Министерство образования и науки 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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 6

По курсу "Анализ Алгоритмов"

Муравьиный алгоритм

Студент:

Чаушев Александър

Группа: ИУ7-56Б

Преподователи:

Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2020 г.

Содержание

Введение

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Целью данной лабораторной работы является изучение муравьиных алгоритмов и приобретение навыков параметризации методов на примере муравьиного алгоритма, примененного к задаче коммивояжера.

В ходе лабораторной предстоит выполнить следующие задачи:

1. рассмотреть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
2. реализовать эти алгоритмы;
3. сравнить время работы алгоритма при разных значениях.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы задачи коммивояжера и муравьиного алгоритма.

1.1 Постановка задачи

Имеется сильно связный взвешенный ориентированный граф с положительными весами, заданный в виде матрицы смежностей. Количество вершин в нем лежит в диапазоне от 5 до 20. Требуется решить задачу коммивояжера для этого графа.

1.2 Задача коммивояжера

Коммивояжёр — бродячий торговец. Задача коммивояжера — особая задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры лучшего маршрута может служить суммарное время в пути, суммарная стоимость дороги, или просто длина маршрута.

1.3 Решение полным перебором

Задача может быть решена перебором всех вариантов объезда и выбором оптимального варианта. Но при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n!. К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом — очень долгое вычисление. Не спасает в этой ситуации даже то, что часть вариантов, которые будут повторяться из-за возможности пройти по графу в обратном направлении можно сократить до $\frac{n!}{2n}$ вариантов.

1.4 Муравьиный алгоритм

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений. Все муравьиные алгоритмы базируются на моделировании поведения колонии муравьев. Колония муравьев может рассматриваться как многоагентная система, в которой каждый муравей функционирует автономно по очень простым правилам.

Идея муравьиного алгоритма - моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь. При своём движении муравей опрыскивает путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным.

Какие же механизмы обеспечивают столь сложное поведение муравьев, и что можем мы позаимствовать у природы для решения своих глобальных задач? Основу социального поведения муравьев составляет самоорганизация — множество динамических механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Принципиальной особенностью такого взаимодействия является использование элементами системы только локальной информации. При этом исключается любое централизованное управление и обращение к глобальному образу, репрезентирующему систему во внешнем мире. Самоорганизация является результатом взаимодействия следующих четырех компонентов:

- случайность;
- многократность;
- положительная обратная связь;
- отрицательная обратная связь.

Рассмотрим случай, когда на оптимальном до сих пор пути возникает преграда. В этом случае необходимо определение нового оптимального пути.

Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако те, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь быстро приведёт к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьёв. Моделирование испарения феромона - отрицательной обратной связи - гарантирует нам, что найденное локально оптимальное решение не будет единственным - муравьи будут искать и другие пути. Если мы моделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма.

Обобщим все выше сказанное. Муравьиный алгоритм, независимо от модификаций, представим в следующем виде:

- Создание муравьев;
- Поиск решения;
- Обновление феромона;

Теперь рассмотрим каждый шаг в цикле более подробно:

1. Создание муравьев

Стартовая точка, куда помещается муравей, зависит ограничений, накладываемых условиями задачи. Потому что для каждой задачи способ размещения муравьёв является определяющим. Либо все они помещаются в одну точку, либо все в разные с повторениями, либо без повторений.

На этом же этапе задается начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероят-

ности перехода в следующую вершину не были нулевыми.

2. Поиск решения

Вероятность перехода из вершины i в вершину j определяется по следующей формуле1

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1}$$

где $\tau_{i,j}$— расстояние от города i до j;

$\eta_{i,j}$—количество феромонов на ребре ij;

α— параметр влияния длины пути;

β— параметр влияния феромона.

3. Обновление феромона

Уровень феромона обновляется в соответствии с приведённой формулой:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{2}$$

где $\rho_{i,j}$ - доля феромона, который испарится;

$\tau_{i,j}$ - количество феромона на дуге ij;

$\Delta\tau_{i,j}$ - количество отложенного феромона, вычисляется по формуле 4.

