Assignment 2

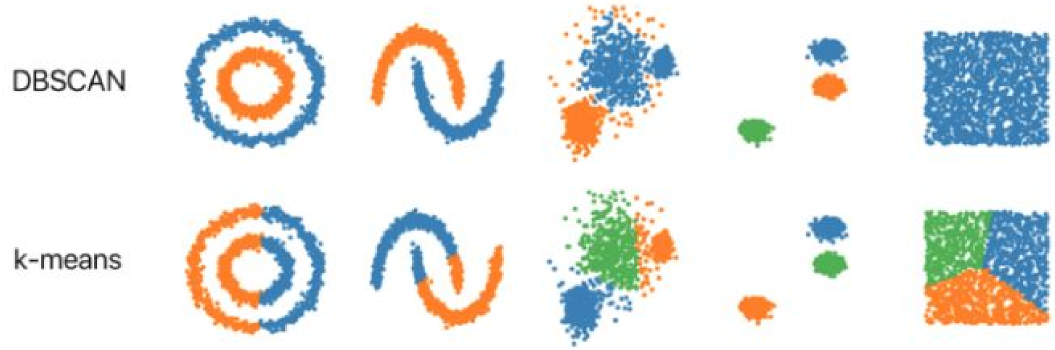
Profiling of Project

Ashish Choudhary
Roll NO: CED18I061

PARALLELISATION OF DBSCAN ALGORITHM : AN ALGORITHM FOR DATA CLUSTERING

What is DBSCAN?

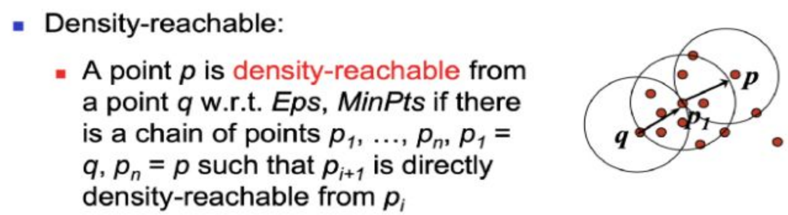
DBSCAN(Density-Based Spatial Clustering of Applications with Noise) is a commonly used unsupervised clustering algorithm proposed in 1996. Unlike the most well known K-mean, DBSCAN does not need to specify the number of clusters. It can automatically detect the number of clusters based on your input data and parameters. More importantly, DBSCAN can find arbitrary shape clusters that k-means are not able to find. For example, a cluster surrounded by a different cluster.



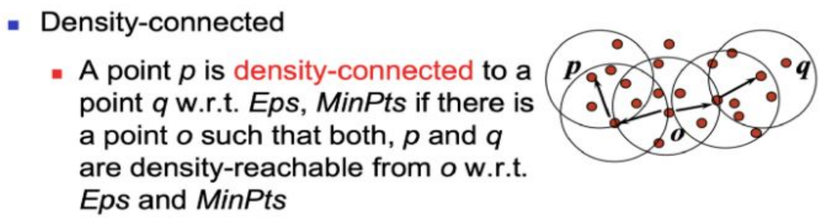
Also, DBSCAN can handle noise and outliers. All the outliers will be identified and marked without been classified into any cluster. Therefore, DBSCAN can also be used for Anomaly Detection (Outlier Detection) Before we take a look at the pseudocode, we need to first understand some basic concepts and terms. Eps, MinPts, Directly density-reachable, density-reachable, density-connected, core point and border point First of all, there are two parameters we need to set for DBSCAN, Eps, and MinPts.

Eps: Maximum radius of the neighborhood
MinPts: Minimum number of points in an Eps-neighbourhood of that point

And there is the concept of Directly density-reachable: A point p is directly density reachable from a point q w.r.t. Eps, MinPts, if NEps (q): {p belongs to D | dist(p,q) ≤ Eps} and |N Eps (q)| ≥ MinPts. Let’s take a look at an example with Minpts = 5, Eps = 1. Let’s take a look at an example to understand density-reachable and density-connected.

