УДК 519.863: 004.9

Нахождение оптимального пути с помощью муравьиного алгоритма в задаче коммивояжёра

Дроздов Андрей Александрович
Приамурский государственный университет им. Шолом-Алейхема
Студент

Баженов Руслан Иванович
Приамурский государственный университет им. Шолом-Алейхема
к.п.н., доцент, зав. кафедрой информационных систем, математики и методик обучения

Аннотация
В данной статье описывается метод решения задачи коммивояжёра, используя муравьиный алгоритм в программной среде Matlab.
Ключевые слова: SciLab, MatLab, муравьиный алгоритм, задача коммивояжёра.

Finding the optimal path using the ant algorithm in the traveling salesman problem

Drozdov Andrey Alexandrovich
Sholom-Aleichem Priamursky State University
Student

Bazhenov Ruslan Ivanovich
Sholom-Aleichem Priamursky State University
Candidate of pedagogical sciences, associate professor, Head of the Department of Information Systems, Mathematics and teaching methods

Abstract
In this article the method of the solution of a task of the direct-sales representative is described, using an ant algorithm in the program Matlab environment.
Keywords: MatLab, Ant algorithm, traveling salesman problem.

В современном мире существует множество оптимизационных задач, которые решаются различными способами. Например, методом линейного программирования, симплекс методом, с помощью муравьиного алгоритма и других. Целью таких задач является нахождение экстремума (максимума или минимума) функции, которые дадут наиболее оптимальный результат в решении таких задач.

Цель данного исследования заключается в рассмотрении способа решения задачи, в которой необходимо найти минимум функции, используя

муравьиный алгоритм в программной среде MatLab и выявить эффективность данного алгоритма. Описание задачи: «Пусть дана сеть из N «городов». Коммивояжёр, выходящий из какого-нибудь города, желает посетить (N-1) других городов и вернуться в изначальный пункт, расстояния между всеми этими городами известны. Требуется установить в каком порядке коммивояжёру следует посетить города, чтобы суммарное пройденное расстояние было минимальным» [11]. В данной задаче необходимо минимизировать критерии эффективности: время пути, суммарное расстояние и оптимальный путь.

Проблему применения муравьиного алгоритма исследовали многие ученые. В статье А.С. Дочкин показывает как упрощает расчёты муравьиный алгоритм [1]. В исследовании С.С. Семенов, Д.Ф. Ткачев, А.В. Педан, М.Р. Батаев, Е.А. Алисевич, А.П. Гусев, Е.Д. Шарапкалиев, Д.В. Киселев рассказывают что муравьиный алгоритм хорошо себя проявляет в задачах поиска маршрутов [2]. В своей статье В.О. Борознов сравнивает распространённые алгоритмы, используемые для решение задачи коммивояжёра и приходит к выводу, что муравьиный является неким компромиссом между всех алгоритмов[3]. С.Д. Штовба рассказывает про реализацию алгоритма в программированной среде MatLab[4]. В исследовании Г.А. Попов, С.В. Скворцов рассказали что муравьиный алгоритм выдаёт приближённые значения и зависим от внешних данных[5]. М.М. Сердюкова, Т.С. Кучинская выяснили что муравьиный алгоритм эффективен в решение сложных комбинаторных задач оптимизации [6]. М. Чураков, А. Якушев сделали обзор на статью Штовба, и указали достоинства и недостатки муравьиного алгоритма [7]. В своей статье С. П. Шаповалов рассмотрел применимость алгоритма к задаче, и вычислил производительность алгоритма для симметричной и асимметричной задачи коммивояжёра[8]. В своём исследовании I.D.Ariyasingha,T.G.I.Fernando сделали обзор на многокритериальную оптимизацию колонии муравьиных алгоритмов и сравнили их на нескольких задачах с разным количеством муравьев и числом итераций[9]. H.Eldem, E.Ülker использовали муравьиный алгоритм для решения неевклидовой задачи коммивояжёра[10].

В данном исследовании мы решим задачу, в которой требуется найти минимальный путь посещения «город» и построить этот маршрут. Для решения этой задачи будет использован муравьиный алгоритм, реализованный в программе MatLab [12].

Задача формулируется как задача поиска минимального замкнутого маршрута по всем вершинам без повторений. Эта задача является NP-трудной, и точный переборный алгоритм её решения имеет факториальную сложность.

