

## OPTICS Clustering

The C++ implementation for OPTICS clustering

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>OPTICS</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	DataSet Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.1.2	Constructor & Destructor Documentation . . . . .	7
4.1.2.1	DataSet() . . . . .	7
4.1.2.2	~DataSet() . . . . .	8
4.1.3	Member Data Documentation . . . . .	8
4.1.3.1	dataMatrix . . . . .	8
4.1.3.2	dataVec . . . . .	8
4.1.3.3	dimension . . . . .	8
4.1.3.4	fullName . . . . .	8
4.1.3.5	maxElements . . . . .	8
4.1.3.6	strName . . . . .	8
4.1.3.7	vertexCount . . . . .	8
4.2	DensityClustering Class Reference . . . . .	9
4.2.1	Detailed Description . . . . .	10
4.2.2	Constructor & Destructor Documentation . . . . .	10

4.2.2.1	DensityClustering(const int &argc, char **argv)	10
4.2.2.2	~DensityClustering()	11
4.2.3	Member Function Documentation	11
4.2.3.1	computeCoredDistance(const float &radius_eps, const int &minPts)	11
4.2.3.2	extractFeatures(const float &radius_eps, const int &minPts)	12
4.2.3.3	getDistRange(float &minDist, float &maxDist)	14
4.2.3.4	getGroup(const float &radius_eps)	15
4.2.3.5	getMinPt_thDist(const int &minPts)	15
4.2.3.6	getReachability(const int &first, const int &target, const int &minPts)	16
4.2.3.7	OPTICS(const float &radius_eps, const int &minPts)	16
4.2.3.8	performClustering()	17
4.2.3.9	regionQuery(const int &index, const float &radius_eps)	18
4.2.3.10	setDataset(const int &argc, char **argv)	18
4.2.3.11	setMinPts()	19
4.2.3.12	setNormOption()	19
4.2.3.13	setTimesMin(const float &minDist, const float &maxDist)	19
4.2.3.14	update(const int &index, const vector< int > &neighbor, LinkedList &seeds, const float &radius_eps, const int &minPts)	20
4.2.3.15	writeReachability()	20
4.2.4	Member Data Documentation	20
4.2.4.1	ds	20
4.2.4.2	isPathlines	21
4.2.4.3	isPBF	21
4.2.4.4	nodeVec	21
4.2.4.5	normOption	21
4.2.4.6	object	21
4.2.4.7	orderedList	21
4.3	LinkedList Class Reference	21
4.3.1	Detailed Description	22
4.3.2	Constructor & Destructor Documentation	22
4.3.2.1	LinkedList()	22

4.3.2.2	<a href="#">LinkedList(const OrderedPoint &amp;value)</a>	22
4.3.2.3	<a href="#">~LinkedList()</a>	22
4.3.3	<a href="#">Member Function Documentation</a>	23
4.3.3.1	<a href="#">deleteNode(pointNode *&amp;pNode)</a>	23
4.3.3.2	<a href="#">insertNode(pointNode *vNode)</a>	23
4.3.3.3	<a href="#">updateNode(const int &amp;index, const float &amp;newDist)</a>	23
4.3.4	<a href="#">Member Data Documentation</a>	24
4.3.4.1	<a href="#">start</a>	24
4.4	<a href="#">OrderedPoint Struct Reference</a>	24
4.4.1	<a href="#">Detailed Description</a>	24
4.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	25
4.4.2.1	<a href="#">OrderedPoint(const int &amp;index, const float &amp;reachabilityDist)</a>	25
4.4.2.2	<a href="#">OrderedPoint()</a>	25
4.4.3	<a href="#">Member Data Documentation</a>	25
4.4.3.1	<a href="#">index</a>	25
4.4.3.2	<a href="#">reachabilityDist</a>	25
4.5	<a href="#">pointNode Struct Reference</a>	25
4.5.1	<a href="#">Detailed Description</a>	26
4.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	26
4.5.2.1	<a href="#">pointNode(const OrderedPoint &amp;value)</a>	26
4.5.2.2	<a href="#">pointNode()</a>	26
4.5.2.3	<a href="#">~pointNode()</a>	26
4.5.3	<a href="#">Member Data Documentation</a>	27
4.5.3.1	<a href="#">next</a>	27
4.5.3.2	<a href="#">value</a>	27
4.6	<a href="#">PointNode Struct Reference</a>	27
4.6.1	<a href="#">Detailed Description</a>	27
4.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	27
4.6.2.1	<a href="#">PointNode()</a>	27
4.6.2.2	<a href="#">~PointNode()</a>	28
4.6.3	<a href="#">Member Data Documentation</a>	28
4.6.3.1	<a href="#">core_distance</a>	28
4.6.3.2	<a href="#">group</a>	28
4.6.3.3	<a href="#">neighbor</a>	28
4.6.3.4	<a href="#">reachabilityDist</a>	28
4.6.3.5	<a href="#">type</a>	28
4.6.3.6	<a href="#">visited</a>	28

<b>5 File Documentation</b>	<b>29</b>
5.1 main.cpp File Reference . . . . .	29
5.1.1 Function Documentation . . . . .	29
5.1.1.1 main(int argc, char **argv) . . . . .	29
5.2 OPTICS.cpp File Reference . . . . .	30
5.2.1 Variable Documentation . . . . .	30
5.2.1.1 activityList . . . . .	30
5.2.1.2 minPts . . . . .	30
5.2.1.3 multiTimes . . . . .	30
5.2.1.4 timeList . . . . .	30
5.3 OPTICS.h File Reference . . . . .	31
5.4 Predefined.h File Reference . . . . .	31
5.4.1 Enumeration Type Documentation . . . . .	33
5.4.1.1 PointType . . . . .	33
5.5 README.md File Reference . . . . .	33
<b>Index</b>	<b>35</b>

## Chapter 1

# OPTICS

We implemented the basic OPTICS as described by [wikipedia](#). The OPTICS is claimed to have better performance than the DBSCAN with more natural clusters generated while solving the leading drawback of DBSCAN, at the cost of more complicated parameter setting.





## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">DataSet</a>	7
<a href="#">DensityClustering</a>	9
<a href="#">LinkedList</a>	21
<a href="#">OrderedPoint</a>	24
<a href="#">pointNode</a>	25
<a href="#">PointNode</a>	27



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	29
<a href="#">OPTICS.cpp</a>	30
<a href="#">OPTICS.h</a>	31
<a href="#">Predefined.h</a>	31



## Chapter 4

# Class Documentation

### 4.1 DataSet Struct Reference

```
#include <Predefined.h>
```

#### Public Member Functions

- [DataSet \(\)](#)
- [~DataSet \(\)](#)

#### Public Attributes

- `vector< vector< float > >` [dataVec](#)
- `Eigen::MatrixXf` [dataMatrix](#)
- `int` [maxElements](#)
- `int` [vertexCount](#)
- `int` [dimension](#)
- `string` [strName](#)
- `string` [fullName](#)

#### 4.1.1 Detailed Description

Definition at line 193 of file Predefined.h.

#### 4.1.2 Constructor & Destructor Documentation

##### 4.1.2.1 DataSet::DataSet ( ) [inline]

Definition at line 204 of file Predefined.h.

```
205     {}
```

#### 4.1.2.2 `DataSet::~DataSet ( ) [inline]`

Definition at line 207 of file `Predefined.h`.

```
208     {}
```

### 4.1.3 Member Data Documentation

#### 4.1.3.1 `Eigen::MatrixXf DataSet::dataMatrix`

Definition at line 196 of file `Predefined.h`.

