|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт искусственного интеллекта | | |
| Кафедра программного обеспечения систем радиоэлектронной аппаратуры | | |

|  |  |
| --- | --- |
| **ОТЧЕТ** | |
| **ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ** | |
|  | |
| **Тема: «Навигация по полю точечных ориентиров»** | |
| Студент группы           КМБО-02-21 | В. А. Бредихин |
| Руководитель практики | А.П. Кирсанов |

|  |  |  |
| --- | --- | --- |
| «Отчет представлен к рассмотрению» | «\_\_\_» \_\_\_\_\_\_\_\_\_\_2024 г. | *(подпись студента)* |
|  |  |  |
| «Отчет утвержден.  Допущен к защите» | «\_\_\_» \_\_\_\_\_\_\_\_\_\_2024 г. | *(подпись руководителя)* |

Москва 2024

**Оглавление**

[**Введение** 2](#_Toc168580918)

[1. Формулировка задачи 2](#_Toc168580919)

[2. Пояснение по задаче 2](#_Toc168580920)

[3. Плюсы и минусы метода 4](#_Toc168580921)

[**Описание алгоритма** 5](#_Toc168580922)

[**Шаги моделирования** 6](#_Toc168580923)

[**Реализованные функции** 7](#_Toc168580924)

[**Статистический анализ алгоритма** 9](#_Toc168580925)

[1. Анализ зависимости работоспособности алгоритма от радиуса измерений. 9](#_Toc168580926)

[2. Анализ зависимости работоспособности алгоритма от количества ТО. 10](#_Toc168580927)

[**Заключение** 11](#_Toc168580928)

# Введение

1. Формулировка задачи

Написание библиотеки С++ для расчета карт расстояний

1. Пояснение по задаче

Допустим, что дана некоторая карта поля точечных ориентиров(ТО), которая представляет собой набор точек на плоскости. Пример такой карты изображен на рисунке 1.

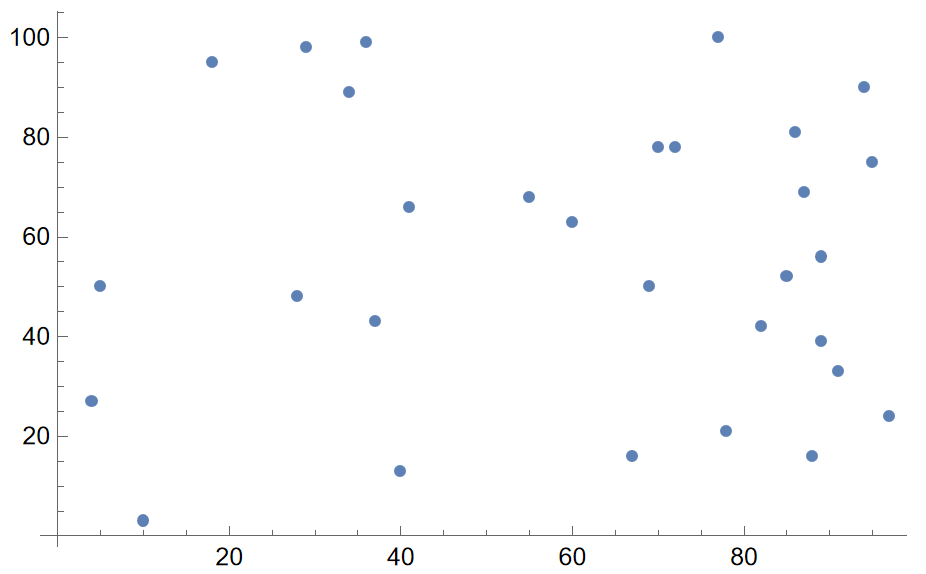


Рисунок 1

На летательном аппарате находятся датчики поля, которые способны обнаруживать точечные ориентиры и считывать расстояния между ними в определенном радиусе вокруг ЛА. Полученную информацию с датчиков можно представлять в виде набора точек с заданными попарными расстояниями. Пример такого набора изображен на рисунке 2.

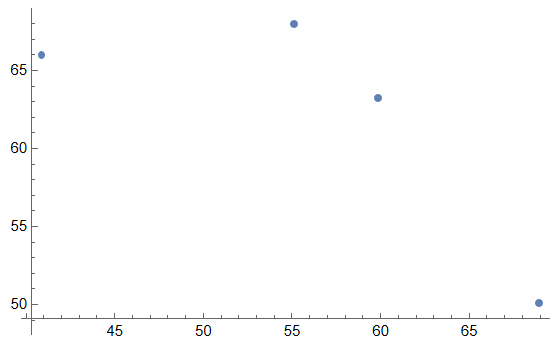


Рисунок 2

Понятно, что набор точек только с заданными попарными расстояниями можно расположить на плоскости бесконечным количеством способов. Однако при условии, что ЛА находится в области представляемой картой и ошибка измерения не превышает предельного значения, такая конфигурация точек будет однозначно соответствовать некоторому участку карты. Пример такого наложения изображен на рисунке 3.

