



Министерство науки и высшего образования 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 № 5  
по курсу «Анализ алгоритмов»  
на тему: «Многопоточные вычисления»

Студент ИУ7-52Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

А. П. Лемешев  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Ю. В. Строганов  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Л. Л. Волкова  
(И. О. Фамилия)

Москва — 2022 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Цели и задачи	4
1.2 Конвейер	4
1.3 Алгоритм кластеризации DBSCAN	4
<b>2 Конструкторская часть</b>	<b>7</b>
Разработка алгоритмов	7
<b>3 Технологическая часть</b>	<b>9</b>
3.1 Требования к программному обеспечению	9
3.2 Средства реализации	9
3.3 Реализация алгоритма	9
3.4 Тестовые данные	12
<b>4 Исследовательская часть</b>	<b>13</b>
4.1 Технические характеристики	13
4.2 Время выполнения алгоритма	13
<b>ЗАКЛЮЧЕНИЕ</b>	<b>15</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>16</b>

## ВВЕДЕНИЕ

Конвейеризация — архитектурный прием, используемый в современных процессорах с целью повышения быстродействия. Идея заключается в разделении обработки компьютерной команды на последовательность независимых стадий с сохранением результатов в конце каждой стадии. Обычно для выполнения каждой инструкции требуется осуществить некоторое количество однотипных операций, например, получение инструкции, декодирование инструкции и т.п. Каждую из этих операций сопоставляют одной ступени конвейера. Некоторые современные процессоры имеют более 30 ступеней в конвейере, что повышает производительность процессора, но, однако, приводит к увеличению длительности простоя (например, в случае ошибки в предсказании условного перехода). Не существует единого мнения по поводу оптимальной длины конвейера: различные программы могут иметь различные требования [1].

Кластеризация — объединение в группы схожих объектов является одной из фундаментальных задач в области анализа данных. Список прикладных областей, где она применяется, широк: сегментация изображений, маркетинг, борьба с мошенничеством, прогнозирование, анализ текстов и многие другие [2].

В рамках данной лабораторной работы конвейерная обработка данных будет исследоваться на алгоритме кластеризации DBSCAN.

# **1 Аналитическая часть**

## **1.1 Цели и задачи**

Цель работы: получить навыки организации асинхронного взаимодействия потоков на примере реализации алгоритма DBSCAN.

Задачи лабораторной работы:

- 1) изучить идею конвейерной обработки данных.
- 2) реализовать алгоритм DBSCAN с использованием конвейерной обработки данных;
- 3) провести сравнительный анализ времени работы алгоритма для различного количества используемых потоков;
- 4) обосновать полученные результаты в отчете.

## **1.2 Конвейер**

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности [1].

Конвейерная обработка данных основана на разделении получаемого трафика от контроллера (управляющего устройства) и параллельном выполнении обработки получаемой информации. Стоит отметить, что конвейеры работают параллельно, однако они могут быть связаны между собой.

## **1.3 Алгоритм кластеризации DBSCAN**

Алгоритм DBSCAN, плотностный алгоритм для кластеризации пространственных данных с присутствием шума, был предложен Мартином Эстер, Гансом-Питером Кригель и коллегами в 1996 году как решение проблемы разбиения данных на кластеры произвольной формы. Большинство алгоритмов, производящих плоское разбиение, создают кластеры по форме близкие к сферическим, так как минимизируют расстояние документов до центра кластера. Авторы

DBSCAN экспериментально показали, что их алгоритм способен распознать кластеры различной формы, например, как на рисунке 1.1 [3, с. 197].