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах след, обратно пропорциональный длине пройденного пути. Итого, новый след феромона вычисляется по формуле 2.

1.5 Муравьиный алгоритм в задаче коммивояжера

Рассмотрим, как реализовать четыре составляющие самоорганизации муравьев при оптимизации маршрута коммивояжера. Многократность взаимодействия реализуется итерационным поиском маршрута коммивояжера одновременно несколькими муравьями. При этом каждый муравей рассматривается как отдельный, независимый коммивояжер, решающий свою задачу. За одну итерацию алгоритма каждый муравей совершаетполный маршрут коммивояжера.Положительная обратная связь реализуется как имитация поведения муравьев типа «оставление следов – перемещение по следам». Чем больше следов оставлено на тропе — реб-

ре графа в задаче коммивояжера — тем больше муравьев будет передвигаться по ней. При этом на тропе появляются новые следы, привлекающие дополнительных муравьев. Для задачи коммивояжера положительная обратная связь реализуется следующим стохастическим правилом: вероятность включения ребра графа в маршрут муравья пропорциональна количеству феромона на нем.

Теперь с учетом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьев при выборе пути.

1. Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через J список городов, которые необходимо посетить муравью k , находящемуся в городе i .

2. Муравьи обладают «зрением» - видимость есть эвристическое желание посетить город j , если муравей находится в городе i . Будем считать, что видимость обратно пропорциональна расстоянию между городами.

3. Муравьи обладают «обонянием» - они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{i,j}(t)$

4. На этом основании мы можем сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода k-ого муравья из города i в город j.

5. Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде:

$$\Delta\tau_{i,j}^{k} = \{\, Q\ /L_k\text{Если k-ый мурваей прошел по ребру ij;0Иначе} \qquad (3)$$

где Q - количество феромона, переносимого муравьем;

Тогда

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \quad (4)$$

где k - количество муравьев в вершине графа с индексами i и j.

1.6 Вывод

В данном разделе были рассмотрены общие принципы муравьиного алгоритма и их применение его к задаче коммивояжера.

2 Конструкторская часть

В данном разделе будут рассмотрены основные требования к программе и схемы алгоритмов.

Требования к вводу: у ориентированного графа должно быть минимум 2 вершины.

Требования к программе: алгоритм полного перебора должен возвращать кратчайший путь в графе.

Входные данные - матрица смежности графа.

Выходные данные - самый выгодный путь в виде списка рёбер.

2.1 Схемы алгоритмов

На рисунках 1 приведена схема алгоритма решения задачи коммивояжера.

Рис. 1: Схема алгоритма решения задачи коммивояжера.

2.2 Вывод

В данном разделе были рассмотрены требования к программе и схемы этих алгоритмов.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

1. программа должна корректно находить кратчайший путь;
2. программа должна обеспечить возможность замера времени работы алгоритма.

3.2 Средства реализации

В качестве языка программирования был выбран `Kotlin`. Данный язык имеет полную совместимость с Java. Как и Java, C и C++, Kotlin — это статически типизированный язык. Он поддерживает как объектно-ориентированное, так и процедурное программирование. Программа, написанная на `Kotlin`, будет доступна на всех платформах.

3.3 Листинг кода

В данном пункте представлен листинг кода.

На листинге 1 представлен код муравьиного алгоритма.