Суть муравьиного алгоритма, состоит в моделирование поведения муравьёв. Так как у муравьёв есть способность быстро находить кратчайший путь от муравейника к источнику пищи и при необходимости адаптироваться к новым условиям. При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Так

муравьи и определяют новый путь, если старый оказывается недоступным. Так муравей, дойдя до преграды, с одинаковой вероятностью будет обходить её справа и слева. То же самое будет происходить и на обратном пути. Чем путь короче, тем больше феромона на нем будет содержаться, медленнее он будет выветриваться, и значит, больше муравьев предпочтут идти по этому маршруту.

Муравьи имеют собственную "память". Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещённых городов, список запретов.

Так как задача коммивояжера изначально имеет заранее известные данные, необходимо их объявить. Так в условии должна быть матрица расстояний. Что это такое? И почему она должна там быть? Задача коммивояжёра подразумевает прохождение определённого маршрута. То есть мы уже изначально знаем те «города», что должны посетить. А это значит, мы знаем расстояние между этими «городами». Таким образом, нам нужна матрица расстояний. Почему матрица? Да потому что, так удобнее смотреть на расстояние между городами.

Возьмём для исследования сеть городов, которые связанны между собой. Следовательно, агент (коммивояжёр) может добраться с одного города в любой другой, который захочет. То есть он не ограничен в передвижениях. На рисунке 1 показана такая сеть.

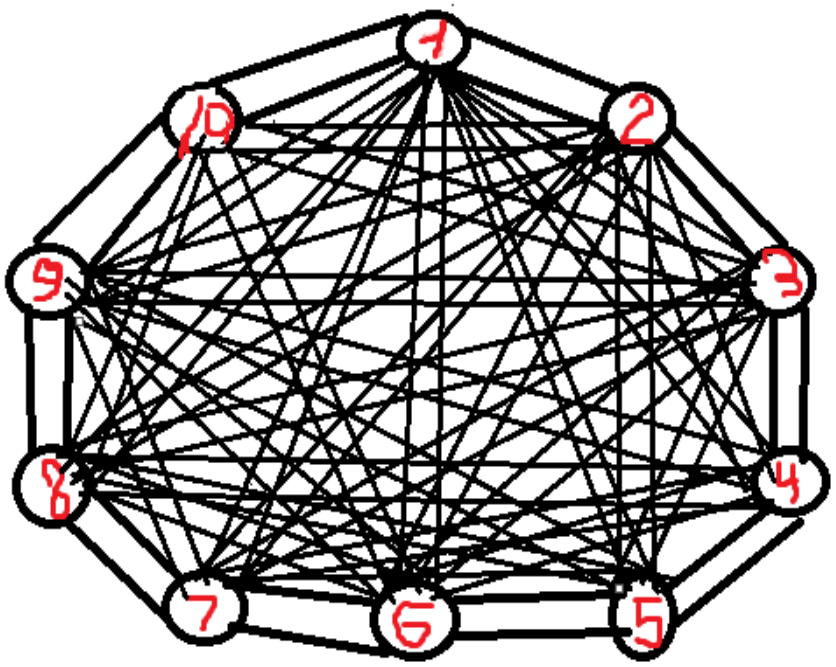

Рисунок 1 – Сеть маршрутов, которые может выбирать муравей

```
0 1.6 2 2.1 2.4 2.3 2.6 2.2 2.2 2.2;
1.6 0 0.4 0.6 0.9 0.8 1.1 0.7 0.7 0.6;
2 0.4 0 0.2 0.5 0.5 0.9 0.5 0.5 0.5;
2.1 0.6 0.2 0 0.4 0.3 0.8 0.5 0.5 0.5;
2.4 0.9 0.5 0.4 0 0.3 0.6 0.4 0.5 0.5;
2.3 0.8 0.5 0.3 0.3 0 0.3 0.1 0.3 0.3;
2.6 1.1 0.9 0.8 0.6 0.3 0 0.4 0.5 0.6;
2.2 0.7 0.5 0.5 0.4 0.1 0.4 0 0.1 0.1;
2.2 0.7 0.5 0.5 0.5 0.3 0.5 0.1 0 0.1;
2.2 0.6 0.5 0.5 0.5 0.3 0.6 0.1 0.1 0;
2 0.4 0.2 0.5 0.7 0.4 0.7 0.2 0.3 0.2;
2.1 0.6 0.2 0.2 0.3 0.3 0.6 0.3 0.3 0.3];
```

