Agglomerative Hierarchical Clustering with distance threshold

The C++ implmentation for AHC with distance threshold

Generated by Doxygen 1.8.11

Contents

Chapter 1

AHC with distance threshold (∗∗not frequently used in flow visualization∗∗)

- It is very similar to ../Birch/README.md "BIRCH" which accepts a distance threshold and will merge all candidates into one group within this distance threshold.
- It has not been applied to flow visualization and the intuition of implementing it is to compare the clustering result to that by Birch
- The implementation can be totally ignored
- The input distance threshold will be used to
  - If merged distance (calculated from the linkage type) is larger than the threshold, the hierarchical merging will terminate
  - If merged distance is below the threshold, the hierarchical merging will continue

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

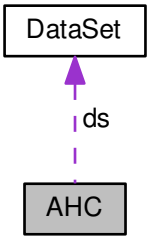
Chapter 4

Class Documentation

4.1 AHC Class Reference

`#include <AHC.h>`

Collaboration diagram for AHC:



Public Member Functions

- AHC ()
- AHC (const int &argc, char **argv)
- ~AHC ()
- void performClustering ()

Private Member Functions

- void extractFeatures (const std::vector< int > &storage, const std::vector< std::vector< int > > &neighborVec, const Eigen::MatrixXf &centroid)
- void setDataset (const int &argc, char **argv)
- void setNormOption ()
- void getDistRange ()

- const float getDistAtNodes (const vector< int > &firstList, const vector< int > &secondList, const int &Linkage)
- void hierarchicalMerging (std::vector< Ensemble > &nodeVec)
- void setLabel (const std::vector< Ensemble > &nodeVec, vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid)
- void bottomUp_byGroup (std::vector< Ensemble > &nodeVec)
- void bottomUp_byThreshold (std::vector< Ensemble > &nodeVec)
- string getLinkageStr ()
- void getEntropyRatio (const std::vector< int > &storage, float &EntropyRatio)
- string getNormStr ()
- string getEntropyStr (const float &EntropyRatio)

Private Attributes

- MetricPreparation object
- bool isPBF
- int normOption
- std::vector< int > group
- std::vector< string > activityList
- std::vector< string > timeList
- float distanceThreshold
- DataSet ds
- int numberOfClusters
- int expectedClusters
- vector< float > distRange
- int linkageOption

4.1.1 Detailed Description

Definition at line 29 of file AHC.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AHC::AHC ( )

Definition at line 17 of file AHC.cpp.

```
17          {
18
19 }
```

4.1.2.2 AHC::AHC (const int & *argc*, char ** *argv*)

Definition at line 31 of file AHC.cpp.

```
31                                      {
32
33     // set the data set and norm option
34     setDataset(argc, argv);
35     setNormOption();
36
37     /* very hard to decide whether needed to perform such pre-processing but still create a class
38      * object as cached before the pairwise distance matrix computation
39      */
40     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
41     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
     ds.dataMatrix.cols(), normOption);
42
43     /* would store distance matrix instead because it would save massive time */
44     struct timeval start, end;
45     double timeTemp;
46     gettimeofday(&start, NULL);
47
48     // calculate the distance matrix
49     getDistanceMatrix(ds.dataMatrix, normOption, object);
50
51     gettimeofday(&end, NULL);
52     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
53             - start.tv_usec) / 1.e6;
54     activityList.push_back("Distance matrix computing takes: ");
55     timeList.push_back(to_string(timeTemp) + " s");
56
57     getDistRange();
58 }
```

4.1.2.3 AHC::~AHC ( )

Definition at line 64 of file AHC.cpp.

```
64           {
65     // delete the distance matrix
66     deleteDistanceMatrix(ds.dataMatrix.rows());
67 }
```

4.1.3 Member Function Documentation

4.1.3.1 void AHC::bottomUp_byGroup (std::vector< Ensemble > & *nodeVec*) [private]