Листинг 1: Муравьиный алгоритм

```
    fun findGoodPath(map: Array<IntArray>): Pair<ArrayList<Int>, Int> {
var pheromonsOnVert = getPheromoneOnVert(map)

var distance = genRandomCitiesDistance()
var tMin = ArrayList<Int>()
var lMin = 0

var ant = 0
while (ant < MAX_ANTS) {
    var pheromonOnVert =
    Array(ANTS_AND_CITIES) { DoubleArray(ANTS_AND_CITIES){0.0} }

    for (k in 0 until ANTS_AND_CITIES) {
        var visited = ArrayList<Int>()
        visited.add(k)
        var lenK = 0 // lenghth of K path
        var i = k // current City

        while (visited.size != ANTS_AND_CITIES) {
            var cities = ArrayList<Int>()
            for (i in 0 until ANTS_AND_CITIES) cities.add(i)

            visited.forEach {
                cities.remove(it)
            }

            var probability = ArrayList<Double>()
            cities.forEach {
                probability.add(0.0)
```

```
}

cities.forEach {
    if (map[i][it] != 0) {

        var tempArray = ArrayList<Double>()
        cities.forEach {l ->
            var a = Math.pow(distance[i][l],
            PATH_INFLUENCE_COEFF)
            var b = Math.pow(pheromonsOnVert[i][l],
            PHEROMONE_COEFF)
            tempArray.add(a * b)
        }
        var buf = tempArray.sum()

        probability[cities.indexOf(it)] =
        Math.pow(distance[i][it],
        PATH_INFLUENCE_COEFF) *
        Math.pow(pheromonsOnVert[i][it],
        PHEROMONE_COEFF) / buf
    } else {
        probability[cities.indexOf(it)] = 0.0
    }
}

var maxProbability = probability.maxOrNull()
if (maxProbability == 0.0) //check
    break

var indexOfChosenCity = probability.indexOf(maxProbability)
visited.add(cities[indexOfChosenCity])
lenK += map[i][cities[indexOfChosenCity]]
//pop
i = cities[indexOfChosenCity]
```

```
                cities.removeAt(indexOfChosenCity)
            }
            var city = map[visited.first()][visited.last()]
            var newLen = lenK + city
            if ((lMin == 0) || newLen < lMin) { //check
                lMin = lenK + map[visited.first()][visited.last()]
                tMin = visited
            }

            for (g in 0 until visited.size - 1) {
                var a = visited[g]
                var b = visited[g + 1]
                pheromonOnVert[a][b] = BEST_PATH_COEFF.toDouble() / lenK
            }

        }

        var leftPheromone =
        if (lMin != 0)
        (PATHFINDER_ANTS * BEST_PATH_COEFF.toDouble() / lMin)
        else 0.0

        distance = update(distance, pheromonOnVert, leftPheromone)

        ant++
    }

    return Pair(tMin, lMin)
}
```

3.4 Вывод

В данном разделе была представлена структура ПО и листинги кода программы.

4 Исследовательская часть

В данном разделе будет проведен эксперимент и сравнительный анализ.

4.1 Системные характеристики

Характеристики компьютера на котором проводился замер времени сортировки массива:

1. операционная система - Unbuntu
2. процессор - Intel(R) Core(TM) i5-4210U CPU 2.60GHz;
3. оперативная память - 8 ГБ;
4. количество ядер - 2;
5. количество логических процессов - 4.

4.2 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. сравнение и анализ времени работы алгоритма при разных входных данных.

4.3 Сравнительный анализ на основе замеров времени работы программы

Был проведен замер времени работ.

В Таблице 1 показаны результаты замера времени при разных входных параметрах. Число муравьев варировалось от 100 до 500. Количество потоков было равно 10. Ниже приведена полученная таблица:

Таблица 1: Результаты временного замера при разных входных параметрах.

N	t	Q	e	α	β	ρ
100	0.03125	5	0	1	1	0.6
100	0.14062	50	0	2	3	0.3
100	0.12500	5	0	3	3	0.3
100	0.12500	25	3	2	2	0.2
100	0.10938	25	1	2	2	0.1
200	0.20312	25	3	1	3	0.7
200	0.23438	25	0	3	1	0.1
200	0.21875	25	2	1	3	0.4
200	0.31250	25	0	1	3	0.5
200	0.26562	25	2	1	1	0.2
300	0.53125	50	0	2	2	0.3
300	0.51562	25	1	2	3	0.6
300	0.53125	5	3	1	3	0.3
300	0.37500	50	0	1	3	0.2
300	0.43750	50	0	1	3	0.5
400	0.56250	25	0	1	3	0.2
400	0.60938	5	3	3	2	0.2
400	0.75000	50	0	1	2	0.1
400	0.56250	50	3	3	3	0.4
400	0.57812	50	2	3	2	0.7
500	1.01562	5	0	1	1	0.7
500	0.90625	5	0	1	1	0.5
500	0.95312	50	2	3	3	0.7
500	1.06250	25	0	1	3	0.2
500	0.78125	25	0	1	2	0.6