Рисунок 2 – Матрица расстояний

Число 1.6, стоящее в первом столбце второй строки, является расстоянием между первым пунктом и вторым. Аналогично расставляются другие значения. В итоге получается матрица, представленная в Рисунке 2

Так же ещё ряд данных.

```
e=0.1;              %коэффициент испарения.
alpha=1;            %эффект зрения муравьев.
beta=4;             %эффект следа.
t=0.1*ones(n);      %первичная трассировка.
el=0.99;            %исключение общей стоимости.
```

Рисунок 3 – Вводные данные

Для решения этой задачи нужен алгоритм муравья, который реализуется кодом представленным ниже.

```
function [new_places]=ant_tour(start_places,m,n,h,t,alpha,beta);
for i=1:m
    mh=h;
    for j=1:n-1
        c=start_places(i,j);
        mh(:,c)=0;
        temp=(t(c,:).^beta).*(mh(c,:).^alpha);
        s=(sum(temp));
        p=(1/s).*temp;
        r=rand;
        s=0;
        for k=1:n
            s=s+p(k);
            if r<=s
                start_places(i,j+1)=k;
                break
            end
        end
    end
end
```

Рисунок 4 – Функция вычисления матрицы тура муравьев в течение одного цикла

```
function [t]=update_the_trace(m,n,t,tour,f,e);
for i=1:m
    for j=1:n
        dt=1/f(i);
        t(tour(i,j),tour(i,j+1))=(1-e)*t(tour(i,j),tour(i,j+1))+dt;
    end
end
```

Рисунок 5 – Функция обновления следов

```
function [cost,f]=calculate_cost(m,n,d,at,el);
for i=1:m
    s=0;
    for j=1:n
        s=s+d(at(i,j),at(i,j+1));
    end
    f(i)=s;
end
cost=f;
f=f-el*min(f);
```

Рисунок 6 – Функция для расчета расстояния тура муравьев

```
for i=1:miter
for j=1:m
    start_places(j,1)=fix(1+rand*(n-1));
end
    [tour]=ant_tour(start_places,m,n,h,t,alpha,beta);
    tour=horzcat(tour,tour(:,1));

    [cost,f]=calculate_cost(m,n,d,tour,el);
    [t]=update_the_trace(m,n,t,tour,f,e);
    average_cost(i)=mean(cost);

    [min_cost(i),best_index]=min(cost);
    besttour(i,:)=tour(best_index,:);
    iteration(i)=i;
end
```

Рисунок 7 – Код, который вызывает муравьиный алгоритм

```
[k,l]=min(min_cost);
for i=1:n+1
    opt_trace(i)=besttour(l,i);
end
toc
disp(['Distance is: ', num2str(k)])
disp('Optimal trace: ')
disp(opt_trace)
end
```

Рисунок 8 – Нахождение лучшего тура и вывод результата

```
Distance is: 6
Optimal trace:
     4     3     2     1     9    10     8     6     7     5     4
```

Рисунок 9 – Итоговый результат

На рисунке 8 представлены итоговые результаты. Показана минимальная дистанция, по которой муравьи прошлись от начального города, по всем городам и обратно. Как видно в пункте Optimal trace, муравьи выбрали оптимальный маршрут, который начинается в «городе» 4 и двигается дальше по оптимальному пути. Число 6, взятое из пункта “Distance is:”, указывает расстояние указанное в условных единицах, которое было преодолено по пути, который выбрали муравьи.