Definition at line 121 of file AHC.cpp.

```
121                                                     {
122     const int& Row = ds.dataMatrix.rows();
123     std::cout << "-------------------------------------------------------------------------------" <<
      std::endl;
124     std::cout << "Expected number of clusters from [0, " << Row << "]:";
125     std::cin >> expectedClusters;
126     assert(expectedClusters > 0 && expectedClusters < Row / 10);
127
128     /* would store distance matrix instead because it would save massive time */
129     struct timeval start, end;
130     double timeTemp;
131     gettimeofday(&start, NULL);
132
133     // perform the clustering algorithm until the approximate clusters are reached
134     int clusterCount = 0;
135     const int& minExpected = 0.8 * expectedClusters;
136     const int& maxExpected = 1.2 * expectedClusters;
137     float minDist = distRange[0], maxDist = distRange[1] / 4.0;
138     int iteration = 0;
```

```
139     std::cout << ".." << std::endl;
140     std::cout << ".." << std::endl;
141     std::cout << "Binary search starts!" << std::endl;
142     while (true && iteration <= 20) {
143         distanceThreshold = (minDist + maxDist) / 2.0;
144         hierarchicalMerging(nodeVec);
145         clusterCount = nodeVec.size();
146         std::cout << "Iteration " << (++iteration) << " finds " << clusterCount
147                 << " groups!" << std::endl;
148         if (clusterCount >= minExpected && clusterCount <= maxExpected)
149             break;
150         else if (clusterCount < minExpected)
151             maxDist = distanceThreshold;
152         else
153             minDist = distanceThreshold;
154     }
155
156     // record relevant information
157     gettimeofday(&end, NULL);
158     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
159             - start.tv_usec) / 1.e6;
160     stringstream ss;
161     ss << expectedClusters;
162     string cluster_str = ss.str();
163
164     ss.str("");
165     ss << iteration;
166
167     activityList.push_back("To achieve " + cluster_str + " groups will take " + ss.str()
168                     + " binary search and take: ");
169     timeList.push_back(to_string(timeTemp) + " s");
170
171 }
```

4.1.3.2 void AHC::bottomUp_byThreshold (std::vector< Ensemble > & *nodeVec*) [private]

Definition at line 179 of file AHC.cpp.

```
179                                                     {
180     std::cout
181             << "--------------------------------------------------------------------------------"
182             << std::endl;
183     std::cout << "Input threshold: [" << distRange[0] << "," <<
    distRange[1]
184             << "]: ";
185     std::cin >> distanceThreshold;
186     assert(distanceThreshold > distRange[0] && distanceThreshold < distRange[1]);
187
188     /* would store distance matrix instead because it would save massive time */
189     struct timeval start, end;
190     double timeTemp;
191     gettimeofday(&start, NULL);
192
193     // merge the nodes
194     hierarchicalMerging(nodeVec);
195
196     // record relevant information
197     gettimeofday(&end, NULL);
198     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
199             - start.tv_usec) / 1.e6;
200     stringstream ss;
201     ss << distanceThreshold;
202     activityList.push_back("To cluster by distance " + ss.str() + " will take: ");
203     timeList.push_back(to_string(timeTemp) + " s");
204 }
```

4.1.3.3 void AHC::extractFeatures (const std::vector< int > & *storage*, const std::vector< std::vector< int > > & *neighborVec*, const Eigen::MatrixXf & *centroid*) [private]