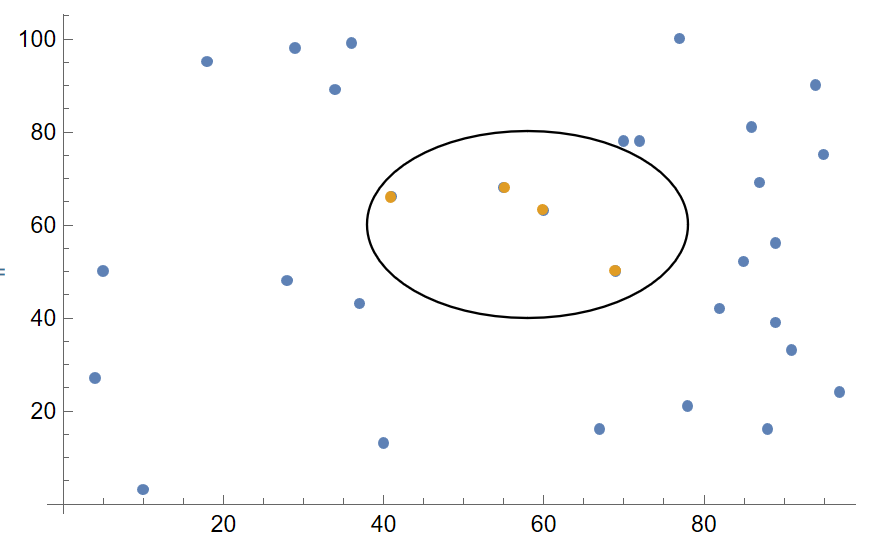


Рисунок 3

Таким образом навигация в таком поле сводится к задаче поиска определенной конфигурации точек на карте.

1. Плюсы и минусы метода

|  |  |
| --- | --- |
| + | - |
| Малое количество необходимой картографической информации | Метод не применим на регулярных картах. Пример регулярной карты изображён на рисунке 4 |
|  | Необходимость одновременного наблюдения как минимум 3 ориентиров для однозначного определения местоположения |
|  | Экспоненциальная сложность алгоритмов нахождения максимальной клики, применяемых в методе |

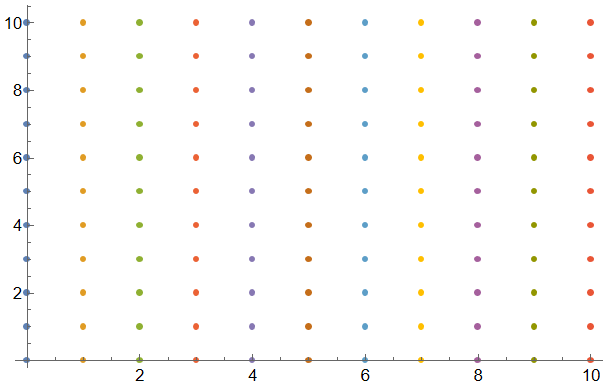


Рисунок 4

**Описание алгоритма**

Исходные данные алгоритма

Эталонная карта (далее просто карта) поля точечных ориентиров и измерения расстояний между ориентирами в наблюдаемом датчиками фрагменте поля и их меток.

Выходные данные алгоритма

Предполагаемые координаты летательного аппарата.

Суть алгоритма заключается в рассмотрении карты и измерений в качестве полных взвешенных графов, где вершины – это точки или ориентиры, а веса ребер – это расстояния между точками, которым соответствуют вершины, между которыми проведено ребро. Тогда задача навигации сводится к задаче поиска максимального пересечения двух графов. Задачу поиска пересечения можно, в свою очередь, свести к задаче поиска максимальной клики в специальном графе построенном на основе двух данных, называемом модульным произведением. Модульным произведением двух взвешенных графов с метками G и H называется граф M, у которого вершинами являются пары вершин , где , у которых совпадают метки в графах G и H соответственно, а ребрами соединены только те вершины и , у которых веса ребер и равны. Имеет смысл рассматривать модульное произведение, в котором условие равенство весов соответствующих ребер заменено на условие отличия весов не более чем на заданную константу. Далее такое модульное произведение называется модульное произведение с параметром, где параметр и задает константу, фигурирующую в определении.

