МГТУ им. Баумана

Лабораторная работа №6

По курсу: "Анализ алгоритмов"

Муравьиный алгоритм

Работу выполнил: Мокеев Даниил, ИУ7-54

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

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.

Задачи данной лабораторной работы:

- рассмотренть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
- реализовать эти алгоритмы;
- сравнить время работы этих алгоритмов.

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 [1] 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 (фр. commis voyageur) — бродячий торговец. Задача коммивояжёра — важная 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 выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута [2].

1.3 Решение полным перебором

Задача может быть решена перебором всех вариантов объезда и выбором оптимального. Но при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (оно равно n! — количеству способов упорядочения пунктов). К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом — не выдержит ни один калькулятор! Мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет биться с задачей на протяжении примерно 3*10144 лет. Увеличение производительности ЭВМ в 1000 раз даст хоть и меньшее в 1000 раз, но по-прежнему чудовищное время перебора вариантов. Не спасает ситуацию даже то, что для каждого варианта маршрута имеется 2n равноценных, отличающихся выбором начального пункта (n вариантов) и направлением обхода (2 варианта). Перебор с учётом этого наблюдения сокращается незначительно [3].

1.4 Муравьиные алгоритмы

Все муравьиные алгоритмы базируются на моделировании поведения колонии муравьев. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент (муравей) функционирует автономно по очень простым правилам. В противовес почтипримитивному поведению агентов, поведение всей системы получается на удивление разумным.

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений.

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[4]. При своём движении муравей метит 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 этих крошечных существ длярешения 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 [5] :

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

Рассмотрим случай, когда на оптимальном доселе 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 муравьи, 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. Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде:

- Создание муравьев;
- Поиск решения;
- Обновление феромона;
- Дополнительные действия (опционально).

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 в разные с повторения, либо без повторений.

На этом же этапе задается начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероятности перехода в следующую вершину не были нулевыми.

2. Поиск решения

Вероятность перехода из вершины i в вершину j определяется по следующей формуле1.1

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

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

$\eta_{i,j}$—эвристическое расстояние от i до j;

α— параметр влияния феромона;

β— параметр влияния расстояния.

3. Обновление феромона

Уровень феромона обновляется в соответствии с приведённой формулой:

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 1.2:

$$\tau_{i,j} = (1 - \rho_{i,j})\tau_{i,j} + \Delta\tau_{i,j}, \tag{1.2}$$

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 1.4.

4. Дополнительные действия

Обычно здесь используется алгоритм локального поиска, однако он может также появиться и после поиска всех решений.

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 иттерацию алгоритма каждый муравей совершает полный маршрут коммивояжера. Положительная обратная связь реализуется как имитация поведения муравьев типа «оставление следов – перемещение по следам». Чем больше следов оставлено на тропе — ребре 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 мура-

вьёв. 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:

$$\Delta\tau_{i,j}^k = \begin{cases} Q/L_k & \text{Если k-ый мурваей прошел по ребру ij;} \\ 0 & \text{Иначе} \end{cases} \tag{1.3}$$

где Q - количество феромона, переносимого муравьем;

Тогда

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \tag{1.4}$$

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

Вывод

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

2 Конструкторская часть

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

2.1 Требования к программе

Требования к вводу:

- У ориентированного графа должно быть хотя бы 2 вершины.

Требования к программе:

- Алгоритм полного перебора должен возвращать кратчайший путь в графе.

.
Входные данные - матрица смежности графа.
Выходные данные - самый выгодный путь.

2.2 Схемы алгоритмов

В данном разделе будут приведены схемы алгоритмов для решения задачи коммивояжора: полный пребор(Рис.2.1) и муравьиный (Рис. 2.2)

Рис. 2.1: Схема алгоритма полного перебора

Рис. 2.2: Схема муравьиного агоритма

Вывод

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

3 Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран golang. Время работы алгоритмов было замерено с помощью time.

3.2 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода полного перебора всех решений (Листинг 3.1) и реализации муравьиного алгоритма (Листинг 3.2)

Листинг 3.1: Перебор всех возможных вариантов

```
1
2 func brute(file_name string) []int{
3   weight := get_weights(file_name)
4   path := make([]int, 0)
5   res := make([]int, len(weight))
6
7   for k:=0; k<len(weight);k++{
8     ways := make([][]int, 0)
9     _ = go_route(k, weight, path, &ways)
10    sum := 1000
11    curr := 0
12    ind := 0
13    for i:=0; i<len(ways);i++{
14      curr = 0
15      for j:=0;j<len(ways[i])-1;j++{
16        curr+=weight[ways[i][j]][ways[i][j+1]]
17      }
18      if curr < sum{
19        sum = curr
20      }
21    }
22    res[k] = sum
23  }
24  return res
25 }
26
27 func contains(s []int, e int) bool {
```

```
28    for _, a := range s {
29      if a == e {
30        return true
31      }
32    }
33    return false
34  }
35
36
37  func go_route(pos int, weight [][]int, path []int, routes *[][]int)[]int
        {
38    path = append(path, pos)
39    if len(path) < len(weight){
40      for i:=0; i < len(weight); i++{
41        if !(contains(path, i)){
42          _ = go_route(i, weight,path, routes)
43        }
44      }
45      }else{
46        *routes = append(*routes, path)
47    }
48    return path
49  }
```