Definition at line 365 of file AHC.cpp.

```
367 {
368     const int& Row = ds.dataMatrix.rows();
369     const int& Column = ds.dataMatrix.cols();
370 
371     std::cout << "Final group number information: " << std::endl;
372     for (int i = 0; i < storage.size(); ++i) {
373         std::cout << storage[i] << " ";
374     }
375     std::cout << std::endl;
376 
377     /* record labeling information */
378     // IOHandler::generateGroups(neighborVec);
379 
380     IOHandler::printClusters(ds.dataVec, group, storage, "norm" + to_string(
    normOption),
381             ds.fullName, ds.dimension);
382 
383     struct timeval start, end;
384     double timeTemp;
385 
386     // calculate the evaluation metrics
387     gettimeofday(&start, NULL);
388     Silhouette sil;
389     sil.computeValue(normOption, ds.dataMatrix, ds.
    dataMatrix.rows(), ds.dataMatrix.cols(), group,
390             object, numberOfClusters, isPBF, neighborVec);
391     gettimeofday(&end, NULL);
392     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
393             - start.tv_usec) / 1.e6;
394     activityList.push_back("Silhouette calculation takes: ");
395     timeList.push_back(to_string(timeTemp) + " s");
396 
397     /* compute the centroid coordinates of each clustered group */
398     gettimeofday(&start, NULL);
399 
400     vector<vector<float> > closest(numberOfClusters);
401     vector<vector<float> > furthest(numberOfClusters);
402 
403     /* extract the closest and furthest streamlines to centroid */
404 
405 #pragma omp parallel for schedule(static) num_threads(8)
406     for (int i = 0; i < numberOfClusters; ++i) {
407         float minDist = FLT_MAX;
408         float maxDist = -10;
409         int minIndex = -1, maxIndex = -1;
410         const std::vector<int>& groupRow = neighborVec[i];
411         const Eigen::VectorXf& eachCentroid = centroid.row(i);
412         for (int j = 0; j < groupRow.size(); ++j) {
413             float distance = getDisimilarity(eachCentroid, ds.dataMatrix,
414                     groupRow[j], normOption, object);
415             if (minDist > distance) {
416                 minDist = distance;
417                 minIndex = groupRow[j];
418             }
419             if (maxDist < distance) {
420                 maxDist = distance;
421                 maxIndex = groupRow[j];
422             }
423         }
424         closest[i] = ds.dataVec[minIndex];
425         furthest[i] = ds.dataVec[maxIndex];
426     }
427 
428     std::vector<std::vector<float> > center_vec(numberOfClusters,
429             vector<float>(Column));
430 #pragma omp parallel for schedule(static) num_threads(8)
431     for (int i = 0; i < center_vec.size(); ++i) {
432         for (int j = 0; j < Column; ++j) {
433             center_vec[i][j] = centroid(i, j);
434         }
435     }
436 
437     // calculate the normalized entropy ratio
438     float EntropyRatio;
439     getEntropyRatio(storage, EntropyRatio);
440 
441     gettimeofday(&end, NULL);
442     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
443             - start.tv_usec) / 1.e6;
444     activityList.push_back("Feature extraction takes: ");
445     timeList.push_back(to_string(timeTemp) + " s");
446 
447     // calculate the normalized validity measurement
448     ValidityMeasurement vm;
449     vm.computeValue(normOption, ds.dataMatrix, group, object, isPBF);
450     activityList.push_back("AHC Validity measure is: ");
451     stringstream fc_ss;
```

```
452     fc_ss << vm.f_c;
453     timeList.push_back(fc_ss.str());
454 
455     std::cout << "Finishing extracting features!" << std::endl;
456 
457     // record relevant information
458     stringstream ss;
459     ss << "norm_" << normOption;
460 
461     string linkage = getLinkageStr();
462     string normStr = getNormStr();
463 
464     // print the featured information as result
465     IOHandler::printFeature(
466             ds.dataName + "_AHC_Dist_" + linkage + "_closest_" + ss.str()
467                     + ".vtk", closest, sil.sCluster, ds.dimension);
468     IOHandler::printFeature(
469             ds.dataName + "_AHC_Dist_" + linkage + "_furthest_" + ss.str()
470                     + ".vtk", furthest, sil.sCluster, ds.dimension);
471     IOHandler::printFeature(
472             ds.dataName + "_AHC_Dist_" + linkage + "_centroid_" + ss.str()
473                     + ".vtk", center_vec, sil.sCluster, ds.dimension);
474 
475     IOHandler::printToFull(ds.dataVec, sil.sData,
476             "AHC_Dist_SValueLine_" + ss.str(), ds.fullName, ds.
    dimension);
477     IOHandler::printToFull(ds.dataVec, group, sil.sCluster,
478             "AHC_Dist_SValueCluster_" + ss.str(), ds.fullName, ds.
    dimension);
479 
480     // generate necessary readme file
481     activityList.push_back("numCluster is: ");
482     timeList.push_back(std::to_string(numberOfClusters));
483 
484     IOHandler::generateReadme(activityList, timeList);
485 
486     IOHandler::writeReadme(
487             "Linkage: " + linkage + ", " + "norm option is " + normStr);
488 
489     IOHandler::writeGroupSize(storage);
490 
491     /* print entropy value for the clustering algorithm */
492     IOHandler::writeReadme(EntropyRatio, sil, "For norm "+std::to_string(normOption));
493 
494     /* measure closest and furthest rotation */
495     std::vector<float> closestRot, furthestRot;
496     const float& closestAverage = getRotation(closest, closestRot);
497     const float& furthestAverage = getRotation(furthest, furthestRot);
498 
499     IOHandler::writeReadme(closestAverage, furthestAverage);
500 }
```