#### 4.1.3.2 `vector<vector<float>> DataSet::dataVec`

Definition at line 195 of file `Predefined.h`.

#### 4.1.3.3 `int DataSet::dimension`

Definition at line 199 of file `Predefined.h`.

#### 4.1.3.4 `string DataSet::fullName`

Definition at line 202 of file `Predefined.h`.

#### 4.1.3.5 `int DataSet::maxElements`

Definition at line 197 of file `Predefined.h`.

#### 4.1.3.6 `string DataSet::strName`

Definition at line 201 of file `Predefined.h`.

#### 4.1.3.7 `int DataSet::vertexCount`

Definition at line 198 of file `Predefined.h`.

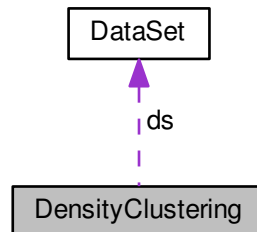
The documentation for this struct was generated from the following file:

- [Predefined.h](#)

## 4.2 DensityClustering Class Reference

```
#include <OPTICS.h>
```

Collaboration diagram for DensityClustering:



### Public Member Functions

- [DensityClustering](#) (const int &argc, char \*\*argv)
- [~DensityClustering](#) ()
- void [performClustering](#) ()

### Private Member Functions

- void [setDataset](#) (const int &argc, char \*\*argv)
- void [setNormOption](#) ()
- void [OPTICS](#) (const float &radius\_eps, const int &minPts)
- void [update](#) (const int &index, const vector< int > &neighbor, [LinkedList](#) &seeds, const float &radius\_eps, const int &minPts)
- const vector< int > [regionQuery](#) (const int &index, const float &radius\_eps)
- void [getDistRange](#) (float &minDist, float &maxDist)
- const int [setMinPts](#) ()
- const float [setTimesMin](#) (const float &minDist, const float &maxDist)
- const float [getReachability](#) (const int &first, const int &target, const int &minPts)
- void [extractFeatures](#) (const float &radius\_eps, const int &minPts)
- void [computeCoredDistance](#) (const float &radius\_eps, const int &minPts)
- void [getGroup](#) (const float &radius\_eps)
- void [writeReachability](#) ()
- const float [getMinPt\\_thDist](#) (const int &minPts)

### Private Attributes

- vector< int > [orderedList](#)
- vector< [PointNode](#) > [nodeVec](#)
- MetricPreparation [object](#)
- int [normOption](#)
- [DataSet](#) [ds](#)
- bool [isPBF](#)
- bool [isPathlines](#)

## 4.2.1 Detailed Description

Definition at line 20 of file OPTICS.h.

## 4.2.2 Constructor & Destructor Documentation

### 4.2.2.1 DensityClustering::DensityClustering ( const int & argc, char \*\* argv )

Definition at line 32 of file OPTICS.cpp.

```

34 {
35     struct timeval start, end;
36     double timeTemp;
37     gettimeofday(&start, NULL);
38
39     setDataset(argc, argv);
40     setNormOption();
41
42     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
43     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
44                         ds.dataMatrix.cols(), normOption);
45
46     /* if the dataset is not PBF, then should record distance matrix for Gamma matrix computation */
47     if(!isPBF)
48     {
49         deleteDistanceMatrix(ds.dataMatrix.rows());
50
51         getDistanceMatrix(ds.dataMatrix, normOption, object);
52
53         std::ifstream distFile("../dataset/"+to_string(normOption)).c_str(), ios::in);
54         if(distFile.fail())
55         {
56             distFile.close();
57             getDistanceMatrix(ds.dataMatrix, normOption, object);
58             std::ofstream distFileOut("../dataset/"+to_string(normOption)).c_str(), ios::out);
59             for(int i=0; i<ds.dataMatrix.rows(); ++i)
60             {
61                 for(int j=0; j<ds.dataMatrix.rows(); ++j)
62                 {
63                     distFileOut << distanceMatrix[i][j] << " ";
64                 }
65                 distFileOut << std::endl;
66             }
67             distFileOut.close();
68         }
69         else
70         {
71             std::cout << "read distance matrix..." << std::endl;
72
73             distanceMatrix = new float*[ds.dataMatrix.rows()];
74             #pragma omp parallel for schedule(static) num_threads(8)
75             for (int i = 0; i < ds.dataMatrix.rows(); ++i)
76             {
77                 distanceMatrix[i] = new float[ds.dataMatrix.rows()];
78             }
79             int i=0, j;
80             string line;
81             stringstream ss;
82             while(getline(distFile, line))
83             {
84                 j=0;
85                 ss.str(line);
86                 while(ss>>line)
87                 {
88                     if(i==j)
89                         distanceMatrix[i][j]=0;
90                     else
91                         distanceMatrix[i][j] = std::atof(line.c_str());
92                     ++j;
93                 }
94                 ++i;
95                 ss.str("");
96                 ss.clear();
97             }
98             distFile.close();
99         }
100     }

```



```

100
101     gettimeofday(&end, NULL);
102     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
103               + end.tv_usec - start.tv_usec) / 1.e6;
104     activityList.push_back("Distance matrix for norm "+to_string(
105         normOption)+" takes: ");
106     timeList.push_back(to_string(timeTemp)+" s");
107     nodeVec = vector<PointNode>(ds.dataMatrix.rows(),
108         PointNode());
109 }

```

#### 4.2.2.2 DensityClustering::~DensityClustering ( )

Definition at line 114 of file OPTICS.cpp.

```

115 {
116
117 }

```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 void DensityClustering::computeCoredDistance ( const float & *radius\_eps*, const int & *minPts* ) [private]

Definition at line 684 of file OPTICS.cpp.

```

686 {
687     #pragma omp parallel for schedule(static) num_threads(8)
688     for (int i = 0; i < ds.dataMatrix.rows(); ++i)
689     {
690         vector<float> distRecord; //record distance value
691         for (int j = 0; j < ds.dataMatrix.rows(); ++j)
692         {
693             if(j==i)
694                 continue;
695             float tempDist;
696             if(distanceMatrix)
697                 tempDist = distanceMatrix[i][j];
698             else
699                 tempDist = getDisimilarity(ds.dataMatrix.row(i),ds.
700 dataMatrix.row(j),i, j, normOption, object);
701             if(tempDist<=radius_eps)
702             {
703                 nodeVec[i].neighbor.push_back(j);
704                 distRecord.push_back(tempDist);
705             }
706             /* find minPts-th smallest element in vector by linear traversal */
707             if(distRecord.size()>=minPts)
708             {
709                 /* A k*n complex algorithm */
710                 /*
711                 vector<float> smallestRange(minPts,FLT_MAX); //update to get minPts-th smallest
712                 for(int k=0;k<distRecord.size();++k)
713                 {
714                     if(distRecord[k]<smallestRange[minPts-1])
715                         smallestRange[minPts-1]=distRecord[k];
716                     for(int l=minPts-1;l>=1;--l)
717                     {
718                         if(smallestRange[l]>smallestRange[l-1])
719                             std::swap(smallestRange[l], smallestRange[l-1]);
720                     }
721                 }
722                 nodeVec[i].core_distance = smallestRange[minPts-1];
723                 */
724
725                 /* instead we shall apply a n*logk algorithm */
726                 std::priority_queue<float> smallestRange;
727                 for(int k=0;k<distRecord.size();++k)
728                 {
729                     smallestRange.push(distRecord[k]);

