоритмов для 11 городов

	Средняя длина полученного пути	Относительная погрешность, %	Затраченное время, с
Полный перебор	0,015	0	0,034
Алгоритм имитации отжига при `coolingRate` = 0.1, `repeatRate` = 1	0,034	132,1	0,004
Алгоритм имитации отжига при `coolingRate` = 0.1, `repeatRate` = 10	0,01508	0,5	0,012
Алгоритм имитации отжига при `coolingRate` = 0.9, `repeatRate` = 1	0,033	124,7	0,004
Алгоритм имитации отжига при `coolingRate` = 0.9, `repeatRate` = 3	0,020	35	0,006
Алгоритм имитации отжига при `coolingRate` = 0.9, `repeatRate` = 10	0,0151	1.06	0,012

Пример 3. Тест состоит из 15 точек. Аналитически был получен ответ 0.01701.

В таблице 3 приведено сравнение полученной длины пути, относительной погрешности и времени исполнения программы при разных значениях параметров `coolingRate` и `repeatRate`. Можно сделать вывод о том, что константа `coolingRate` очень слабо влияет на погрешность, однако при ее увеличении погрешность как правило снижается, а время, затраченное на исполнение программы совсем немного увеличивается. Кроме того, увеличение параметра `repeatRate` значительно улучшает точность результата, но в то же время заметно повышает время исполнения программы.

Таблица 3. Пример работы алгоритмов для 15 городов

`coolingRate`	`repeatRate`	Средняя длина полученного пути	Относительная погрешность, %	Затраченное время, с
0,1	10	0,04	135,3	0,018
0,1	20	0,026	52,9	0,045
0,1	50	0,019	8,9	0,076
0,1	100	0,01706	0,3	0,137
0,9	10	0,038	124,3	0,35
0,9	20	0,026	51,9	0,046
0,9	50	0,0181	6,4	0,074
0,9	100	0,01713	0,7	0,137
0,99	10	0,035	139,4	0,035
0,99	20	0,027	55,5	0,045
0,99	50	0,0184	8,3	0,075
0,99999	20	0,0255	50,4	0,044

Заключение

Результатом выполнения практического задания стало знакомство с задачей коммивояжера и изучением метода перебора и имитации отжига. Были написаны реализации этих методов на C++ и Wolfram Mathematica, а также выявлены их преимущества и недостатки.

Список использованных источников

1. next_permutation // cppreference.
   URL: https://en.cppreference.com/w/cpp/algorithm/next_permutation
   (дата обращения: 01.06.2022).
2. Задача о коммивояжера // Википедия. Свободная энциклопедия.
   URL: https://ru.wikipedia.org/wiki/Задача_коммивояжёра
   (дата обращения: 01.06.2022).
3. Введение в оптимизацию. Имитация отжига // Хабр.
   URL: https://habr.com/ru/post/209610/
   (дата обращения: 01.06.2022).
4. List-Based Simulated Annealing Algorithm for Traveling Salesman Problem // Hindawi.
   URL: https://www.researchgate.net/figure/Flowchart-of-simulated-annealing-algorithm_fig1_298209081
   (дата обращения: 01.06.2022).
5. Statistical Science: Simulated Annealing / D. Bertsimas, J. Tsitsiklis // Ref. Libr. – 1993. Vol. 8. № 1. – P.10–15.