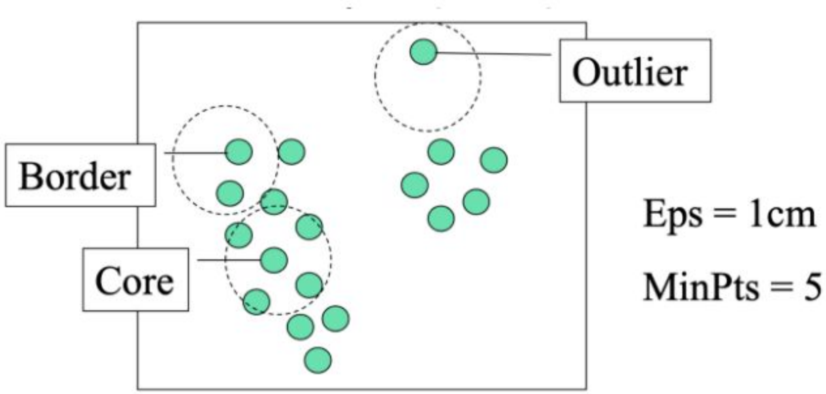


Density-reachable example



Density-connected example

Finally, a point is a core point if it has more than a specified number of points (MinPts) within Eps. These are points that are at the interior of a cluster A. And a border point has fewer than MinPts within Eps, but is in the neighborhood of a core point. We can also define the outlier(noise) point, which is the points that are neither core nor border points.



Core point, Border point, Outlier Point examples

Pseudocode of Serial DBSCAN algo:

```
procedure DBSCAN(X, eps, minpts)
    for each unvisited point x ∈ X do
        mark x as visited
        N ← GetNeighbors(x, eps)
        if |N| < minpts then
            mark x as noise
        else
            C ← {x}
            for each point x′ ∈ N do
                N ← N \ x′
                if x′ is not visited then
                    mark x′ as visited
                    N′ ← GetNeighbors(x′, eps)
                    if |N′| ≥ minpts then
                        N ← N ∪ N′
                if x′ is not yet member of any cluster then
                    C ← C ∪ {x′}
```

Serial Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

double ep;
double pts[1000][50];
int clusters[1000][1000];
int siz[1000];
int minpts,dim,num_pts;
int vis[100000] = {0};

double sqrd_dist(int i,int j)
{
   double sum = 0;
   for(int k = 0; k < dim ; k++)
   {
       sum += pow(pts[i][k] - pts[j][k],2);
   }
   return sqrt(sum);
}


int is_core_node(int i)
```

```
{
   if(siz[i] >= minpts-1 )
       return 1;
   return 0;
}


void get_neighbours()
{
   for(int i = 0; i+1 < num_pts; i++)
   {
       for(int j = i+1; j < num_pts; j++)
       {
           if(i == j)
           continue;
           if(sqrd_dist(i,j) <= ep)
           {
               clusters[i][siz[i]] = j;
               clusters[j][siz[j]] = i;
               siz[i]++;
               siz[j]++;
           }
       }
   }
}
void dfs(int i)
{
   vis[i] = 1;
```

```
    printf("%d ",i+1);
    for(int a  = 0; a < siz[i] ; a++)
    {
        if(vis[clusters[i][a]] != 1)
            dfs(clusters[i][a]);
    }
}
void dbscan()
{
    get_neighbours();

    int cnt = 0;
    for(int i = 0; i < num_pts; i++)
    {
        if(vis[i] != 1 && siz[i] >= minpts)
        {
            cnt++;
            printf("cluster %d : ",cnt);
            dfs(i);
            printf("\n");
        }

    }
    printf("NOISE :");
    for(int i = 0; i < num_pts; i++)
    {
        if(vis[i] != 1)
            printf("%d ",i+1);
```

```c
    }
}



int main()
{
    printf("Enter the ep distance:");
    scanf("%lf",&ep);
    if(ep < 0)
    {
        printf("INVALID EPSILON DISTANCE");
        return 0;
    }
    printf("Enter the minimum points:");
    scanf("%d",&minpts);
    if(minpts < 1)
    {
        printf("INVALID MIN PTS");
        return 0;
    }
    printf("Enter the dimesions of the points:");
    scanf("%d",&dim);
    if(dim < 1)
    {
        printf("INVALID DIMENSIONS");
        return 0;
```