```

```

730         if(smallestRange.size()>minPts)
731             smallestRange.pop();
732     }
733     nodeVec[i].core_distance = smallestRange.top();
734 }
735 }
736
737 std::cout << "Precomputing for cored-distance is done!" << std::endl;
738 }

```

#### 4.2.3.2 void DensityClustering::extractFeatures ( const float & radius\_eps, const int & minPts ) [private]

Definition at line 473 of file OPTICS.cpp.

```

475 {
476     int maxGroup = -INT_MAX+1;
477 #pragma omp parallel num_threads(8)
478 {
479     #pragma omp for nowait
480     for (int i = 0; i < nodeVec.size(); ++i)
481     {
482         int groupID = nodeVec[i].group;
483         #pragma omp critical
484         {
485             if(groupID!=-1 && groupID>maxGroup)
486                 maxGroup = groupID;
487         }
488     }
489 }
490 std::cout << "Max group is: " << maxGroup << std::endl;
491
492 /* re-index the group id by increasing number */
493 int numClusters = maxGroup+1;
494 std::vector<int> container(numClusters,0);
495 for (int i = 0; i < nodeVec.size(); ++i)
496 {
497     if(nodeVec[i].group!=-1)
498         ++container[nodeVec[i].group];
499 }
500
501 int increasingOrder[numClusters];
502 std::multimap<int,int> groupMap;
503
504 for (int i = 0; i < numClusters; ++i)
505     groupMap.insert(std::pair<int,int>(container[i],i));
506
507 std::fill(container.begin(), container.end(), 0);
508 int groupNo = 0;
509 for (std::multimap<int,int>::iterator it=groupMap.begin(); it!=groupMap.end(); ++it)
510 {
511     if(it->first>0)
512     {
513         increasingOrder[it->second] = groupNo;
514         container[groupNo] = it->first;
515         ++groupNo;
516     }
517 }
518
519 numClusters = groupNo+1; /* plus -1 as group */
520
521 #pragma omp parallel for schedule(static) num_threads(8)
522 for (int i = 0; i < nodeVec.size(); ++i)
523 {
524     if(nodeVec[i].group!=-1)
525         nodeVec[i].group=increasingOrder[nodeVec[i].group];
526 }
527
528 /* in case -1, we use 0 to record number of -1 as noise */
529
530 std::vector<int> item_cids(nodeVec.size());
531 std::vector<std::vector<int> > storage(numClusters);
532 /* -1 group as group[0] */
533 for (int i = 0; i < nodeVec.size(); ++i)
534 {
535     item_cids[i] = nodeVec[i].group;
536     storage[nodeVec[i].group+1].push_back(i);
537 }
538
539 container.insert(container.begin(), storage[0].size());
540

```

```

541     const int& Row = ds.dataMatrix.rows();
542     float entropy = 0.0, probability;
543     for(int i=0;i<container.size();++i)
544     {
545         probability = float(container[i])/float(Row);
546         entropy+=probability*log2f(probability);
547     }
548     entropy = -entropy/log2f(numClusters);
549
550
551     IOHandler::printClustersNoise(ds.dataVec,item_cids,container,
552     "norm"+to_string(normOption),ds.fullName,ds.
dimension);
553
554     struct timeval start, end;
555     double timeTemp;
556
557     numClusters-=1;
558
559     const int& numNoise = storage[0].size();
560     storage.erase(storage.begin());
561
562
563     /* record labeling information */
564     // IOHandler::generateGroups(storage);
565
566     /* compute the centroid coordinates of each clustered group */
567
568     gettimeofday(&start, NULL);
569
570     Eigen::MatrixXf centroid = MatrixXf::Zero(numClusters,ds.dataMatrix.cols());
571     vector<vector<float>> > cenVec(numClusters);
572     #pragma omp parallel for schedule(static) num_threads(8)
573     for (int i=0;i<numClusters;++i)
574     {
575         const std::vector<int>& groupRow = storage[i];
576         for (int j = 0; j < groupRow.size(); ++j)
577         {
578             centroid.row(i)+=ds.dataMatrix.row(groupRow[j]);
579         }
580         centroid.row(i)/=groupRow.size();
581         const Eigen::VectorXf& vec = centroid.row(i);
582         cenVec[i] = vector<float>(vec.data(), vec.data()+ds.dataMatrix.cols());
583     }
584
585     vector<vector<float>> > closest(numClusters);
586     vector<vector<float>> > furthest(numClusters);
587
588     #pragma omp parallel for schedule(static) num_threads(8)
589     for (int i=0;i<numClusters;++i)
590     {
591         float minDist = FLT_MAX;
592         float maxDist = -10;
593         int minIndex = -1, maxIndex = -1;
594         const std::vector<int>& groupRow = storage[i];
595         const Eigen::VectorXf& eachCentroid = centroid.row(i);
596         for (int j = 0; j < groupRow.size(); ++j)
597         {
598             float distance = getDisimilarity(eachCentroid,ds.dataMatrix,groupRow[j],
normOption,object);
599             if(minDist>distance)
600             {
601                 minDist = distance;
602                 minIndex = groupRow[j];
603             }
604             if(maxDist<distance)
605             {
606                 maxDist = distance;
607                 maxIndex = groupRow[j];
608             }
609         }
610         closest[i] = ds.dataVec[minIndex];
611         furthest[i] = ds.dataVec[maxIndex];
612     }
613
614     /* measure closest and furthest rotation */
615     std::vector<float> closestRot, furthestRot;
616     const float& closestAverage = getRotation(closest, closestRot);
617     const float& furthestAverage = getRotation(furthest, furthestRot);
618
619     gettimeofday(&end, NULL);
620     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
621     activityList.push_back("Feature extraction takes: ");
622     timeList.push_back(to_string(timeTemp)+" s");
623
624     ValidityMeasurement vm;
625     vm.computeValue(normOption, ds.dataMatrix, item_cids, object,

```

```

        isPBF);
626     activityList.push_back("Optics Validity measure is: ");
627     stringstream fc_ss;
628     fc_ss << vm.f_c;
629     timeList.push_back(fc_ss.str());
630
631     gettimeofday(&start, NULL);
632     Silhouette sil;
633     sil.computeValue(normOption, ds.dataMatrix, ds.
dataMatrix.rows(), ds.dataMatrix.cols(),
634                     item_cids, object, numClusters, isPBF);
635
636     gettimeofday(&end, NULL);
637     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
638                + end.tv_usec - start.tv_usec) / 1.e6;
639     activityList.push_back("Silhouette calculation takes: ");
640     timeList.push_back(to_string(timeTemp) + " s");
641
642     std::cout << "Finishing extracting features!" << std::endl;
643     IOHandler::printFeature("norm"+to_string(normOption)+"_closest.vtk",
644                             closest, sil.sCluster, ds.dimension);
645     IOHandler::printFeature("norm"+to_string(normOption)+"_furthest.vtk",
646                             furthest, sil.sCluster, ds.dimension);
647     IOHandler::printFeature("norm"+to_string(normOption)+"_centroid.vtk",
648                             cenVec, sil.sCluster, ds.dimension);
649
650     IOHandler::printToFull(ds.dataVec, sil.sData,
651                             "norm"+to_string(normOption)+"_SValueLine", ds.fullName,
ds.dimension);
652     IOHandler::printToFull(ds.dataVec, item_cids, sil.sCluster,
653                             "norm"+to_string(normOption)+"_SValueCluster", ds.
fullName, ds.dimension);
654
655     activityList.push_back("Norm option is: ");
656     timeList.push_back(to_string(normOption));
657
658     activityList.push_back("numCluster is: ");
659     timeList.push_back(to_string(numClusters));
660
661     activityList.push_back("Noise number is: ");
662     timeList.push_back(to_string(numNoise));
663
664     activityList.push_back("radius eps is: ");
665     timeList.push_back(to_string(multiTimes));
666
667     activityList.push_back("MinPts is: ");
668     timeList.push_back(to_string(minPts));
669
670     IOHandler::generateReadme(activityList, timeList);
671
672     IOHandler::writeReadme(closestAverage, furthestAverage);
673
674     IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));
675 }