**Шаги моделирования**

Шаг 1

Сгенерировать набор точек с координатами, задаваемыми случайной равномерно распределенной на отрезке [a, b] величиной.

Шаг 2

Сгенерировать полный взвешенный граф с помеченными вершинами по карте поля ТО, где в роли вершин будут выступать ТО, в роли весов на рёбрах - расстояние между ориентирами, которые соответствуют вершинам, в роли меток на вершинах – метки на соответствующих ориентирах.

Шаг 3

Задать координаты местоположения ЛА относительно карты

Шаг 4

Сгенерировать измерения в виде точек на карте. Каждой точке будет соответствовать некоторый ориентир, который находится в определенном радиусе от ЛА и имеет координаты, которые отличаются от координат точки по каждой координате на некоторую случайную величину, распределенную по нормальному распределению c нулевым математическим ожиданием и задаваемым среднеквадратичным отклонением, называемой далее ошибкой измерения. Метка точки измерения с определенной вероятностью может отличаться от метки ориентира, который ей соответствует.

Шаг 5

Провести шаг 2, где карта заменена на точки измерения из шага 4.

Шаг 6

Задать ε и найти модульное произведение графов из шага 2 и 5 с параметром ε.

Шаг 7

Найти в модульном произведении из предыдущего шага максимальную клику.

Шаг 8

Выбрать 3 случайные точки из клики.

Шаг 9

Найти пересечение 3 окружностей, центрами которых выступают точки из предыдущего шага, а радиусами – расстояния от точек до местоположения ЛА.

Шаг 10

Пересечение из шага 9 и будет точкой предполагаемого местоположения ЛА.

**Реализованные функции**

Для реализации шагов моделирования были написаны следующие функции на языке Wolfram:

GenerateMap[range\_, numberOfPoints\_] – функция генерации карты. Функция принимает в качестве аргументов отрезок range и неотрицательное целое число numberOfPoints и выдает список из пар чисел из отрезка range в количестве numberOfPoints. Числа в парах генерируются по равномерному на отрезке range закону.

WeightedDistanceGraph[vert\_] – функция генерации взвешенного графа расстояний. Функция принимает на вход список из точек vert и выдает полный взвешенный граф, где вершинами выступают точки, а весами рёбер выступают расстояния между точками, которым соответствуют вершины, между которыми проведено ребро.

GenerateMapGraph[map\_, weights\_] – функция генерации графа по полю ТО. Функция принимает в качестве аргументов список из пар чисел map и список weights из меток и выдает граф аналогичный графу, который бы получился при вызове функции WeightedDistanceGraph от списка map, но где все вершины будут помечены в соответствии со списком weights.

TakeMeasures[curPosition\_,radius\_, map\_, mapGraph\_, σ\_, markErrorChance\_] – функция генерации измерений. Функция принимает в качестве аргументов точку curPosition, число radius, список из точек map, граф mapGraph, неотрицательное число σ, неотрицательное число markErrorChance. Генерирует измерения в соответствии с шагом 4, где местоположение ЛА – curPosition, радиус генерации точек – radius, карта – map, среднеквадратичное отклонение ошибки измерения – σ, вероятность изменения метки ориентира markErrorChance. На выходе дает пару: список из точек измерения и список из меток измерения.

ModularProduct[graph1\_, graph2\_] – функция генерации модульного произведения графов. Функция принимает на вход два графа graph1 и graph2, которые соответствуют графам, которые описаны соответственно в шагах 1 и 5, и возвращает модульное произведение графов graph1 и graph2.

FindIntersection[curPosition\_, radius\_, map\_, mapGraph\_, σ\_, MarkErrorChance\_] – функция поиска пересечения графов, графа расстояний карты и графа расстояний измерений. Функция принимает в качестве аргументов точку curPosition, число radius, список из точек map, граф mapGraph, неотрицательное число σ, неотрицательное число markErrorChance. Функция генерирует измерения с помощью функции TakeMeasures в радиусе radius от точки curPosition и граф расстояний по этим измерениям MeasuresMapGraph. Далее находит модульное произведение графа mapGraph и графа MeasuresMapGraph и максимальную клику в нем. Возвращает список точек карты, которым соответствуют вершины максимальной клики.

FindLocation[curPosition\_, radius\_, map\_, mapGraph\_, σ\_, MarkErrorChance\_] – функция поиска предполагаемого местоположения ЛА. Функция принимает в качестве аргументов точку curPosition, число radius, список из точек map, граф mapGraph, неотрицательное число σ, неотрицательное число markErrorChance. Функция с помощью функции FindIntersection генерирует список точек. Если точек больше 2, то функция выбирает из списка 2 случайные точки, если их ровно 2, и 3 случайные точки в остальных случаях, и возвращает пересечение окружностей, центрами которых выступают ранее выбранные точки, а радиусами – расстояния от этих точек до точки curPosition. Если точек меньше чем 2, то функция возвращает 0.