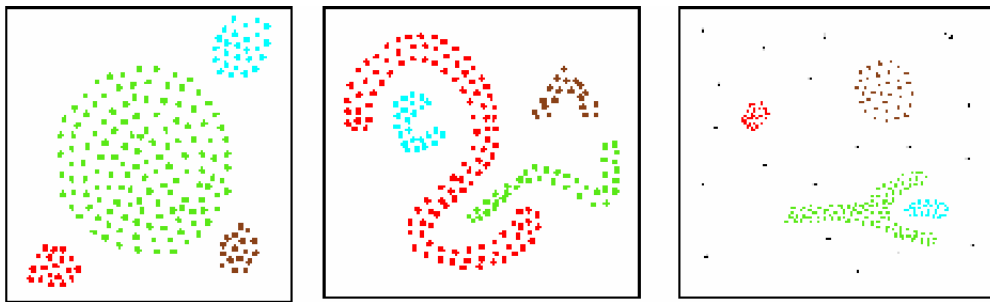


Рисунок 1.1 – Примеры кластеров произвольной формы

Идея, положенная в основу алгоритма, заключается в том, что внутри каждого кластера наблюдается типичная плотность точек, которая заметно выше, чем плотность снаружи кластера, а также плотность в областях с шумом ниже плотности любого из кластеров. Ещё точнее, что для каждой точки кластера её соседство заданного радиуса должно содержать не менее некоторого числа точек, это число точек задаётся пороговым значением. Перед изложением алгоритма дадим необходимые определения.

*Eps-соседство точки  $p$* , обозначаемое как  $N_{Eps}(p)$  1.1 определяется как множество документов, находящихся от точки  $p$  на расстояния не более  $Eps$ .

$$N_{Eps}(p) = \{q \in D | dist(p, q) \leq Eps\} \quad (1.1)$$

Поиска точек, чьё  $N_{Eps}(p)$  содержит хотя бы минимальное число точек ( $MinPt$ ) не достаточно, так как точки бывают двух видов: ядровые и граничные.

Точка  $p$  *непосредственно плотно-достижима* из точки  $p$  (при заданных  $Eps$  и  $MinPt$ ), если  $p \in N_{Eps}(p)$  и  $|N_{Eps}(p)| \geq MinPt$ .

Точка  $p$  *плотно-достижима* из точки  $q$  (при заданных  $Eps$  и  $MinPt$ ), если существует последовательность точек 1.2 непосредственно плотно-достижимы из  $p_i$ .

$$q = p_1, p_2, \dots, p_n = p : p_{i+1} \quad (1.2)$$

Это отношение транзитивно, но не симметрично в общем случае, однако симметрично для двух ядровых точек.

Точка  $p$  *плотно-связана* с точкой  $q$  (при заданных  $Eps$  и  $MinPt$ ), если существует точка  $o$ :  $p$  и  $q$  плотно-достижимы из  $o$  (при заданных  $Eps$

и  $MinPt$ ).

*Кластер*  $C_j$  (при заданных  $Eps$  и  $MinPt$ ) — это не пустое подмножество документов, удовлетворяющее следующим условиям:

- 1)  $\forall p, q$  : если  $p \in C_j$  и  $q$  плотно-достижима из  $p$  (при заданных  $Eps$  и  $MinPt$ ), то;  $q \in C_j$ ;
- 2)  $\forall p, q$  :  $p$  плотно-связана с  $q$  (при заданных  $Eps$  и  $MinPt$ ).

*Шум* — это подмножество документов, которые не принадлежат ни одному кластеру 1.3.

$$\{p \in D | \forall j : p \notin C_j, j = \overline{1, |C|}\} \quad (1.3)$$

Алгоритм DBSCAN для заданных значений параметров  $Eps$  и  $MinPt$  исследует кластер следующим образом: сначала выбирает случайную точку, являющуюся ядровой, в качестве затравки, затем помещает в кластер саму затравку и все точки, плотно-достижимые из неё.

## Вывод

В данном разделе были рассмотрены понятия конвейерной обработки данных и алгоритм DBSCAN.

## 2 Конструкторская часть

### Разработка алгоритмов

На рисунке 2.1 показана схема алгоритма DBSCAN.