```

#### 4.2.3.3 void DensityClustering::getDistRange ( float & minDist, float & maxDist ) [private]

Definition at line 386 of file OPTICS.cpp.

```

388 {
389     const float& Percentage = 0.1;
390     const int& Rows = ds.dataMatrix.rows();
391     const int& chosen = int(Percentage*Rows);
392     minDist = FLT_MAX;
393     maxDist = -1.0;
394     #pragma omp parallel num_threads(8)
395     {
396         #pragma omp for nowait
397         for (int i = 0; i < chosen; ++i)
398         {
399             for (int j = 0; j < Rows; ++j)
400             {
401                 if(i==j)
402                     continue;
403                 float dist;
404                 if(distanceMatrix)
405                     dist = distanceMatrix[i][j];
406                 else
407                     dist = getDisimilarity(ds.dataMatrix.row(i), ds.
dataMatrix.row(j), i, j, normOption, object);

```

```

408         #pragma omp critical
409         {
410             if (dist<minDist)
411                 minDist=dist;
412             if (dist>maxDist)
413                 maxDist=dist;
414         }
415     }
416 }
417 }
418 }

```

#### 4.2.3.4 void DensityClustering::getGroup ( const float & radius\_eps ) [private]

Definition at line 771 of file OPTICS.cpp.

```

772 {
773     std::cout << "----Parameter regime----" << std::endl;
774     int continueOption;
775     do
776     {
777         /* group tag information */
778         int tag = 0;
779         std::cout << "Input threshold for OPTICS reachability-plot: ";
780         float threshold;
781         std::cin >> threshold;
782         threshold*=radius_eps;
783         std::cout << threshold << std::endl;
784         bool findSummit = false;
785         for(int i=0;i<orderedList.size();++i)
786         {
787             if(nodeVec[orderedList[i]].reachabilityDist>=threshold)
788             {
789                 findSummit = true;
790             }
791             else
792             {
793                 if(findSummit)
794                 {
795                     findSummit = false;
796                     ++tag;
797                 }
798                 nodeVec[orderedList[i]].group = tag;
799             }
800         }
801         std::cout << "Finally it forms " << (tag+1) << " clusters!" << std::endl;
802         std::cout << "Want to continue with parameter? 1. Yes, 0. No." << std::endl;
803         std::cin >> continueOption;
804         assert(continueOption==1||continueOption==0);
805     }while(continueOption==1);
806     writeReachability();
807 }

```

#### 4.2.3.5 const float DensityClustering::getMinPt\_thDist ( const int & minPts ) [private]

Definition at line 836 of file OPTICS.cpp.

```

837 {
838     float result = 0.0;
839     const int& rowSize = ds.dataMatrix.rows();
840     const int& seletedRow = 0.2*rowSize;
841     #pragma omp parallel num_threads(8)
842     {
843         #pragma omp for nowait
844         for (int i = 0; i < seletedRow; ++i)
845         {
846             /* a linear k*n implementation by directly using a vector for linear mapping */
847             /*
848             std::vector<float> minDistVec(minPts, FLT_MAX);
849             float tempDist;
850             for (int j=0;j<rowSize;++j)
851             {

```

```

852         if(i==j)
853             continue;
854         if(distanceMatrix)
855             tempDist = distanceMatrix[i][j];
856         else
857             tempDist=getDisimilarity(ds.dataMatrix.row(i), ds.dataMatrix.row(j),i,j,normOption,
object);
858
859         if(tempDist<minDistVec[minPts-1])
860             minDistVec[minPts-1]=tempDist;
861         for(int l=minPts-1;l>=1;--l)
862         {
863             if(minDistVec[l]>minDistVec[l-1])
864                 std::swap(minDistVec[l], minDistVec[l-1]);
865         }
866     }*/
867
868     /* use a priority_queue<float> with n*logk time complexity */
869     std::priority_queue<float> minDistArray;
870     float tempDist;
871     for (int j=0; j<rowSize;++j)
872     {
873         if(i==j)
874             continue;
875         if(distanceMatrix)
876             tempDist = distanceMatrix[i][j];
877         else
878             tempDist=getDisimilarity(ds.dataMatrix.row(i),
ds.dataMatrix.row(j),i,j,normOption, object);
879
880         minDistArray.push(tempDist);
881         if(minDistArray.size()>minPts)
882             minDistArray.pop();
883     }
884 }
885
886 #pragma omp critical
887     result += minDistArray.top();
888 }
889 }
890 return result/seletedRow;
891 }

```

#### 4.2.3.6 const float DensityClustering::getReachability ( const int & first, const int & target, const int & minPts ) [private]

Definition at line 749 of file OPTICS.cpp.

```

752 {
753     const int& size = nodeVec[target].neighbor.size();
754     if(size<minPts)
755         return -1.0;
756     float dist;
757
758     if(distanceMatrix)
759         dist = distanceMatrix[first][target];
760     else
761         dist = getDisimilarity(ds.dataMatrix.row(first),ds.
dataMatrix.row(target),first, target, normOption, object);
762     return std::max(nodeVec[target].core_distance, dist);
763 }

```

#### 4.2.3.7 void DensityClustering::OPTICS ( const float & radius\_eps, const int & minPts ) [private]

Definition at line 172 of file OPTICS.cpp.

```

174 {
175     computeCoredDistance(radius_eps, minPts);
176
177     for(int i=0;i<ds.dataMatrix.rows();++i)
178     {
179         if(nodeVec[i].visited)

```

```

180         continue;
181         const vector<int>& neighbor = nodeVec[i].neighbor;
182         /*regionQuery(i,radius_eps)*/;
183         nodeVec[i].visited = true;
184         orderedList.push_back(i);
185         if(nodeVec[i].core_distance!=-1.0)
186         {
187             /* linkedList is good, but would be O(n), so delete it */
188             LinkedList seeds;
189
190             /* should use priority queue */
191             update(i, neighbor, seeds, radius_eps, minPts);
192             pointNode *temp = seeds.start;
193             while(temp)
194             {
195                 if(nodeVec[temp->value.index].visited)
196                 {
197                     temp = temp->next;
198                     continue;
199                 }
200                 const vector<int>& neighborChild = nodeVec[temp->value.
index].neighbor;
201                 /*regionQuery(temp->value.index, radius_eps)*/;
202                 nodeVec[temp->value.index].visited = true;
203                 orderedList.push_back(temp->value.index);
204                 if(nodeVec[temp->value.index].core_distance!=-1.0)
205                     update(temp->value.index, neighborChild, seeds, radius_eps,
minPts);
206                 temp = seeds.start;
207             }
208         }
209     }
210 }