В результате проведенных испытаний было установлено, что:

1. Судя по проведенным выше дааным, самое точное приближение не всегда является самым выгодным. Проверим это утверждение на тестовом случае с фиксацией всех параметров, кроме Q. Проведем несколько тестовых запусков с $Q \in [1, 50]; t = 200; m = 5, e = 1; \rho = 0.3; \alpha = 1; \beta = 1$. На Рисунке 2 показаны результаты этого эксперимента:

   Как видим результат варьируется от случая к случаю.

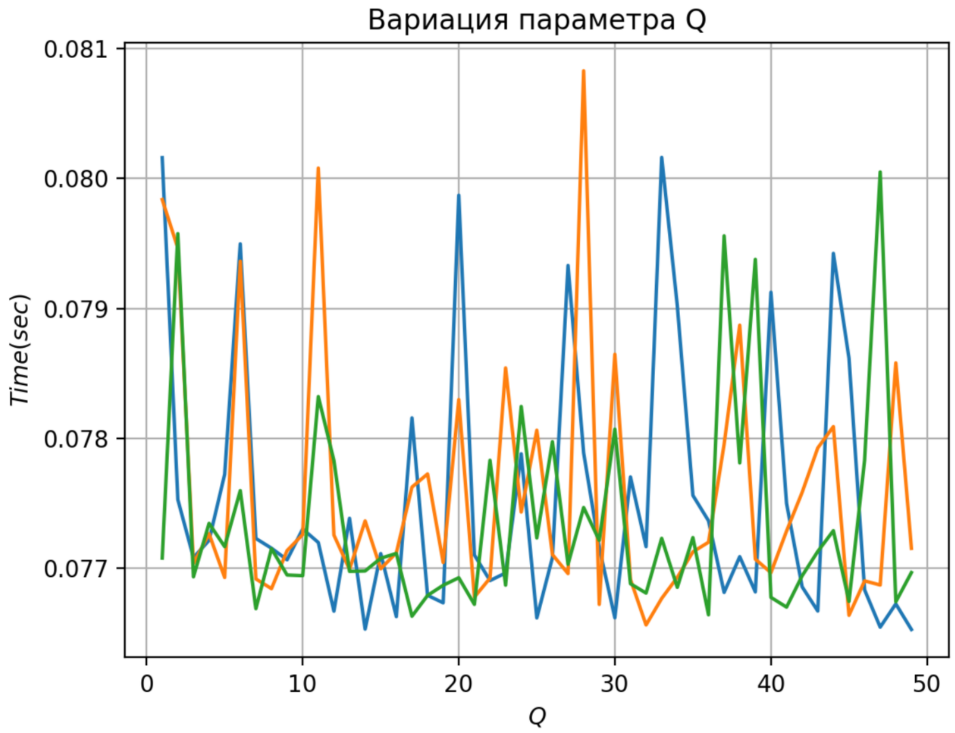

Рис. 2: Эксперимент с $Q \in [1, 50]; t = 200; m = 5, e = 1; \rho = 0.3; \alpha = 1; \beta = 1$.