Пример работы некоторых из вышеперечисленных функций изображен на рисунке 5



Рисунок 5

**Статистический анализ алгоритма**

1. Анализ зависимости работоспособности алгоритма от радиуса измерений.

Анализ проводился при следующих параметрах:

- карта состояла из 30 ТО, которые находились в квадратe

[0, 100]×[0, 100];

- ошибка измерения имела среднеквадратичное отклонение равное 0.1;

- вероятность ошибки измерения метки отсутствовала

- параметр модульного произведения равнялся 0.5

Была сгенерирована карта согласно шагу 1 и равномерно в области карты было сгенерировано 100 точек, задающих различные потенциальные расположения ЛА, и проведена процедура вычисления предполагаемого местоположения ЛА согласно алгоритму для различных радиусов измерения. Результаты приведены в виде графика на рисунке 5. На графике по оси абсцисс отложены радиусы измерения, по оси ординат – количество точек, для которых предполагаемое местоположение ЛА совпало с реальным.

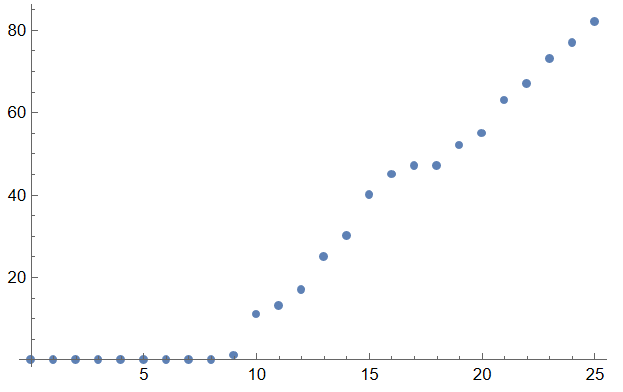


Рисунок 5

1. Анализ зависимости работоспособности алгоритма от количества ТО.

Анализ проводился при следующих параметрах:

- карта состояла из переменного количества ТО, которые находились в квадратe

[0, 100]×[0, 100];

- ошибка измерения имела среднеквадратичное отклонение равное 0.1;

- вероятность ошибки измерения метки отсутствовала

- параметр модульного произведения равнялся 0.5

- радиус измерения равнялся 20

Было сгенерировано 26 карт с различным количество ТО и равномерно в области карты было сгенерировано 100 точек, задающих различные потенциальные расположения ЛА, и проведена процедура вычисления предполагаемого местоположения ЛА согласно алгоритму для различного количества ТО на карте. Результаты приведены в виде графика на рисунке 6. На графике по оси абсцисс отложено количество ТО на карте, по оси ординат – количество точек, для которых предполагаемое местоположение ЛА совпало с реальным.

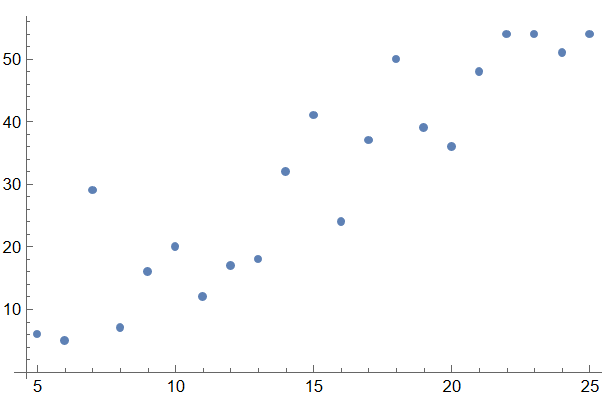


Рисунок 6

**Заключение**

В итоге выполнения работы была решена задача реализации алгоритма навигации по полю точечных ориентиров и задача моделирования работы алгоритма. Были получены определенные результаты в области работоспособности алгоритма в зависимости от различных параметров. Также были получены определенные навыки написания программ на языке Wolfram.