```

#### 4.2.3.8 void DensityClustering::performClustering ( )

Definition at line 123 of file OPTICS.cpp.

```

124 {
125     float radius_eps;
126
127     int minPts = setMinPts();
128
129     int epsOption = 2;
130     /*std::cout << "Choose eps selection method. 1. user input of multiplication, 2. minPt-th dist." <<
std::endl;
std::cin >> epsOption;
assert(epsOption==1||epsOption==2);*/
131     std::cin >> epsOption;
132     assert(epsOption==1||epsOption==2);*/
133
134     if(epsOption==1)
135     {
136         float minDist, maxDist;
137         getDistRange(minDist, maxDist);
138         std::cout << "Distance range is [" << minDist << ", "
139             << maxDist << "]." << std::endl;
140         multiTimes = setTimesMin(minDist, maxDist);
141
142         radius_eps = maxDist*multiTimes;
143     }
144     else if(epsOption==2)
145     {
146         radius_eps = getMinPt_thDist(minPts);
147     }
148
149     struct timeval start, end;
150     double timeTemp;
151     gettimeofday(&start, NULL);
152
153     OPTICS(radius_eps, minPts);
154
155     gettimeofday(&end, NULL);
156     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u
157         + end.tv_usec - start.tv_usec) / 1.e6;
158     activityList.push_back("OPTICS clustering for norm "+to_string(
normOption)+" takes: ");
159     timeList.push_back(to_string(timeTemp)+" s");
160
161     getGroup(radius_eps);
162     extractFeatures(radius_eps, minPts);
163 }

```

#### 4.2.3.9 `const vector< int > DensityClustering::regionQuery ( const int & index, const float & radius_eps )` [private]

Definition at line 264 of file OPTICS.cpp.

```

266 {
267     vector<int> neighborArray;
268     neighborArray.push_back(index);
269     float tempDist;
270     for (int i = 0; i < ds.dataMatrix.rows(); ++i)
271     {
272         if(i==index)
273             continue;
274         if(distanceMatrix)
275             tempDist = distanceMatrix[index][i];
276         else
277             tempDist=getDisimilarity(ds.dataMatrix.row(index),ds.
dataMatrix.row(i),index,i,normOption, object);
278         if(tempDist<=radius_eps)
279             neighborArray.push_back(i);
280     }
281     return neighborArray;
282 }
```

#### 4.2.3.10 `void DensityClustering::setDataset ( const int & argc, char ** argv )` [private]

Definition at line 291 of file OPTICS.cpp.

```

293 {
294     if(argc!=3)
295     {
296         std::cout << "Input argument should have 3!" << endl
297         << " ./cluster inputFile_name(in dataset folder) "
298         << "data_dimension(3)" << endl;
299         exit(1);
300     }
301     ds.strName = string("../dataset/") + string(argv[1]);
302     ds.dimension = atoi(argv[2]);
303
304     /* need to judge whether it is a PBF dataset or not */
305     std::cout << "It is a PBF dataset? 1.Yes, 0.No." << std::endl;
306     int PBFInput;
307     std::cin >> PBFInput;
308     assert(PBFInput==1||PBFInput==0);
309     isPBF = (PBFInput==1);
310
311     std::cout << "It is a pathlines dataset? 1.Yes, 0.No." << std::endl;
312     std::cin >> PBFInput;
313     assert(PBFInput==1||PBFInput==0);
314     isPathlines = (PBFInput==1);
315
316     int sampleOption;
317
318     if(isPathlines)
319         sampleOption = 1;
320     else
321     {
322         std::cout << "choose a sampling method for the dataset?" << std::endl
323         << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
324         std::cin >> sampleOption;
325     }
326     assert(sampleOption==1||sampleOption==2);
327
328     IOHandler::readFile(ds.strName,ds.dataVec,ds.vertexCount,
ds.dimension,ds.maxElements);
329
330     ds.fullName = ds.strName+"_differentNorm_full.vtk";
331     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
vertexCount, ds.dimension);
332
333     if(sampleOption==1)
334         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
dimension,ds.maxElements);
335     else if(sampleOption==2)
336         IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.
dimension,ds.maxElements);
337 }
```



**4.2.3.11** `const int DensityClustering::setMinPts ( )` [private]

Definition at line 424 of file OPTICS.cpp.

```

425 {
426     /*std::cout << std::endl;
427     std::cout << "Input the minPts for OPTICS in [0" << ", "
428         << ds.dataMatrix.rows() << "], 6 is recommended:" << std::endl;
429     */
430     int minPts = 6;
431     //std::cin >> minPts;
432
433     if(minPts<=0 || minPts>=ds.dataMatrix.rows())
434     {
435         std::cout << "Error for out-of-range minPts!" << std::endl;
436         exit(1);
437     }
438     return minPts;
439 }
```

**4.2.3.12** `void DensityClustering::setNormOption ( )` [private]

Definition at line 343 of file OPTICS.cpp.

```

344 {
345     if(isPathlines)
346     {
347         std::cout << "Choose a norm from 0-17!" << std::endl;
348         std::cin >> normOption;
349         assert(normOption>=0 && normOption<=17);
350     }
351     else
352     {
353         std::cout << "Choose a norm from 0-15!" << std::endl;
354         std::cin >> normOption;
355         assert(normOption>=0 && normOption<=15);
356     }
357     /* 0: Euclidean Norm
358        1: Fraction Distance Metric
359        2: piece-wise angle average
360        3: Bhattacharyya metric for rotation
361        4: average rotation
362        5: signed-angle intersection
363        6: normal-direction multivariate distribution
364        7: Bhattacharyya metric with angle to a fixed direction
365        8: Piece-wise angle average \times standard deviation
366        9: normal-direction multivariate un-normalized distribution
367        10: x*y/|x||y| borrowed from machine learning
368        11: cosine similarity
369        12: Mean-of-closest point distance (MCP)
370        13: Hausdorff distance min_max(x_i,y_i)
371        14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
372        15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
373        16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
374        17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
S0097849318300128
375         for pathlines only
376     */
377 }
```

**4.2.3.13** `const float DensityClustering::setTimesMin ( const float & minDist, const float & maxDist )` [private]

Definition at line 449 of file OPTICS.cpp.

```

451 {
452     std::cout << std::endl;
453     float lowerBound = minDist/maxDist;
454     std::cout << "Input the multiplication for OPTICS radius in ["
455         << lowerBound << ",1.0]:" << std::endl;
456     float multiTimes;
457     std::cin >> multiTimes;
458     if(multiTimes>=1.0 || multiTimes<=lowerBound)
459     {
460         std::cout << "Error for out-of-range minPts!" << std::endl;
461         exit(1);
462     }
463     return multiTimes;
464 }
```

#### 4.2.3.14 void DensityClustering::update ( const int & index, const vector< int > & neighbor, LinkedList & seeds, const float & radius\_eps, const int & minPts ) [private]

Definition at line 222 of file OPTICS.cpp.

```

225 {
226     const float& coredist = nodeVec[index].core_distance;
227     for(int i=0;i<neighbor.size();++i)
228     {
229         if(!nodeVec[neighbor[i]].visited)
230         {
231             float dist_toCenter;
232             if(distanceMatrix)
233                 dist_toCenter = distanceMatrix[neighbor[i]][index];
234             else
235                 dist_toCenter = getDisimilarity(ds.dataMatrix.row(neighbor[i]),
236                 ds.dataMatrix.row(index), neighbor[i], index,
237 normOption, object);
238             const float& biggerDist = std::max(coredist, dist_toCenter);
239             if(nodeVec[neighbor[i]].reachabilityDist== -1.0)
240             {
241                 nodeVec[neighbor[i]].reachabilityDist=biggerDist;
242                 seeds.insertNode(new pointNode(OrderedPoint(neighbor[i],
243 biggerDist)));
244             }
245             else
246             {
247                 float& reachDist = nodeVec[neighbor[i]].reachabilityDist;
248                 if(biggerDist<reachDist)
249                 {
250                     reachDist = biggerDist;
251                     seeds.updateNode(neighbor[i], reachDist);
252                 }
253             }
254 }