4.1.3.4 const float AHC::getDistAtNodes (const vector< int > & *firstList*, const vector< int > & *secondList*, const int & *Linkage*) [private]

Definition at line 654 of file AHC.cpp.

```
656 {
657     const int& m = firstList.size();
658     const int& n = secondList.size();
659     assert(m != 0);
660     assert(n != 0);
661     /* 0: single linkage, min(x_i,y_j)
662      * 1: complete linkdage, max(x_i,y_j)
663      * 2: average linkage, sum/x_i/y_j
664      */
665     float result, value;
666     switch (Linkage)
667     {
668     case 0: //single linkage
669         {
670             result = FLT_MAX;
671         #pragma omp parallel for reduction(min:result) num_threads(8)
672             for (int i = 0; i < m; ++i) {
673                 for (int j = 0; j < n; ++j) {
674                     if (distanceMatrix)
675                         value = distanceMatrix[firstList[i]][secondList[j]];
676                     else
677                         value = getDisimilarity(ds.dataMatrix, firstList[i],
```

```
678                                 secondList[j], normOption, object);
679                     result = std::min(result, value);
680                 }
681             }
682         }
683         break;
684
685     case 1: //complete linkage
686         {
687             result = FLT_MIN;
688         #pragma omp parallel for reduction(max:result) num_threads(8)
689             for (int i = 0; i < m; ++i) {
690                 for (int j = 0; j < n; ++j) {
691                     if (distanceMatrix)
692                         value = distanceMatrix[firstList[i]][secondList[j]];
693                     else
694                         value = getDisimilarity(ds.dataMatrix, firstList[i],
695                                 secondList[j], normOption, object);
696                     result = std::max(result, value);
697                 }
698             }
699         }
700         break;
701
702     case 2:     // average linkage
703         {
704             result = 0;
705         #pragma omp parallel for reduction(+:result) num_threads(8)
706             for (int i = 0; i < m; ++i) {
707                 for (int j = 0; j < n; ++j) {
708                     if (distanceMatrix)
709                         value = distanceMatrix[firstList[i]][secondList[j]];
710                     else
711                         value = getDisimilarity(ds.dataMatrix, firstList[i],
712                                 secondList[j], normOption, object);
713                     result += value;
714                 }
715             }
716             result /= m * n;
717         }
718         break;
719
720     default:    // error
721         std::cout << "error!" << std::endl;
722         exit(1);
723     }
724     return result;
725 }
```

4.1.3.5 void AHC::getDistRange ( ) [private]

Definition at line 608 of file AHC.cpp.

```
608                         {
609     const float& Percentage = 0.05;
610     const int& Row = ds.dataMatrix.rows();
611
612     distRange = vector<float>(2);
613     distRange[0] = FLT_MAX, distRange[1] = FLT_MIN;
614     const int& totalSize = int(Percentage * Row);
615 #pragma omp parallel num_threads(8)
616     {
617     #pragma omp for nowait
618         for (int i = 0; i < totalSize; ++i) {
619             float tempDist, i_min = FLT_MAX, i_max = FLT_MIN;
620             for (int j = 0; j < Row; ++j) {
621                 if (distanceMatrix)
622                     tempDist = distanceMatrix[i][j];
623                 else
624                     tempDist = getDisimilarity(ds.dataMatrix, i, j,
    normOption,
625                                 object);
626                 if (tempDist < i_min) {
627                     i_min = tempDist;
628                 }
629                 if (tempDist > i_max) {
630                     i_max = tempDist;
631                 }
632             }
633     #pragma omp critical
```

```
634             {
635                 distRange[0] = std::min(distRange[0], i_min);
636                 distRange[1] = std::max(distRange[1], i_max);
637             }
638         }
639     }
640
641     std::cout << "Distance threshold is: [" << distRange[0] << ", " << distRange[1] << "]."
642             << std::endl;
643 }
```