2. Наличие элитных муравьев улучшает сходимость. Однако в оптимальных наборах их количество сильно связано с коэффициентами $\alpha\beta$, отвечающими за мощность феромонного запаха и за желание муравья передвигаться по этому пути соответственно. Объяснить это можно так: так и тот, и другой аспект алгоритма усиливает феромоновую дорожку, то их подбор должен быть сбалансирован, так как возможно "зависание"на локальных эсктремумах. На Рисунке 3 показаны графики вариации компонентов e, α, β, ρ:

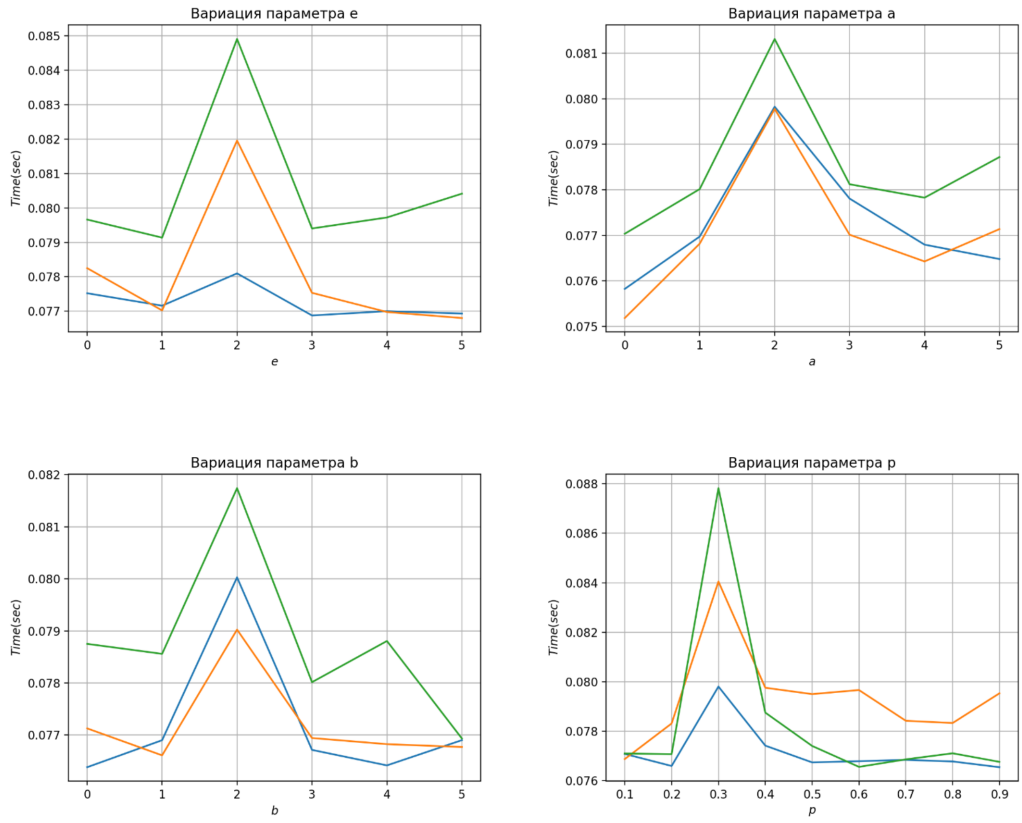

Рис. 3: Эскперимент с вариацией комнонетов e, α, β, ρ.

3. На графиках видны "нежелательные"значения параметров: $e = 2, \alpha = 2, \beta = 2, \rho = 0.3$. Это может быть обусловлено математическими особенностями алгоритма.

4.4 Вывод

По проведенному анализу эксперимента можно сделать вывод, что самое точное приближение не всегда является самым выгодным. Наличие элитных муравьев улучшает сходимость. Однако в оптимальных наборах их количество сильно связано с коэффициентами $\alpha\beta$, отвечающими за мощность феромонного запаха и за желание муравья передвигаться по этому пути соответственнно. Объяснить это можно так: так и тот, и другой аспект алгоритма усиливает феромоновую дорожку, то их подбор должен быть сбалансирован, так как возможно "зависание"на локальных эсктремумах.

Заключение

В рамках данной работы успешно изучены основы муравьиного алгоритма. Применен метод асинхронного программирования. Проведен сравнительный анализ при разных параметрах. Подтверждены экспериментально "нежелательные"параметры значения параметро. Дано описание и обоснование полученных результатов.