```

#### 4.2.3.15 void DensityClustering::writeReachability ( ) [private]

Definition at line 813 of file OPTICS.cpp.

```

814 {
815     ofstream ofile("../dataset/reachability.txt", ios::out);
816     if(!ofile)
817     {
818         std::cout << "Error creating the file!" << std::endl;
819         exit(1);
820     }
821     for (int i = 0; i < orderedList.size(); ++i)
822     {
823         ofile << orderedList[i] << " " << nodeVec[orderedList[i]].
reachabilityDist
824         << " " << nodeVec[orderedList[i]].group << std::endl;
825     }
826     ofile << std::endl;
827     ofile.close();
828 }

```

## 4.2.4 Member Data Documentation

### 4.2.4.1 DataSet DensityClustering::ds [private]

Definition at line 70 of file OPTICS.h.

4.2.4.2 `bool DensityClustering::isPathlines` [private]

Definition at line 80 of file OPTICS.h.

4.2.4.3 `bool DensityClustering::isPBF` [private]

Definition at line 75 of file OPTICS.h.

4.2.4.4 `vector<PointNode> DensityClustering::nodeVec` [private]

Definition at line 55 of file OPTICS.h.

4.2.4.5 `int DensityClustering::normOption` [private]

Definition at line 65 of file OPTICS.h.

4.2.4.6 `MetricPreparation DensityClustering::object` [private]

Definition at line 60 of file OPTICS.h.

4.2.4.7 `vector<int> DensityClustering::orderedList` [private]

Definition at line 50 of file OPTICS.h.

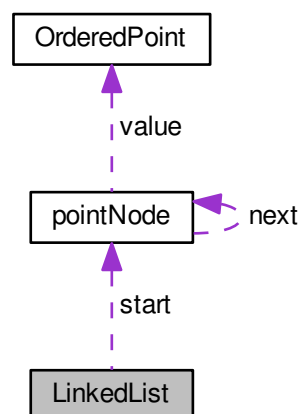
The documentation for this class was generated from the following files:

- [OPTICS.h](#)
- [OPTICS.cpp](#)

## 4.3 LinkedList Class Reference

```
#include <Predefined.h>
```

Collaboration diagram for LinkedList:



## Public Member Functions

- [LinkedList](#) ()
- [LinkedList](#) (const [OrderedPoint](#) &value)
- [~LinkedList](#) ()
- void [insertNode](#) ([pointNode](#) \*vNode)
- void [updateNode](#) (const int &index, const float &newDist)

## Public Attributes

- [pointNode](#) \* [start](#)

## Private Member Functions

- void [deleteNode](#) ([pointNode](#) \*&pNode)

### 4.3.1 Detailed Description

Definition at line 93 of file Predefined.h.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 [LinkedList::LinkedList](#) ( ) [inline]

Definition at line 98 of file Predefined.h.

```

98         : start (NULL)
99     {}

```

#### 4.3.2.2 [LinkedList::LinkedList](#) ( const [OrderedPoint](#) & *value* ) [inline]

Definition at line 101 of file Predefined.h.

```

101         : start (new pointNode (value))
102     {}

```

#### 4.3.2.3 [LinkedList::~~LinkedList](#) ( ) [inline]

Definition at line 104 of file Predefined.h.

```

105     {
106         deleteNode (start);
107     }

```

### 4.3.3 Member Function Documentation

#### 4.3.3.1 void LinkedList::deleteNode ( pointNode \* pNode ) [inline], [private]

Definition at line 175 of file Predefined.h.

```

176     {
177         if(pNode==NULL)
178             return;
179         else if (pNode && pNode->next==NULL)
180         {
181             delete pNode;
182             pNode = NULL;
183             return;
184         }
185         deleteNode(pNode->next);
186     }

```

#### 4.3.3.2 void LinkedList::insertNode ( pointNode \* vNode ) [inline]

Definition at line 109 of file Predefined.h.

```

110     {
111         if(start==NULL)
112         {
113             start = vNode;
114             return;
115         }
116         else if (vNode->value.reachabilityDist<start->
value.reachabilityDist)
117         {
118             vNode->next = start;
119             start = vNode;
120             return;
121         }
122         pointNode *temp = start;
123         while(temp->next && temp->next->value.reachabilityDist<vNode->
value.reachabilityDist)
124         {
125             temp=temp->next;
126         }
127         vNode->next = temp->next;
128         temp->next = vNode;
129     }

```

#### 4.3.3.3 void LinkedList::updateNode ( const int & index, const float & newDist ) [inline]

Definition at line 132 of file Predefined.h.

```

133     {
134         pointNode *temp;
135         if(start==NULL)
136         {
137             std::cout << "Linked list is empty!" << std::endl;
138             return;
139         }
140         if(start->value.index==index)
141         {
142             temp = start;
143             start = start->next;
144             temp->next = NULL;
145             temp->value.reachabilityDist = newDist;
146             insertNode(temp);
147             return;
148         }
149         pointNode *before = start;
150

```

```

151         temp = before->next;
152         while(temp)
153         {
154             if(temp->value.index==index)
155                 break;
156             temp=temp->next;
157             before=before->next;
158         }
159         if(temp==NULL)
160         {
161             std::cout << "Indexed node is not inside the linked list!" << std::endl;
162             return;
163         }
164         else
165         {
166             temp->value.reachabilityDist = newDist;
167             before->next = temp->next;
168             temp->next = NULL;
169             insertNode(temp);
170         }
171     }

```

### 4.3.4 Member Data Documentation

#### 4.3.4.1 pointNode\* LinkedList::start

Definition at line 96 of file Predefined.h.

The documentation for this class was generated from the following file:

- [Predefined.h](#)

## 4.4 OrderedPoint Struct Reference

```
#include <Predefined.h>
```

### Public Member Functions

- [OrderedPoint](#) (const int &[index](#), const float &[reachabilityDist](#))
- [OrderedPoint](#) ()

### Public Attributes

- int [index](#)
- float [reachabilityDist](#)

#### 4.4.1 Detailed Description

Definition at line 49 of file Predefined.h.

## 4.4.2 Constructor & Destructor Documentation

### 4.4.2.1 OrderedPoint::OrderedPoint ( const int & *index*, const float & *reachabilityDist* ) [inline]

Definition at line 54 of file Predefined.h.

```
54                                     :  
55     index(index), reachabilityDist(reachabilityDist)  
56     {}
```

### 4.4.2.2 OrderedPoint::OrderedPoint ( ) [inline]

Definition at line 58 of file Predefined.h.

```
58                                     : index(-1), reachabilityDist(-1.0)  
59     {}
```

## 4.4.3 Member Data Documentation

### 4.4.3.1 int OrderedPoint::index

Definition at line 51 of file Predefined.h.

### 4.4.3.2 float OrderedPoint::reachabilityDist

Definition at line 52 of file Predefined.h.