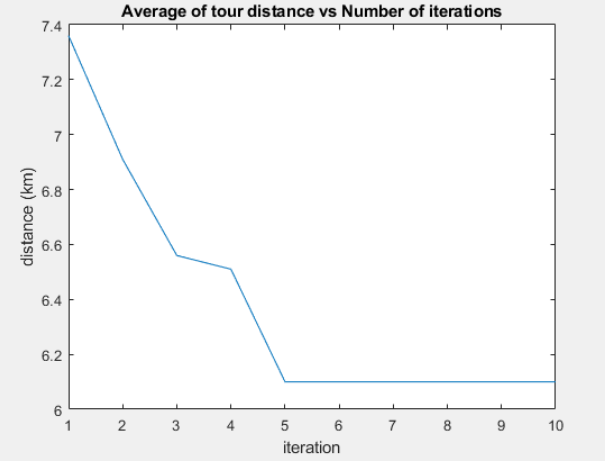

Рисунок 10 – График зависимости дистанции и итераций

На рисунке 10, представлен график зависимости дистанции и итераций, который показывает на какой итерации был найден оптимальный путь. То есть сначала муравьи на первой итерации нашли путь размером 7,4 условных единиц. Далее после ряда итераций агенты находят оптимальный путь.

Для наглядности проведём ещё один эксперимент, но только уже с другой матрицей. На это раз возьмём матрицу более подходящую под реальный мир.

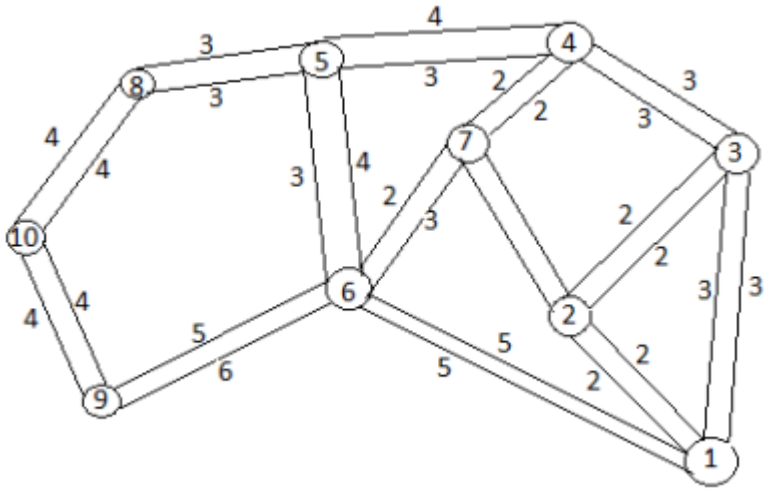

Рисунок 11 – Сеть маршрутов

На основе рисунка 11, создадим матрицу расстояний. Она будет выглядеть следующим образом.

```
0   2   3   999 999 5   999 999 999 999
2   0   2   999 999 999 4   999 999 999
3   2   0   3   999 999 999 999 999 999
999 999 3   0   4   999 2   999 999 999
999 999 999 3   0   3   999 3   999 999
5   999 999 999 4   0   2   999 5   999
999 4   999 2   999 3   0   999 999 999
999 999 999 999 3   999 999 0   999 4
999 2   999 999 999 6   999 999 0   4
999 999 999 999 999 999 999 4   4   0
```

Рисунок 12 – Матрица расстояний

На рисунке 12 видно, что в столбцах присутствуют большие значения, а именно 999. Это связано с тем, что необходимо показать что нет расстояния между определёнными городами. Но MatLab не понимает значение 0 в матрице и выдаёт ошибку связанную с логистикой массива. Поэтому было принято решение на основе алгоритма использовать достаточно большое число, чтобы муравьи не считали данный путь, так как он априори будет огромным. А мы знаем, что муравьиный алгоритм ищет оптимальный минимальный путь, следовательно, путь, где расстояние от одного города до другого будет очень большим, учитываться не будет.

Тогда при запуске программы снова уже с новой матрицей, получается следующий результат.

```
Distance is: 34
Optimal trace:
     1     2     7     6     9    10     8     5     4     3     1
```

Рисунок 13 – Итоговый результат

Как видно из результатов изображённых на Рисунке 13, муравьи выбрали самый оптимальный путь, начиная с пункта 1. А дистанция пройденного пути составляет 34 ус.ед. .