4.1.3.6 void AHC::getEntropyRatio (const std::vector< int > & *storage*, float & *EntropyRatio*) [private]

Definition at line 758 of file AHC.cpp.

```
759 {
760     EntropyRatio = 0;
761     const int& Row = ds.dataMatrix.rows();
762     for (int i = 0; i < storage.size(); ++i) {
763         float ratio = float(storage[i]) / float(Row);
764         EntropyRatio -= ratio * log2f(ratio);
765     }
766     EntropyRatio /= log2f(storage.size());
767 }
```

4.1.3.7 string AHC::getEntropyStr (const float & *EntropyRatio*) [private]

Definition at line 786 of file AHC.cpp.

```
787 {
788     stringstream ss;
789     ss << EntropyRatio;
790     return ss.str();
791 }
```

4.1.3.8 string AHC::getLinkageStr ( ) [private]

Definition at line 732 of file AHC.cpp.

```
733 {
734     string result;
735     switch (linkageOption)
736     {
737     case 0:
738         result = "single";
739         break;
740
741     case 1:
742         result = "complete";
743         break;
744
745     case 2:
746         result = "average";
747         break;
748     }
749     return result;
750 }
```

4.1.3.9 string AHC::getNormStr ( ) [private]

Definition at line 774 of file AHC.cpp.

```
774                         {
775     stringstream ss;
776     ss << normOption;
777     return ss.str();
778 }
```

4.1.3.10 void AHC::hierarchicalMerging (std::vector< Ensemble > & *nodeVec*) [private]

Definition at line 212 of file AHC.cpp.

```
212                                                         {
213     const int Row = ds.dataMatrix.rows();
214
215     nodeVec.clear();
216     //could have used vector, but since there're too many operations inside so should use set
217     nodeVec = std::vector<Ensemble>(Row);
218     //create node in forest structure
219 #pragma omp parallel for schedule(static) num_threads(8)
220     for (int i = 0; i < nodeVec.size(); ++i) {
221         nodeVec[i].index = i;
222         nodeVec[i].element.push_back(i);
223     }
224
225     //two iterators to record positions of set
226     std::vector<Ensemble>::iterator iter_i, iter_j;
227
228     //vector to store new node
229     std::vector<Ensemble> newNodeList;
230     do {
231         //insert new node obtained from previous step
232         if (!newNodeList.empty()) {
233             nodeVec.insert(nodeVec.end(), newNodeList.begin(),
234                     newNodeList.end());
235             newNodeList.clear();
236         }
237         //iter_i prior, iter_j consecutive
238         iter_i = nodeVec.begin();
239         iter_j = iter_i;
240         ++iter_j;
241         int mergedCount = 0;
242         while (iter_i != nodeVec.end()) {
243             // j reaches end or i is merged, should move i forward
244             if (iter_j == nodeVec.end() || (*iter_i).merged) {
245                 ++iter_i;
246                 iter_j = iter_i;
247                 ++iter_j;
248             }
249             // j node already merged, so no longer consideration
250             else if ((*iter_j).merged)
251                 ++iter_j;
252             // move j and calculate distance for mutual pairs
253             else {
254                 //compute distance between two clusters by single/complete/average linkages
255                 const float& linkageDist = getDistAtNodes((*iter_i).element,
256                         (*iter_j).element, linkageOption);
257
258                 //larger distance than threshold, then move forward
259                 if (linkageDist > distanceThreshold)
260                     ++iter_j;
261                 //merge two clusters into one cluster if smaller than threshold
262                 else {
263                     //add merged node whose index is total element size
264                     vector<int> first = (*iter_i).element, second =
265                             (*iter_j).element;
266                     Ensemble newNode = Ensemble(first.size() + second.size());
267                     newNode.element = first;
268                     newNode.element.insert(newNode.element.begin(),
269                             second.begin(), second.end());
270                     newNodeList.push_back(newNode);
271
272                     (*iter_i).merged = true;
273                     (*iter_j).merged = true;
```

```
274
275                     ++iter_i;
276                     iter_j = iter_i;
277                     ++iter_j;
278
279                     mergedCount += 2;
280                 }
281             }
282         }
283
284         /* erase would cost so much time so we'd better directly use copy
285          for (auto iter=nodeVec.begin(); iter!=nodeVec.end();)
286          {
287          if((*iter).merged)
288          iter=nodeVec.erase(iter);
289          else
290          ++iter;
291          }*/
292
293         /* use copy and backup to delete those merged elements */
294         assert(nodeVec.size() >= mergedCount);
295         std::vector<Ensemble> copyNode(nodeVec.size() - mergedCount);
296         int c_i = 0;
297         for (int i = 0; i < nodeVec.size(); ++i) {
298             if (!nodeVec[i].merged)
299                 copyNode[c_i++] = nodeVec[i];
300         }
301         nodeVec.clear();
302         nodeVec = copyNode;
303         copyNode.clear();
304
305         mergedCount = 0;
306
307     } while (!newNodeList.empty()); //merging happens constantly
308
309     newNodeList.clear();
310
311     numberOfClusters = nodeVec.size();
312
313     /* use alpha function to sort the group by its size */
314     std::sort(nodeVec.begin(), nodeVec.end(), [](const Ensemble& e1, const
      Ensemble& e2)
315             {   return e1.element.size()<e2.element.size() ||(e1.element.size()==e2.element.
      size()&&e1.index<e2.index);});
316 }
```