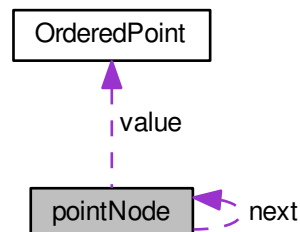
The documentation for this struct was generated from the following file:

- [Predefined.h](#)

## 4.5 pointNode Struct Reference

```
#include <Predefined.h>
```

Collaboration diagram for pointNode:



## Public Member Functions

- [pointNode](#) (const [OrderedPoint](#) &[value](#))
- [pointNode](#) ()
- [~pointNode](#) ()

## Public Attributes

- [OrderedPoint](#) [value](#)
- [pointNode](#) \* [next](#)

### 4.5.1 Detailed Description

Definition at line 66 of file `Predefined.h`.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 `pointNode::pointNode ( const OrderedPoint & value )` `[inline]`

Definition at line 71 of file `Predefined.h`.

```
71                                     : value(value), next(NULL)
72     {
73     }
```

#### 4.5.2.2 `pointNode::pointNode ( )` `[inline]`

Definition at line 75 of file `Predefined.h`.

```
75                                     : value(OrderedPoint()), next(NULL)
76     {
77     }
```

#### 4.5.2.3 `pointNode::~~pointNode ( )` `[inline]`

Definition at line 79 of file `Predefined.h`.

```
80     {
81         if(next)
82         {
83             delete next;
84             next = NULL;
85         }
86     }
```



### 4.5.3 Member Data Documentation

#### 4.5.3.1 `pointNode*` `pointNode::next`

Definition at line 69 of file `Predefined.h`.

#### 4.5.3.2 `OrderedPoint` `pointNode::value`

Definition at line 68 of file `Predefined.h`.

The documentation for this struct was generated from the following file:

- [Predefined.h](#)

## 4.6 PointNode Struct Reference

```
#include <Predefined.h>
```

### Public Member Functions

- [PointNode](#) ()
- [~PointNode](#) ()

### Public Attributes

- int [type](#)
- bool [visited](#)
- int [group](#)
- float [reachabilityDist](#)
- float [core\\_distance](#)
- vector< int > [neighbor](#)

### 4.6.1 Detailed Description

Definition at line 30 of file `Predefined.h`.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 `PointNode::PointNode ( )` [`inline`]

Definition at line 38 of file `Predefined.h`.

```
38         :type(-1), visited(false), group(-1),  
          reachabilityDist(-1.0), core_distance(-1.0)  
39     {}
```

#### 4.6.2.2 `PointNode::~~PointNode ( )` `[inline]`

Definition at line 41 of file `Predefined.h`.

```
42     {}
```

### 4.6.3 Member Data Documentation

#### 4.6.3.1 `float PointNode::core_distance`

Definition at line 36 of file `Predefined.h`.

#### 4.6.3.2 `int PointNode::group`

Definition at line 34 of file `Predefined.h`.

#### 4.6.3.3 `vector<int> PointNode::neighbor`

Definition at line 37 of file `Predefined.h`.

#### 4.6.3.4 `float PointNode::reachabilityDist`

Definition at line 35 of file `Predefined.h`.

#### 4.6.3.5 `int PointNode::type`

Definition at line 32 of file `Predefined.h`.

#### 4.6.3.6 `bool PointNode::visited`

Definition at line 33 of file `Predefined.h`.

The documentation for this struct was generated from the following file:

- [Predefined.h](#)

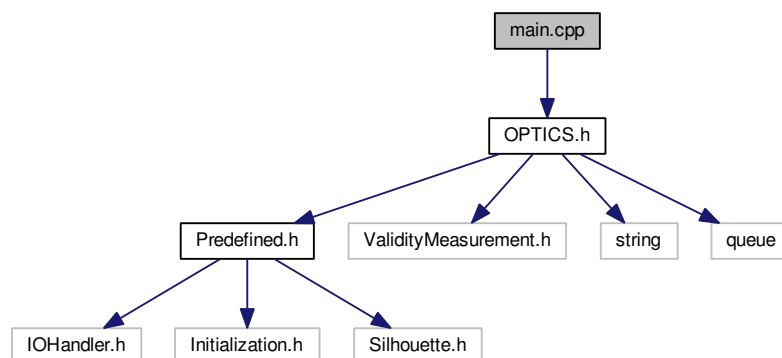
## Chapter 5

# File Documentation

### 5.1 main.cpp File Reference

```
#include "OPTICS.h"
```

Include dependency graph for main.cpp:



### Functions

- int `main` (int argc, char \*\*argv)

#### 5.1.1 Function Documentation

##### 5.1.1.1 int main ( int argc, char \*\* argv )

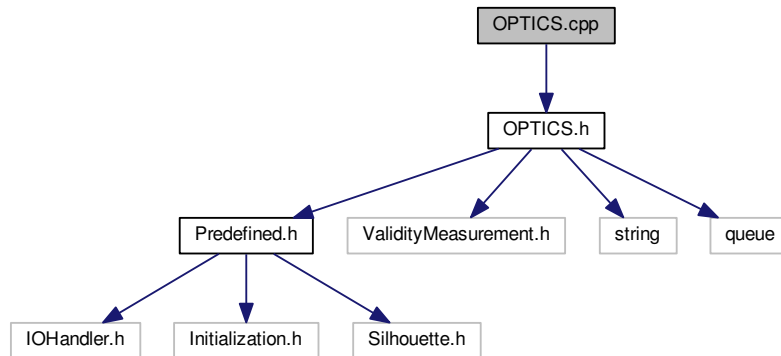
Definition at line 9 of file main.cpp.

```
10 {  
11     DensityClustering dclustering(argc, argv);  
12     dclustering.performClustering();  
13     return 0;  
14 }
```

## 5.2 OPTICS.cpp File Reference

```
#include "OPTICS.h"
```

Include dependency graph for OPTICS.cpp:



### Variables

- `std::vector< string >` [activityList](#)
- `std::vector< string >` [timeList](#)
- `float` [multiTimes](#)
- `int` [minPts](#)

### 5.2.1 Variable Documentation

#### 5.2.1.1 `std::vector<string>` [activityList](#)

Definition at line 11 of file OPTICS.cpp.

#### 5.2.1.2 `int` [minPts](#)

Definition at line 23 of file OPTICS.cpp.

#### 5.2.1.3 `float` [multiTimes](#)

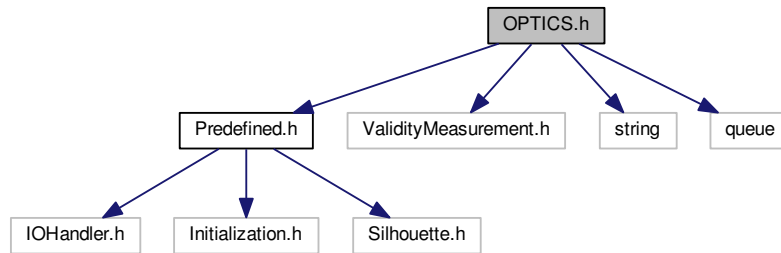
Definition at line 18 of file OPTICS.cpp.

#### 5.2.1.4 `std::vector<string>` [timeList](#)

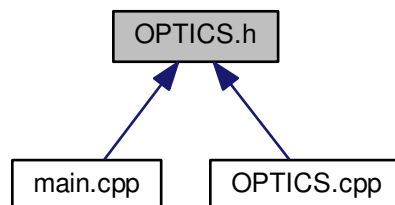
Definition at line 12 of file OPTICS.cpp.