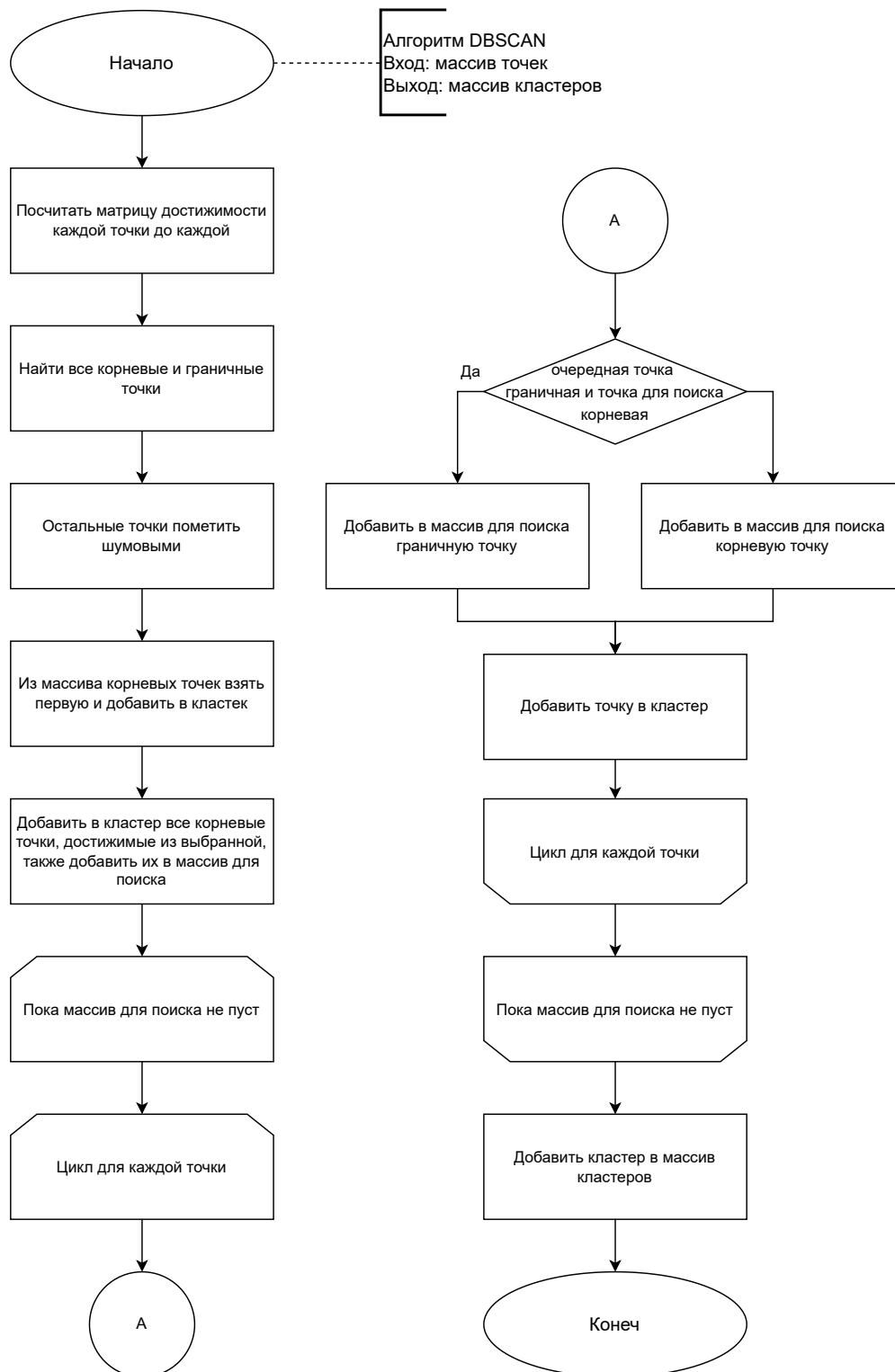


Рисунок 2.1 – Схема алгоритма DBSCAN

## Вывод

На основе формул и теоретических данных, полученных в аналитическом разделе, была спроектирована схема алгоритма DBSCAN.