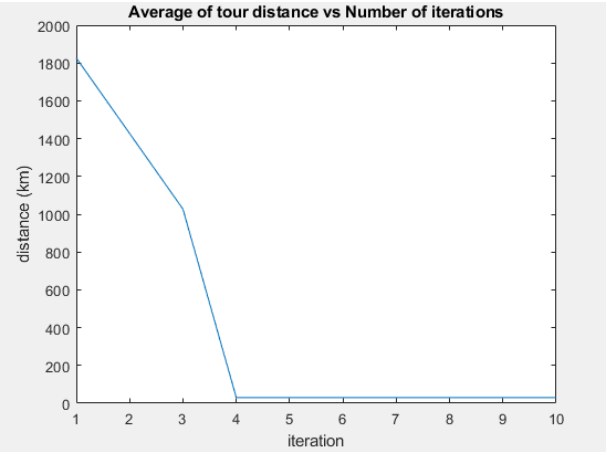

Рисунок 14 – График зависимости дистанции и итераций

Теперь проведём эксперимент с реальными городами. Расстояния и маршруты были взяты для автомобиля из Google карт[13].

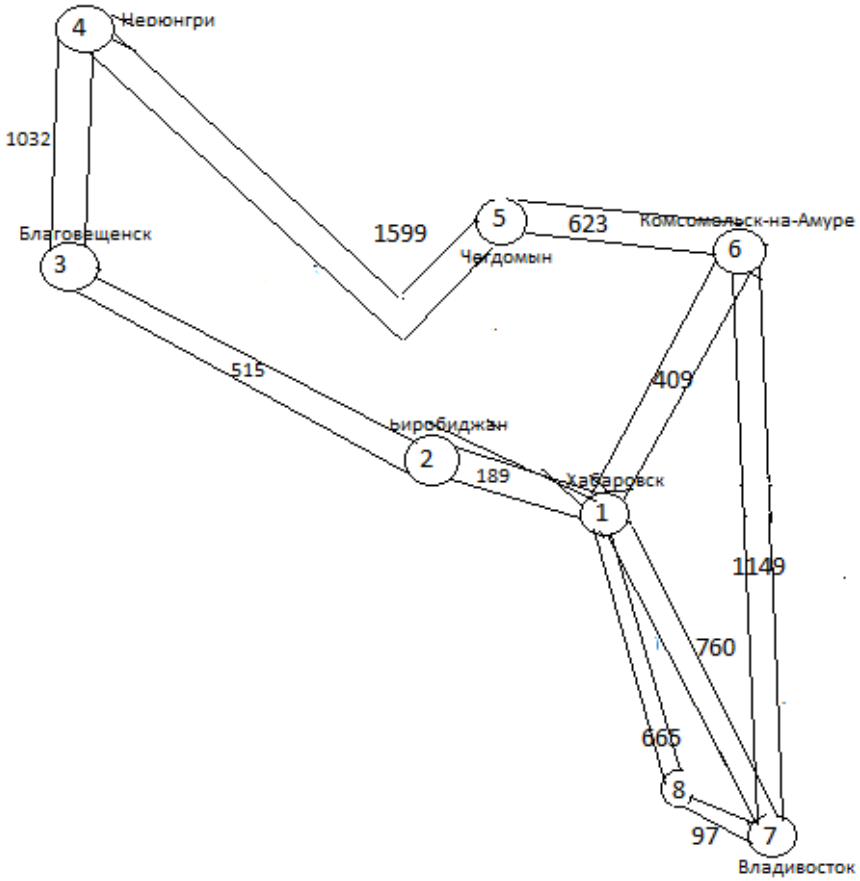

Рисунок 15 – Сеть маршрутов

Рисунок 15 показывает визуально, как расположены города, для которых мы будем строить оптимальный маршрут. Для пояснения почему на Рисунке 15 из г. Хабаровск можно попасть в г. Уссурийск отдельно от маршрута в г. Владивосток, объясню, действительно в г. Уссурийск можно попасть только по дороге из г. Хабаровск в г. Владивосток, но следуя данному маршруту мы просто проезжаем г. Уссурийск не заезжая в него, следовательно, мы его не посещаем, а это значит чтобы попасть в г. Уссурийск нам необходимо свернуть с маршрута г. Хабаровск- г. Владивосток., что позволяет нам считать - это отдельным маршрутом. Далее вводим матрицу расстояний.

```
0       189    9999999 9999999 9999999 409     760     665
189     0      515     9999999 9999999 9999999 9999999 9999999
9999999 515    0       1032    9999999 9999999 9999999 9999999
9999999 9999999 9999999   0    1599    9999999 9999999 9999999
9999999 9999999 9999999 1599    0       623     9999999 9999999
409     9999999 9999999 9999999 623     0       1149    9999999
760     9999999 9999999 9999999 9999999 1149     0      97
665     9999999 9999999 9999999 9999999 9999999 97      0
```