```
    }
    printf("Enter the number of points:");
    scanf("%d",&num_pts);
    if(num_pts < 1)
    {
        printf("INVALID NUMBER OF PTS");
        return 0;
    }
    printf("Enter points:");
    for(int i = 0 ; i < num_pts; i++)
    {
        for(int j = 0; j < dim; j++)
        {
            scanf("%lf",&pts[i][j]);
        }
    }
    dbscan();
    return 0;
}
```

Functional Profiling Results :

Flat Profile:

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00     1378     0.00     0.00  sqrd_dist
  0.00      0.00     0.00        4     0.00     0.00  dfs
  0.00      0.00     0.00        1     0.00     0.00  dbscan
  0.00      0.00     0.00        1     0.00     0.00  get_neighbours
```

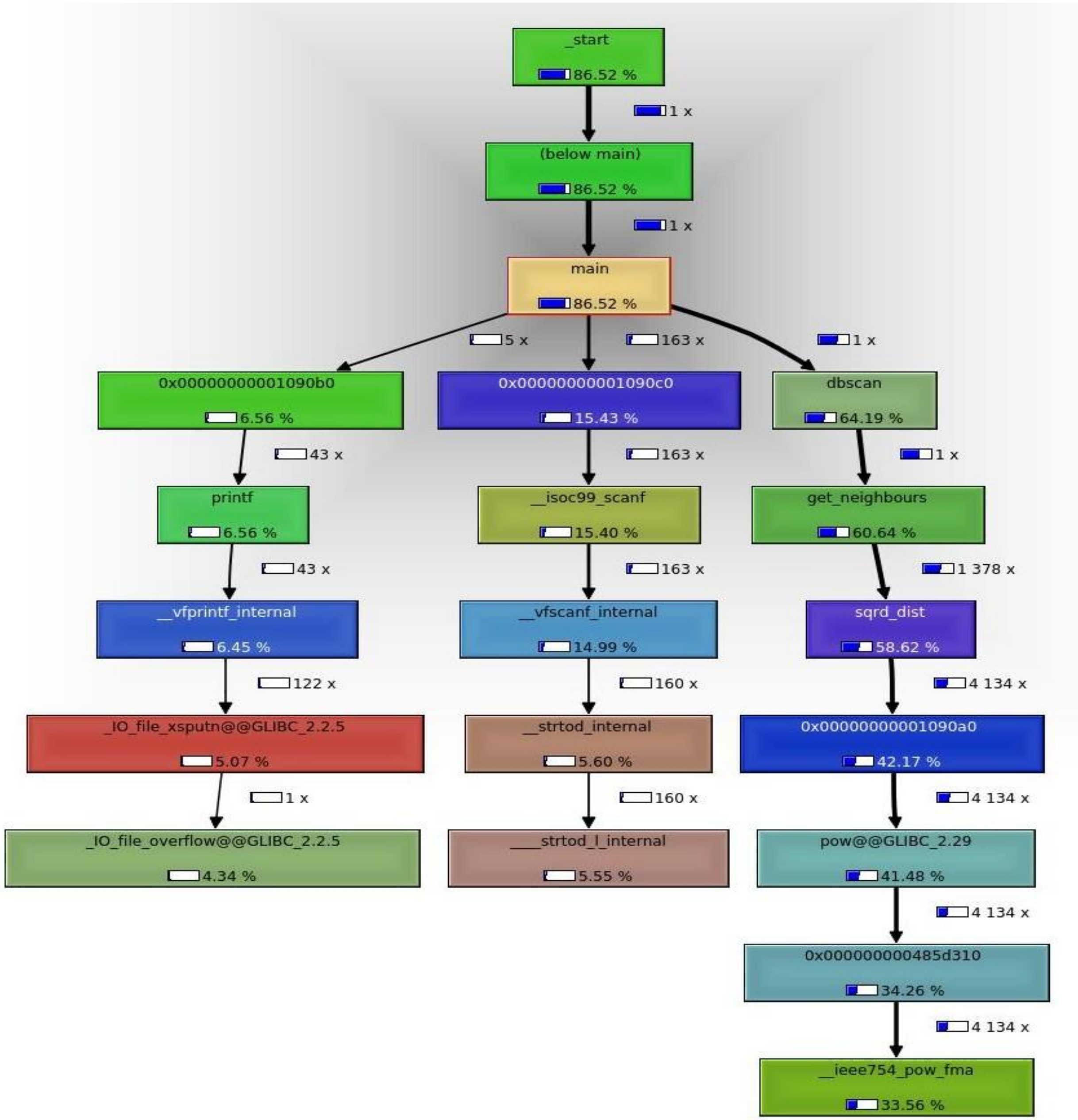
Call Graph:

```
Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00    1378/1378        get_neighbours [4]
[1]      0.0    0.00    0.00    1378         sqrd_dist [1]
-----------------------------------------------
                                  48             dfs [2]
                0.00    0.00       4/4           dbscan [3]
[2]      0.0    0.00    0.00       4+48      dfs [2]
                                  48             dfs [2]
-----------------------------------------------
                0.00    0.00       1/1           main [11]
[3]      0.0    0.00    0.00       1         dbscan [3]
                0.00    0.00       4/4           dfs [2]
                0.00    0.00       1/1           get_neighbours [4]
-----------------------------------------------
                0.00    0.00       1/1           dbscan [3]
[4]      0.0    0.00    0.00       1         get_neighbours [4]
                0.00    0.00    1378/1378        sqrd_dist [1]
-----------------------------------------------
```