4.1.3.11 void AHC::performClustering ( )

Definition at line 78 of file AHC.cpp.

```
78                             {
79
80     std::vector<Ensemble> nodeVec;
81
82     /* perform hierarchical clustering */
83     std::cout << "----------------------" << std::endl;
84     std::cout
85             << "1. clustering by a fixed group, 2. clustering by a distance threshold."
86             << std::endl;
87     int clusteringOption;
88     std::cin >> clusteringOption;
89     assert(clusteringOption == 1 || clusteringOption == 2);
90
91     // choose AHC by distance threshold or fixed group
92     if (clusteringOption == 1)
93         bottomUp_byGroup(nodeVec);
94     else if (clusteringOption == 2)
95         bottomUp_byThreshold(nodeVec);
96
97     vector<vector<int>> neighborVec(numberOfClusters);
98     // element size for all groups
99     vector<int> storage(numberOfClusters);
100
101      // geometric center
102      Eigen::MatrixXf centroid = Eigen::MatrixXf::Zero(numberOfClusters,
103             ds.dataMatrix.cols());
104
105     // set label information
106     setLabel(nodeVec, neighborVec, storage, centroid);
107
```

```
108     nodeVec.clear();
109
110     extractFeatures(storage, neighborVec, centroid);
111 }
```

4.1.3.12 void AHC::setDataset (const int & *argc*, char ** *argv*) [private]

Definition at line 509 of file AHC.cpp.

```
509                                                 {
510     if (argc != 3) {
511         std::cout << "Input argument should have 3!" << endl
512                 << "./cluster inputFile_name(in dataset folder) "
513                 << "data_dimension(3)" << endl;
514         exit(1);
515     }
516     ds.strName = string("../dataset/") + string(argv[1]);
517     ds.dataName = string(argv[1]);
518     ds.dimension = atoi(argv[2]);
519
520     /* get the bool tag for isPBF */
521     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
522     int PBFjudgement;
523     std::cin >> PBFjudgement;
524     assert(PBFjudgement == 1 || PBFjudgement == 0);
525     isPBF = (PBFjudgement == 1);
526
527     // set the sampling option
528     int sampleOption;
529     std::cout << "choose a sampling method for the dataset?" << std::endl
530             << "1.directly filling with last vertex; 2. uniform sampling."
531             << std::endl;
532     std::cin >> sampleOption;
533     assert(sampleOption == 1 || sampleOption == 2);
534
535     // read from the file
536     IOHandler::readFile(ds.strName, ds.dataVec, ds.
    vertexCount, ds.dimension,
537             ds.maxElements);
538
539     ds.fullName = ds.strName + "_full.vtk";
540     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
    vertexCount, ds.dimension);
541
542     // perform sampling
543     if (sampleOption == 1)
544         IOHandler::expandArray(ds.dataMatrix, ds.dataVec,
    ds.dimension,
545                 ds.maxElements);
546     else if (sampleOption == 2)
547         IOHandler::sampleArray(ds.dataMatrix, ds.dataVec,
    ds.dimension,
548                 ds.maxElements);
549
550     group = std::vector<int>(ds.dataMatrix.rows());
551
552     // choose linkage type
553     std::cout << "----------------------------" << std::endl;
554     std::cout
555             << "Input linkage option: 0.single linkage, 1.complete linkage, 2.average linkage"
556             << std::endl;
557     std::cin >> linkageOption;
558     assert(linkageOption == 0 || linkageOption == 1 || linkageOption == 2);
559 }
```