Листинг 3.2: Муравьиный алгоритм

```
1
2  func start (env *env, days int) []int{
3    shortest_dist := make([]int, len(env.weight))
4    for n := 0; n < days; n++{
5      for i:= 0; i< len(env.weight); i++{
6        ant := env.new_ant(i)
7        ant.ant_go()
8
9        cur_dist := ant.get_distance()
10       if (shortest_dist[i] == 0) || (cur_dist < shortest_dist[i]){
11         shortest_dist[i] = cur_dist
12       }
13     }
14   }
15   return shortest_dist
16 }
17
18 func (ant *ant) ant_go(){
```

```
19    for{
20      prob := ant.count_probapility()
21      choosen_path := choose_path(prob)
22      if choosen_path == -1{
23        break}
24      ant.go_path(choosen_path)
25      ant.renew_pheromon()
26    }
27  }
28
29  func (ant *ant)count_probapility() []float64{
30    res := make([]float64, 0);
31    var d float64;
32    var sum float64;
33    for i, lenght := range ant.visited[ant.position]{
34      if lenght != 0{
35        d = math.Pow((float64(1)/float64(lenght)), ant.env.alpha) * math.
              Pow(ant.env.pheromon[ant.position][i], ant.env.betta)
36        res = append(res, d)
37        sum += d
38      } else{
39        res = append(res, 0)
40      }
41      }
42      for _, lenght := range res{
43        lenght = lenght / sum
44    }
45    return res
46  }
47
48  func choose_path(probab []float64) int{
49    var sum float64
50    for _, j := range probab{
51      sum += j
52    }
53    r := rand.New(rand.NewSource(time.Now().UnixNano()))
54    random_fl := r.Float64() * sum
55    sum = 0
56    for i , j := range probab{
57      if random_fl > sum && random_fl<sum+j{
58        return i
59      } else{
60      sum+=j
```

```
61      }
62    }
63    return -1
64 }
```

Вывод

В данном разделе были рассмотрены основные сведения о модулях программы и листинг кода алгоритмов.

4 Исследовательская часть

В даннном разделе будет проведен сравнительный временной анализ алгоритмов и рассмотрена параметризация муравьиного алгоритма. Замеры времени были произведены на: Intel Core i5-6200U.

4.1 Исследование зависимости времени работы алгоритмов от размера графа

В данном разделе будет приведены результаты сравнения времени работы реализованных алгоритмов в зависимоти от размера матрицы смежности (Рис. 4.1). Время измерено в миллисекундах.

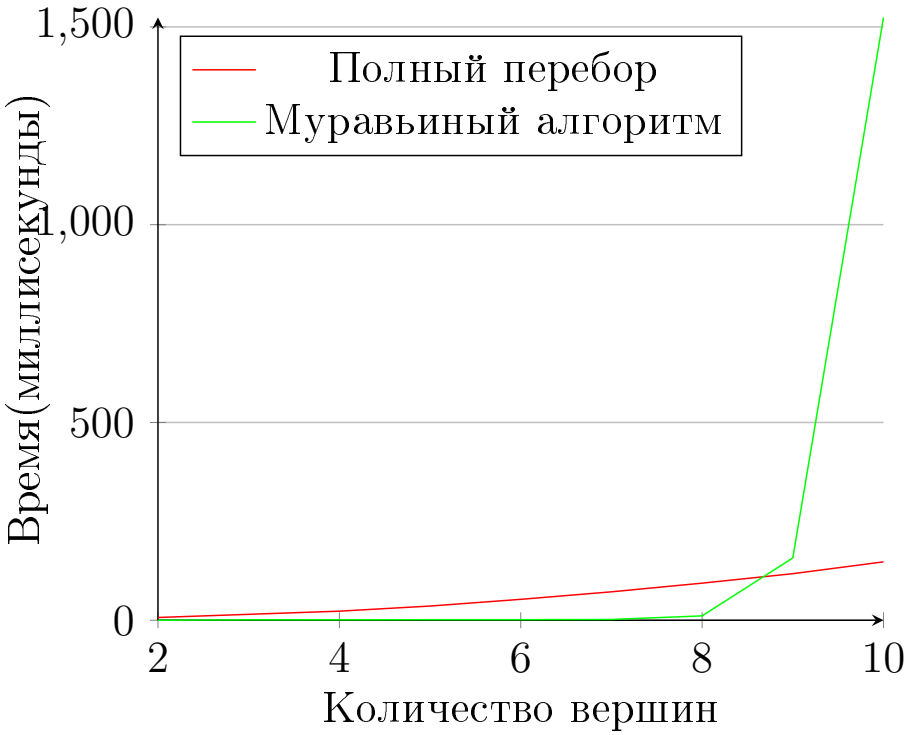

Рис. 4.1: Сравнение параллельного и обычного алгоритмов

4.2 Выводы исследовательского раздела

Была исследована зависимоть времени работы реализованных алгоритмов от размера матрицы смежности графа. По результатам эксперимента на малых размерах графа полный перебор значительно выигрывает муравьиных алгоритм в скорости, однако на размера графа больше 8 сложность полного перебора растет очень быстро,

а так как муравьиный алгоритм обладает полиноминальной сложностью, он работает быстрее перебора.

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы решения задачи коммивояжера - полный перебор и муравьиный алгоритм.

Временной анализ показал, что неэффективно использовать полный перебор на графе размера больше 8.

Литература

[1] Белоусов А.И., Ткачев С.Б(2006). Дискретная математика, 4-е издание.

[2] Т.М. Товстик, Е.В. Жукова - Алгоритм приближенного решения задачи коммивояжера.

[3] Задача коммивояжера[Электронный ресурс] - режим доступа http://mech.math.msu.su/ shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml

[4] Муравьиные алгоритмы[Электронный ресурс] - режим доступа http://www.machinelearning.ru/wiki/index.php?title=

[5] Штовба С.Д. - Муравьиные алгоритмы.

[6] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16