**Приложение**

Исходный код функций

(\*Основные функции\*)

ModularProduct[graph1\_, graph2\_] := Module[{ModProductVertexList, GraphMatrix1, GraphMatrix2, ModProduct, ModProductEdgesList, check},

ModProductVertexList = Distribute[{VertexList[graph1], VertexList[graph2]}, List];

ModProductVertexList = Select[ModProductVertexList,

AnnotationValue[{graph1, #[[1]]}, VertexWeight] === AnnotationValue[{graph2, #[[2]]}, VertexWeight]&];

GraphMatrix1 = WeightedAdjacencyMatrix[graph1];

GraphMatrix2 = WeightedAdjacencyMatrix[graph2];

ModProduct = AdjacencyGraph[ModProductVertexList,

AdjacencyMatrix[CompleteGraph[Length[ModProductVertexList]]]];

ModProductEdgesList = EdgeRules[ModProduct];

check := ((#[[1, 1]] == #[[2, 1]]) || (#[[1, 2]] == #[[2, 2]]))&;

ModProductEdgesList = Select[ModProductEdgesList,

If[check[#], False, (Abs[AnnotationValue[{graph1, #[[1, 1]]\[UndirectedEdge]#[[2, 1]]}, EdgeWeight] - AnnotationValue[{graph2, #[[1, 2]]\[UndirectedEdge]#[[2, 2]]}, EdgeWeight]] <= 0.5)]&];

ModProduct = UndirectedGraph[EdgeRules[ModProductEdgesList]];

Return[ModProduct]]

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

WeightedDistanceGraph[vert\_] := Graph[

Flatten[

Table[

Annotation[UndirectedEdge @@ {vert[[a]], vert[[b]]}, EdgeWeight -> EuclideanDistance @@ vert[[{a, b}]]],

{a, 1, Length[vert] - 1}, {b, a + 1, Length[vert]}], 1], VertexCoordinates -> vert]

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

TakeMeasures[curPosition\_,radius\_, map\_, mapGraph\_, \[Sigma]\_, markErrorChance\_] := Module[{localMap, localWeights, MeasureVertexWeights, Measures},

localMap = Select[map, (EuclideanDistance[curPosition, #] <= radius)&];

localWeights = Map[AnnotationValue[{mapGraph, #}, VertexWeight]&, localMap];

Measures = Map[{RandomVariate[NormalDistribution[#[[1]], \[Sigma] ]], RandomVariate[NormalDistribution[#[[2]], \[Sigma] ]]}&, localMap];

MeasureVertexWeights = Map[If[RandomVariate[BinomialDistribution[1, 1 - markErrorChance]] == 1,

#, RandomChoice[VertexWeightings]]&,

localWeights];

Return[{Measures, localWeights}]]

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

GenerateMap[range\_, numberOfPoints\_] := Module[{MapList},

MapList = RandomInteger[range, {numberOfPoints, 2}];

Return[MapList]]

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

GenerateMapGraph[map\_, weights\_]:= Module[{graph},

graph = WeightedDistanceGraph[map];

graph = Graph[graph, VertexWeight->weights, VertexLabels->"VertexWeight", EdgeLabels->"EdgeWeight"];

Return[graph]]

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

FindLocation[curPosition\_, radius\_, map\_, mapGraph\_, \[Sigma]\_, MarkErrorChance\_] := Module[{Measures, MeasuresMapGraph, ModProduct, intersection, location},

Measures = TakeMeasures[curPosition, radius, map,mapGraph, \[Sigma], MarkErrorChance];

If[Length[Measures[[1]]]<3, Return[0]];

MeasuresMapGraph = GenerateMapGraph[Measures[[1]], Measures[[2]]];

ModProduct = ModularProduct[mapGraph, MeasuresMapGraph];

intersection = Map[Part[#, 1]&, FindClique[ModProduct][[1]]];

location = Intersection[ResourceFunction["CircleIntersection"][{intersection[[1]], EuclideanDistance[curPosition, intersection[[1]]]},

{intersection[[2]], EuclideanDistance[curPosition, intersection[[2]]]}],

ResourceFunction["CircleIntersection"][{intersection[[1]], EuclideanDistance[curPosition, intersection[[1]]]},

{intersection[[3]], EuclideanDistance[curPosition, intersection[[3]]]}]];

Return[location]]

(\*\*\*\*\*\*\*\*\*\*\*\*)

FindIntersection[curPosition\_, radius\_, map\_, mapGraph\_, \[Sigma]\_, MarkErrorChance\_] := Module[{Measures, MeasuresMapGraph, ModProduct, intersection},

Measures = TakeMeasures[curPosition, radius, map,mapGraph, \[Sigma], MarkErrorChance];

MeasuresMapGraph = GenerateMapGraph[Measures[[1]], Measures[[2]]];

ModProduct = ModularProduct[mapGraph, MeasuresMapGraph];

intersection = Map[Part[#, 1]&, FindClique[ModProduct][[1]]];

Return[intersection]]