*Государственное образовательное учреждение высшего профессионального образования*

*«Московский Государственный Технический Университет имени Н. Э. Баумана»*

ОТЧЕТ

По лабораторной работе №7

По курсу «Анализ алгоритмов»

Тема: «Решение задачи коммивояжера»

|  |  |
| --- | --- |
| Студент: | Терентьев В.А. |
| Группа: | ИУ7-52 |
| Преподаватель: | Волкова Л.Л. |

Москва 2017

Оглавление

**[Постановка задачи.](#_Toc500448652)** [3](#_Toc500448652)

[**Алгоритм муравьиной колонии.** 4](#_Toc500448653)

[**Листинг** 6](#_Toc500448654)

[**Тестирование** 10](#_Toc500448655)

[**Заключение** 16](#_Toc500448656)

[**Список использованной литературы** 17](#_Toc500448657)

# **Постановка задачи.**

Написать программу решение задачи коммивояжера используя алгоритм муравьиной колонии. Найти оптимальные параметры алгоритма, при которых задача решается относительно недолго.

# **Алгоритм муравьиной колонии.**

*Начальная популяция*

После создания популяция муравьев поровну распределяется по узлам сети.

Необходимо равное разделение муравьев между узлами, чтобы все узлы имели

одинаковые шансы стать отправной точкой. Если все муравьи начнут движение

из одной точки, это будет означать, что данная точка является оптимальной для

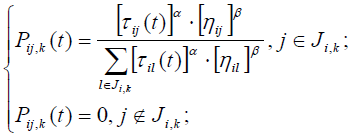
старта, а на самом деле мы этого не знаем.

*Движение муравья*

Движение муравья основывается на одном и очень простом вероятностном

уравнении. Если муравей еще не закончил путь, то есть не посетил все узлы

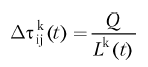
сети, для определения следующей грани пути используется уравнение:



Здесь τ – интенсивность фермента на грани между узлами i, j и u, η –функция, которая представляет измерение обратного расстояния для грани, α –вес фермента, а β – коэффициент эвристики. Параметры α и β определяют относительную значимость двух параметров, а также их влияние на уравнение.

*Путешествие муравья*

Пройденный муравьем путь отображается, когда муравей посетит все узлы диаграммы. Обратите внимание, что циклы запрещены, поскольку в алгоритм включен список табу. После завершения длина пути может быть подсчитана – она равна сумме всех граней, по которым путешествовал муравей.



Уравнение показывает количество фермента, который был оставлен на каждой грани(i,j) пути для муравья k.

Переменная Q является константой порядка всего пути.

*Испарение фермента*

В начале пути у каждой грани есть шанс быть выбранной. Чтобы постепенно

удалить грани, которые входят в худшие пути в сети, ко всем граням применяется

процедура испарения фермента (Pheromone evaporation). Используя константу ρ, мы получаем уравнение

Поэтому для испарения фермента используется обратный коэффициент обновления пути.

# **Листинг**

void init()

{

int from, to, ant;

//creating cities

for (from = 0; from < MAX\_CITIES; from++)

{

//randomly place cities

cities[from].x = rand() % MAX\_DIST;

cities[from].y = rand() % MAX\_DIST;

//printf("\n %d %d",cities[from].x, cities[from].y);

for (to = 0; to<MAX\_CITIES; to++)

{

dist[from][to] = 0.0;

phero[from][to] = INIT\_PHER;

}

}

//computing distance

for (from = 0; from < MAX\_CITIES; from++)

{

for (to = 0; to < MAX\_CITIES; to++)

{

if (to != from && dist[from][to] == 0.0)

{

int xd = pow(abs(cities[from].x - cities[to].x), 2);

int yd = pow(abs(cities[from].y - cities[to].y), 2);

dist[from][to] = sqrt(xd + yd);

dist[to][from] = dist[from][to];

}

}

}

to = 0;

for (ant = 0; ant < MAX\_ANTS; ant++)

{

if (to == MAX\_CITIES)

to = 0;

ants[ant].curCity = to++;

for (from = 0; from < MAX\_CITIES; from++)

{

ants[ant].tabu[from] = 0;

ants[ant].path[from] = -1;

}

ants[ant].pathIndex = 1;

ants[ant].path[0] = ants[ant].curCity;

ants[ant].nextCity = -1;

ants[ant].tourLength = 0;

//loading first city into tabu list

ants[ant].tabu[ants[ant].curCity] = 1;

}

}

void MyMainWindow::restartAnts()

{

int ant, i, to = 0;

for (ant = 0; ant<MAX\_ANTS; ant++)

{

if (ants[ant].tourLength < best)

{

best = ants[ant].tourLength;

bestIndex = ant;

UpdatePicture();

qDebug() << best;

}

ants[ant].nextCity = -1;

ants[ant].tourLength = 0.0;

for (i = 0; i<MAX\_CITIES; i++)

{

ants[ant].tabu[i] = 0;

ants[ant].path[i] = -1;

}

if (to == MAX\_CITIES)

to = 0;

ants[ant].curCity = to++;

ants[ant].pathIndex = 1;

ants[ant].path[0] = ants[ant].curCity;

ants[ant].tabu[ants[ant].curCity] = 1;

}

}

double antProduct(int from, int to)

{

return((pow(phero[from][to], ALPHA) \* pow((1.0 / dist[from][to]), BETA)));

}

int selectNextCity(int ant)