## 5.3 OPTICS.h File Reference

```
#include "Predefined.h"  
#include "ValidityMeasurement.h"  
#include <string>  
#include <queue>  
Include dependency graph for OPTICS.h:
```



This graph shows which files directly or indirectly include this file:



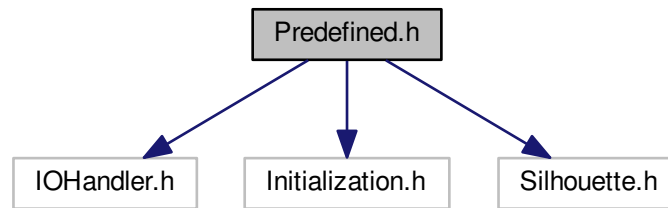
### Classes

- class [DensityClustering](#)

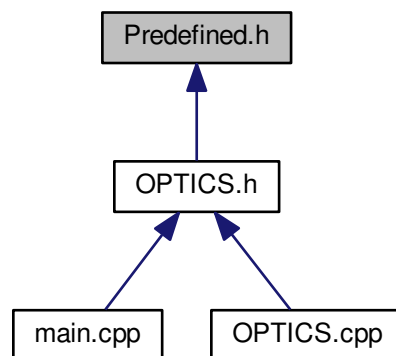
## 5.4 Predefined.h File Reference

```
#include "IOHandler.h"  
#include "Initialization.h"  
#include "Silhouette.h"
```

Include dependency graph for Predefined.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [PointNode](#)
- struct [OrderedPoint](#)
- struct [pointNode](#)
- class [LinkedList](#)
- struct [DataSet](#)

## Enumerations

- enum [PointType](#) { [CORE](#) = 0, [BORDER](#), [NOISE](#) }

### 5.4.1 Enumeration Type Documentation

#### 5.4.1.1 enum PointType

Enumerator

***CORE***

***BORDER***

***NOISE***

Definition at line 19 of file Predefined.h.

```
20 {  
21     CORE = 0,  
22     BORDER,  
23     NOISE  
24 };
```

## 5.5 README.md File Reference





# Index

- ~DataSet
  - DataSet, [7](#)
- ~DensityClustering
  - DensityClustering, [11](#)
- ~LinkedList
  - LinkedList, [22](#)
- ~PointNode
  - PointNode, [27](#)
- ~pointNode
  - pointNode, [26](#)
- activityList
  - OPTICS.cpp, [30](#)
- BORDER
  - Predefined.h, [33](#)
- CORE
  - Predefined.h, [33](#)
- computeCoredDistance
  - DensityClustering, [11](#)
- core\_distance
  - PointNode, [28](#)
- dataMatrix
  - DataSet, [8](#)
- DataSet, [7](#)
  - ~DataSet, [7](#)
  - dataMatrix, [8](#)
  - DataSet, [7](#)
  - dataVec, [8](#)
  - dimension, [8](#)
  - fullName, [8](#)
  - maxElements, [8](#)
  - strName, [8](#)
  - vertexCount, [8](#)
- dataVec
  - DataSet, [8](#)
- deleteNode
  - LinkedList, [23](#)
- DensityClustering, [9](#)
  - ~DensityClustering, [11](#)
  - computeCoredDistance, [11](#)
  - DensityClustering, [10](#)
  - ds, [20](#)
  - extractFeatures, [12](#)
  - getDistRange, [14](#)
  - getGroup, [15](#)
  - getMinPt\_thDist, [15](#)
  - getReachability, [16](#)
- isPBF, [21](#)
- isPathlines, [20](#)
- nodeVec, [21](#)
- normOption, [21](#)
- OPTICS, [16](#)
- object, [21](#)
- orderedList, [21](#)
- performClustering, [17](#)
- regionQuery, [17](#)
- setDataset, [18](#)
- setMinPts, [18](#)
- setNormOption, [19](#)
- setTimesMin, [19](#)
- update, [19](#)
- writeReachability, [20](#)
- dimension
  - DataSet, [8](#)
- ds
  - DensityClustering, [20](#)
- extractFeatures
  - DensityClustering, [12](#)
- fullName
  - DataSet, [8](#)
- getDistRange
  - DensityClustering, [14](#)
- getGroup
  - DensityClustering, [15](#)
- getMinPt\_thDist
  - DensityClustering, [15](#)
- getReachability
  - DensityClustering, [16](#)
- group
  - PointNode, [28](#)
- index
  - OrderedPoint, [25](#)
- insertNode
  - LinkedList, [23](#)
- isPBF
  - DensityClustering, [21](#)
- isPathlines
  - DensityClustering, [20](#)
- LinkedList, [21](#)
  - ~LinkedList, [22](#)
  - deleteNode, [23](#)
  - insertNode, [23](#)
  - LinkedList, [22](#)

- start, [24](#)
  - updateNode, [23](#)
- main
  - main.cpp, [29](#)
- main.cpp, [29](#)
  - main, [29](#)
- maxElements
  - DataSet, [8](#)
- minPts
  - OPTICS.cpp, [30](#)
- multiTimes
  - OPTICS.cpp, [30](#)
- NOISE
  - Predefined.h, [33](#)
- neighbor
  - PointNode, [28](#)
- next
  - pointNode, [27](#)
- nodeVec
  - DensityClustering, [21](#)
- normOption
  - DensityClustering, [21](#)
- OPTICS.cpp, [30](#)
  - activityList, [30](#)
  - minPts, [30](#)
  - multiTimes, [30](#)
  - timeList, [30](#)
- OPTICS.h, [31](#)
- OPTICS
  - DensityClustering, [16](#)
- object
  - DensityClustering, [21](#)
- orderedList
  - DensityClustering, [21](#)
- OrderedPoint, [24](#)
  - index, [25](#)
  - OrderedPoint, [25](#)
  - reachabilityDist, [25](#)
- performClustering
  - DensityClustering, [17](#)
- PointNode, [27](#)
  - ~PointNode, [27](#)
  - core\_distance, [28](#)
  - group, [28](#)
  - neighbor, [28](#)
  - PointNode, [27](#)
  - reachabilityDist, [28](#)
  - type, [28](#)
  - visited, [28](#)
- pointNode, [25](#)
  - ~pointNode, [26](#)
  - next, [27](#)
  - pointNode, [26](#)
  - value, [27](#)
- PointType
  - Predefined.h, [33](#)
- Predefined.h, [31](#)
  - BORDER, [33](#)
  - CORE, [33](#)
  - NOISE, [33](#)
  - PointType, [33](#)
- README.md, [33](#)
- reachabilityDist
  - OrderedPoint, [25](#)
  - PointNode, [28](#)
- regionQuery
  - DensityClustering, [17](#)
- setDataset
  - DensityClustering, [18](#)
- setMinPts
  - DensityClustering, [18](#)
- setNormOption
  - DensityClustering, [19](#)
- setTimesMin
  - DensityClustering, [19](#)
- start
  - LinkedList, [24](#)
- strName
  - DataSet, [8](#)
- timeList
  - OPTICS.cpp, [30](#)
- type
  - PointNode, [28](#)
- update
  - DensityClustering, [19](#)
- updateNode
  - LinkedList, [23](#)
- value
  - pointNode, [27](#)
- vertexCount
  - DataSet, [8](#)
- visited
  - PointNode, [28](#)
- writeReachability
  - DensityClustering, [20](#)