## 3 Технологическая часть

### 3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

- программа получает на вход файл с точками, минимальное количество точек в кластере, минимальное расстояние и количество потоков;
- программа выдает в результате файл с кластеризованными точками;
- алгоритм реализован с помощью конвейерной обработки данных;
- программа должна измерять реальное время.

### 3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран *C++* ввиду следующих причин:

- мое желанием расширить свои знания в области применения данного языка;
- возможность измерять реальное время выполнения алгоритма;
- данный язык предоставляет широкие возможности для работы с потоками.

Таким образом, с помощью языка программирования *C++* можно реализовать программное обеспечение, которое соответствует перечисленным выше требованиям.

### 3.3 Реализация алгоритма

В листинге 3.1 приведена реализация алгоритма DBSCAN. В листинге 3.2 приведена функция кластеризации, используемая в внешнем цикле DBSCAN.

### Листинг 3.1 – Реализация алгоритма DBSCAN

```

1 void DBSCAN::dbscan()
2 {
3     vector<int> tmp;
4     prl_calc_mtx(0, m_points.size());
5     prl_calc_kernel(0, m_points.size(), tmp);
6     for (int &index: tmp)
7         erase_vector(unclassified, index);
8     tmp.clear();
9     prl_calc_border(0, unclassified.size(), tmp);
10    for (int &index: tmp)
11        erase_vector(unclassified, index);
12    tmp.clear();
13    prl_calc_noise(0, unclassified.size(), tmp);
14    for (int &index: tmp)
15        erase_vector(unclassified, index);
16    prl_unclustered_init(0, kernel.size());
17    while (unclustered.size() != 0)
18    {
19        vector<int> cluster;
20        vector<Check> checker;
21        vector<Check> tmp_checker;
22        int index = unclustered[0];
23        cluster.push_back(index);
24        prl_clustering_init(0, m_points.size(), index,
25                           cluster, checker);
26        while (checker.size() != 0)
27        {
28            prl_clustering_step(0, checker.size(), cluster,
29                               checker, tmp_checker);
30            checker.swap(tmp_checker);
31            tmp_checker.clear();
32        }
33        for (int &index: cluster)
34            erase_vector(unclustered, index);
35        clusters.push_back(cluster);
36    }
37 }

```

### Листинг 3.2 – Функция кластеризации

```
1 void DBSCAN::prl_clustering_step(  
2     int start, int end,  
3     vector<int> &cluster,  
4     vector<Check> &checker,  
5     vector<Check> &tmp_checker  
6 )  
7 {  
8     for (int i = 0; i < checker.size(); i++)  
9         for (int j = 0; j < m_points.size(); j++)  
10            {  
11                bool is_reachable = mtx[checker[i].index][j];  
12                bool in_cluster = find(  
13                    cluster.begin(), cluster.end(), j  
14                ) != cluster.end();  
15                bool is_border = find(  
16                    border.begin(), border.end(), j  
17                ) != border.end();  
18                bool is_kernel = find(  
19                    kernel.begin(), kernel.end(), j  
20                ) != kernel.end();  
21                if (is_reachable && !in_cluster)  
22                {  
23                    if (is_border && checker[i].is_kernel)  
24                        tmp_checker.push_back(Check{j, false});  
25                    else  
26                        tmp_checker.push_back(Check{j, true});  
27                    cluster.push_back(j);  
28                }  
29            }  
30 }
```

### 3.4 Тестовые данные

В таблице 3.1 приведены тесты для функции, реализующей алгоритм DBSCAN. Тесты выполнялись по методологии черного ящика. Все тесты пройдены успешно.

Таблица 3.1 – Тестовые данные

Точки	MinPt	Eps	Количество кластеров
1 1 1 2 2 1	3	0.5	1
1 1 1 2 2 1	3	1.5	0
1 1 1 2 2 1 5 5 5 6 6 5 10 10	3	1	2

### Вывод

На основе схемы, полученной в конструкторском разделе, был реализован и протестирован алгоритм DBSCAN.