{

int from, to;

double denom = 0.0;

from = ants[ant].curCity;

for (to = 0; to < MAX\_CITIES; to++) { if (ants[ant].tabu[to] == 0) { denom += antProduct(from, to); } } assert(denom != 0.0); do {

double p; to++; if (to >= MAX\_CITIES)

to = 0;

if (ants[ant].tabu[to] == 0)

{

p = antProduct(from, to) / denom;

double x = ((double)rand() / RAND\_MAX);

if (x < p)

{

break;

}

}

} while (1);

return to;

}

int simulateAnts()

{

int k;

int moving = 0;

for (k = 0; k<MAX\_ANTS; k++)

{

if (ants[k].pathIndex < MAX\_CITIES) {

ants[k].nextCity = selectNextCity(k);

ants[k].tabu[ants[k].nextCity] = 1;

ants[k].path[ants[k].pathIndex++] = ants[k].nextCity;

ants[k].tourLength += dist[ants[k].curCity][ants[k].nextCity];

if (ants[k].pathIndex == MAX\_CITIES)

{

ants[k].tourLength += dist[ants[k].path[MAX\_CITIES - 1]][ants[k].path[0]];

}

ants[k].curCity = ants[k].nextCity;

moving++;

}

}

return moving;

}

void updateTrails()

{

int from, to, i, ant;

for (from = 0; from<MAX\_CITIES; from++)

{

for (to = 0; to<MAX\_CITIES; to++)

{

if (from != to)

{

phero[from][to] \*= (1.0 - RHO);

if (phero[from][to]<0.0)

{

phero[from][to] = INIT\_PHER;

}

}

}

}

for (ant = 0; ant<MAX\_ANTS; ant++)

{

for (i = 0; i<MAX\_CITIES; i++)

{

if (i < MAX\_CITIES - 1)

{

from = ants[ant].path[i];

to = ants[ant].path[i + 1];

}

else

{

from = ants[ant].path[i];

to = ants[ant].path[0];

}

phero[from][to] += (QVAL / ants[ant].tourLength);

phero[to][from] = phero[from][to];

}

}

for (from = 0; from < MAX\_CITIES; from++)

{

for (to = 0; to<MAX\_CITIES; to++)

{

phero[from][to] \*= RHO;

}

}

}

# **Тестирование**

Для тестирования данного было выбрана 60 городов, расставленных в произвольном порядке. Каждый раз запускался таймер, равный 6000 временным циклам. И в этот интервал времени мы пытаемся получить наиболее меньшее расстояние. Попробуем найти такие параметры α, β, ρ, при которых алгоритм работает максимально эффективно. Почему были выбраны такие значения для фиксации переменных, которые не используются в параметризации будет объяснено после тестов.

Пусть β =1, ρ = 0.9

|  |  |  |
| --- | --- | --- |
| α | Расстояние |  |
| 0 | 6538.93 |  |
| 0,2 | 4246.47 |  |
| 0,4 | 3709.38 |  |
| 0,6 | 3340.41 |  |
| 0,8 | 2938.36 |  |
| 1 | 2574.63 |  |
| 1,2 | 2836.14 |  |
| 1,4 | 2898.8 |  |

Теперь найдем такое b, при котором алгоритм работает эффективней всего.

α = 1, ρ = 0.9

|  |  |  |
| --- | --- | --- |
| β | Расстояние |  |
| 0.5 | 3307.84 |  |
| 1 | 2574.63 |  |
| 2 | 2638.72 |  |
| 3 | 2611.08 |  |
| 4 | 2613.11 |  |
| 5 | 2721.69 |  |

Теперь найдем такое b, при котором алгоритм работает эффективней всего.

α = 1, β = 1

|  |  |  |
| --- | --- | --- |
| ρ | Расстояние |  |
| 0.2 | 2713.43 |  |
| 0.4 | 2698.71 |  |
| 0.5 | 2685.32 |  |
| 0.8 | 2582.43 |  |
| 0.9 | 2574.63 |  |

Как мы можем видеть, эффективнее всего алгоритм работает при a = 1, b = 1, p = 0.9

Марко Дориго (изобретатель оптимизации по принципу муравьиной колонии) предлагает очень интересную дискуссию по параметрам алгоритма в статье «Система муравьев: оптимизация с помощью колонии сотрудничающих агентов». Был открыт ряд комбинаций α/β, которые позволяют находить хорошие решения за небольшое время.[[1]](#footnote-1)

|  |  |
| --- | --- |
| **a** | **b** |
| 0.5 | 5 |
| 1 | 1 |
| 1 | 2 |
| 1 | 5 |

Параметр α ассоциируется с количеством фермента, а параметр β – с видимостью (длинной грани). Чем больше значение параметра, тем он важнее для вероятностного уравнения, которое используется при выборе грани. Обратите внимание, что в одном случае значимость параметров равна. Во всех других случаях видимость более важна при выборе пути.

Как было показано на тестах данные типы параметров действительно соответствуют одним из самых лучших результатов. Также стоит сказать, что р следует выбирать в пределах от 0,5 до 1 не включая, тогда получаются наиболее приемлемые результаты. Другие значения р дают неудовлетворительные результаты.

# **Заключение**

В ходе лабораторной работы были реализована программа решения задачи коммивояжера используя алгоритм муравьиной колонии. Были найдены оптимальные параметры алгоритма, при которых задача решается относительно надолго.

**Список использованной литературы**

1. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс .— 2-е изд. — М. : ДМК-Пресс, 2011 .— 313 с. : ил. — пер. с англ. - ISBN 1-58450-278-9 (англ.) .— ISBN 978-5-94074-746-8

1. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс .— 2011 .— С 82 [↑](#footnote-ref-1)