Рисунок 16 – Матрица расстояний

```
Distance is: 5869
Optimal trace:
     1     2     3     4     5     6     7     8     1
```

Рисунок 17 – Итоговый результат повторного эксперимента

Как видно муравьиный алгоритм вычисляет оптимальный путь, но у данного алгоритма есть недостаток. Он всегда выдаёт приближённое значение оптимального пути, то есть не факт, что при первом эксперименте нам алгоритм выдаст самый оптимальный путь. Так например, при повторном эксперименте первого опыта, будет выдаваться уже другой маршрут и его длина. На Рисунке 18, для сравнения показаны вместе итоговые результаты первого эксперимента.

```
Distance is: 6
Optimal trace:
     4     3     2     1     9    10     8     6     7     5     4

Distance is: 6.3
Optimal trace:
     9     7     6     8    10     5     4     3     2     1     9
```

Рисунок 18 – Итоговый результат повторно проведённого первого эксперимента

Таким образом видно, что недостаток алгоритма-это приближённость данных. Но к плюсам можно отнести его быстродействие. Иногда этот фактор является основным в работе, и даже минимальная погрешность не столь важна.

В статье было рассмотрено решение задачи коммивояжёра, в которой требовалось найти оптимальный путь с помощью муравьиного алгоритма в программной среде MatLab. Были проведены несколько экспериментов и рассмотрены данные приближённые к реальным. Муравьиные алгоритмы важны в решении множества оптимизационных задач, без которых оптимального решения невозможно было бы составить, кроме как использования других методов оптимизации для решения подобного рода задач.

Библиографический список

1. Дочкин А.С. Использование муравьиного алгоритма поиска кратчайшего пути в решении задачи коммивояжера // научные исследования и разработки в эпоху глобализации. Уфа: общество с ограниченной ответственностью "Аэтерна", 2017. С. 33-35.
2. Семенов С.С., Ткачев Д.Ф., Педан А.В., Батаев М.Р., Алисевич Е.А., Гусев А.П., Шарапкалиев Е.Д., Киселев Д.В. Программа для решения задачи коммивояжёра с помощью муравьиного алгоритма // Хроники объединенного фонда электронных ресурсов наука и образование . 2016. №6 (85) .С.31.

3. Борознов В.О Исследование решения задачи коммивояжера // Вестник астраханского государственного технического университета. Серия: управление, вычислительная техника и информатика . 2009. №2. С.147-151.
4. Штовба С.Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. 2003. №4. С. 70–75
5. Попов Г.А., Скворцов С.В., Параллельное решение задачи коммивояжера муравьиным алгоритмом // новые информационные технологии в научных исследованиях. Рязань: Рязанский государственный радиотехнический университет , 2017. С. 157-158.
6. Муравьиные алгоритмы // allbest URL: https://otherreferats.allbest.ru/programming/00039144_0.html (дата обращения: 02.01.2018).
7. Муравьиные алгоритмы // rain.ifmo URL: http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2006/article.pdf (дата обращения: 02.01.2018).
8. Алгоритм муравья для решения задачи коммивояжера // pandia URL: http://pandia.ru/text/80/261/70366.php (дата обращения: 02.01.2018).
9. I.D.Ariyasingha,T.G.I.Fernando Swarm and Evolutionary Computation // Swarm and Evolutionary Computation. August 2015.C. Volume 2, Pages1-64.
10. H.Eldem, E.Ülker The application of ant colony optimization in the solution of 3D traveling salesman problem on a sphere // Engineering Science and Technology, an International Journal. 2017. Т.20. № 4. С. 1242-1248.
11. Муравьиные алгоритмы. URL:http://www.machinelearning.ru/wiki/index.php?title=Муравьиные_алгоритмы (дата обращения 02.01.2018)
12. Муравьиные алгоритмы URL:https://vk.com/ant_colony_optimization (дата обращения 30.12.2017).
13. Google карты URL:https://www.google.ru/maps/dir///@49.6015836,126.8347755,6z/data=!4m2!4m1!3e0 (дата обращения 10.01.2018).