## 4 Исследовательская часть

### 4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками.

1. Операционная система: Ubuntu Linux 64-bit.
2. Память: 8 ГБ.
3. Процессор: AMD Ryzen 5 3550H.

### 4.2 Время выполнения алгоритма

Время работы функции замерено с помощью ассемблерной инструкции **rdtsc**, которая читает счетчик Time Stamp Counter и возвращает 64-битное количество тиков с момента последнего сброса процессора.

Результаты тестирования приведены в таблице 4.1. На рисунке 4.1 приведена зависимость времени работы алгоритма от количества точек.

Таблица 4.1 – Время работы алгоритма

Количество точек	Время работы, тики		
	1 поток	2 потока	4 потока
10	253612	3127633	3458776
25	2642367	4742276	4029533
50	6417121	7537831	4957453
75	20587893	13592053	8238764
100	38474381	23279471	13287412

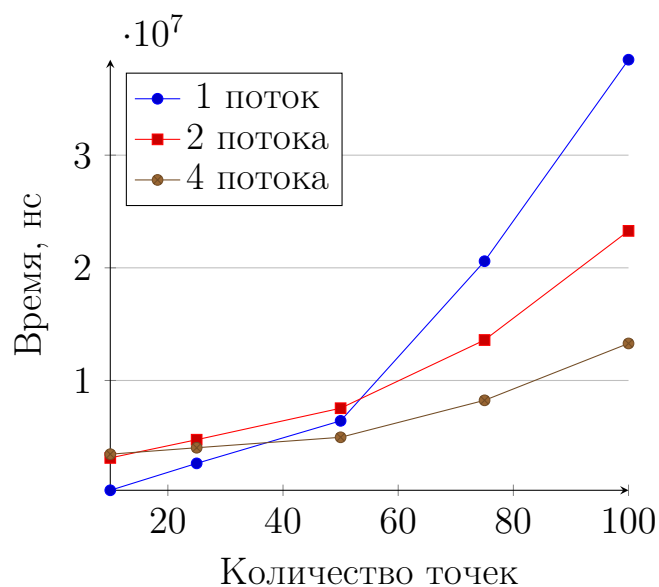


Рисунок 4.1 – Зависимость времени выполнения от количества потоков

## Вывод

Согласно полученным при проведении эксперимента данным, при малом количестве входных данных, последовательно работающий алгоритм работает быстрее, чем многопоточные. Это объясняется тем, что на выделение поток и их синхронизацию тратиться значительное количество ресурсов. С увеличением входных данных, многопоточные начинают обгонять последовательный алгоритм, поскольку время, затраченное на выделение потоков и синхронизацию, становится незначительным, по сравнению со временем, затраченным на вычисления.

## ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была достигнута поставленная цель: было изучено понятие конвейерной обработки данных и с помощью организации асинхронного взаимодействия потоков реализован алгоритм DBSCAN.

Решены все поставленные задачи:

- 1) изучена идея конвейерной обработки данных.
- 2) реализован алгоритм DBSCAN с использованием конвейерной обработки данных;
- 3) проведен сравнительный анализ времени работы алгоритма для различного количества используемых потоков;
- 4) обоснован полученный результат в отчете.

Исходя из результатов, полученных в исследовательской части, при большом количестве данных, наиболее эффективной является многопоточная реализация алгоритма DBSCAN. При этом, если требуется обработать небольшое количество данных, лучше использовать последовательную реализацию.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конвейер [Электронный ресурс]. — Режим доступа: <https://dic.academic.ru/dic.nsf/ruwiki/1309321> (дата обращения 06.12.2022).
2. *Ершов К., Романова Т.* Анализ и классификация алгоритмов кластеризации // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 274—279.
3. *Большакова Е. И., Клышинский Э. С.* Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. — 2021.