Call Graph using valgrind:



_start
86.52 %
1 x
(below main)
86.52 %
1 x
main
86.52 %
5 x
163 x
1 x
0x00000000001090b0
6.56 %
0x00000000001090c0
15.43 %
dbscan
64.19 %
43 x
163 x
1 x
printf
6.56 %
__isoc99_scanf
15.40 %
get_neighbours
60.64 %
43 x
163 x
1 378 x
__vfprintf_internal
6.45 %
__vfscanf_internal
14.99 %
sqrd_dist
58.62 %
122 x
160 x
4 134 x
_IO_file_xsputn@@GLIBC_2.2.5
5.07 %
__strtod_internal
5.60 %
0x00000000001090a0
42.17 %
1 x
160 x
4 134 x
_IO_file_overflow@@GLIBC_2.2.5
4.34 %
____strtod_l_internal
5.55 %
pow@@GLIBC_2.29
41.48 %
4 134 x
0x000000000485d310
34.26 %
4 134 x
__ieee754_pow_fma
33.56 %

Line - Based Profiling Results:

Serial.c.gcov file link:
https://drive.google.com/file/d/1aZVBVYqr31j19KC2skX7TTY8CwktLHgv/view?usp=sharing

Process Resource Utilization Report:

Group 1: L3

Event	Counter	HWThread 0	HWThread 1	HWThread 2
INSTR_RETIRED_ANY	FIXC0	231330	4137	0
CPU_CLK_UNHALTED_CORE	FIXC1	321123	23341	0
CPU_CLK_UNHALTED_REF	FIXC2	723750	52800	0
L2_LINES_IN_ALL	PMC0	9065	690	0
L2_TRANS_L2_WB	PMC1	2507	58	0

Metric	HWThread 0	HWThread 1	HWThread 2
Runtime (RDTSC) [s]	0.0012	0.0012	0.0012
Runtime unhalted [s]	0.0002	1.296728e-05	0
Clock [MHz]	798.6444	795.7124	-
CPI	1.3882	5.6420	-
L3 load bandwidth [MBytes/s]	474.2809	36.1008	0
L3 load data volume [GBytes]	0.0006	4.416000e-05	0
L3 evict bandwidth [MBytes/s]	131.1663	3.0346	0
L3 evict data volume [GBytes]	0.0002	3.712000e-06	0
L3 bandwidth [MBytes/s]	605.4471	39.1354	0
L3 data volume [GBytes]	0.0007	4.787200e-05	0

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	0.0036	0.0012	0.0012	0.0012
Runtime unhalted [s] STAT	0.0002	0	0.0002	0.0001
Clock [MHz] STAT	1594.3568	795.7124	798.6444	531.4523
CPI STAT	7.0302	1.3882	5.6420	2.3434
L3 load bandwidth [MBytes/s] STAT	510.3817	0	474.2809	170.1272
L3 load data volume [GBytes] STAT	0.0006	0	0.0006	0.0002
L3 evict bandwidth [MBytes/s] STAT	134.2009	0	131.1663	44.7336
L3 evict data volume [GBytes] STAT	0.0002	0	0.0002	0.0001
L3 bandwidth [MBytes/s] STAT	644.5825	0	605.4471	214.8608
L3 data volume [GBytes] STAT	0.0007	0	0.0007	0.0002

Observations :

1)
We can reduce all the DFS calls by using a disjoint set data structure.It will decrease time complexity by a constant factor.

2)
For finding all the neighbours we can use parallelization to make it faster.