4.1.3.13 void AHC::setLabel (const std::vector< Ensemble > & *nodeVec*, vector< vector< int > > & *neighborVec*, vector< int > & *storage*, Eigen::MatrixXf & *centroid*) [private]

Definition at line 327 of file AHC.cpp.

```
329 {
330 // group tag by increasing order
331     int groupID = 0;
332
333     // element list for each group
334     vector<int> eachContainment;
335
336     // find group id and neighboring vec
337     for (auto iter = nodeVec.begin(); iter != nodeVec.end(); ++iter)
338     {
339         eachContainment = (*iter).element;
340         neighborVec[groupID] = eachContainment;
341     #pragma omp parallel num_threads(8)
342         {
343         #pragma omp for nowait
344             for (int i = 0; i < eachContainment.size(); ++i) {
345                 group[eachContainment[i]] = groupID;
346             #pragma omp critical
347                 centroid.row(groupID) += ds.dataMatrix.row(eachContainment[i]);
348             }
349         }
350         storage[groupID] = (*iter).element.size();
351         centroid.row(groupID) /= eachContainment.size();
352         ++groupID;
353         eachContainment.clear();
354     }
355 }
```

4.1.3.14 void AHC::setNormOption ( ) [private]

Definition at line 565 of file AHC.cpp.

```
565                         {
566     std::cout << "Input a norm option 0-12!" << std::endl;
567     std::cin >> normOption;
568     std::cout << std::endl;
569
570     // choose distance metrics according to number
571     /*  0: Euclidean Norm
572      1: Fraction Distance Metric
573      2: piece-wise angle average
574      3: Bhattacharyya metric for rotation
575      4: average rotation
576      5: signed-angle intersection
577      6: normal-direction multivariate distribution
578      7: Bhattacharyya metric with angle to a fixed direction
579      8: Piece-wise angle average \times standard deviation
580      9: normal-direction multivariate un-normalized distribution
581      10: x*y/|x||y| borrowed from machine learning
582      11: cosine similarity
583      12: Mean-of-closest point distance (MCP)
584      13: Hausdorff distance min_max(x_i,y_i)
585      14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
586      15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
587      16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
588      17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/S0097849318300128
589          for pathlines only
590      */
591     bool found = false;
592     for (int i = 0; i < 16 && !found; ++i) {
593         if (normOption == i) {
594             found = true;
595             break;
596         }
597     }
598     if (!found) {
599         std::cout << "Cannot find the norm!" << std::endl;
600         exit(1);
601     }
602 }
```

4.1.4 Member Data Documentation

4.1.4.1 std::vector<string> AHC::activityList [private]

Definition at line 103 of file AHC.h.

4.1.4.2 float AHC::distanceThreshold [private]

Definition at line 113 of file AHC.h.

4.1.4.3 vector<float> AHC::distRange [private]

Definition at line 133 of file AHC.h.

4.1.4.4 DataSet AHC::ds [private]

Definition at line 118 of file AHC.h.

4.1.4.5 int AHC::expectedClusters [private]

Definition at line 128 of file AHC.h.

4.1.4.6 std::vector<int> AHC::group [private]

Definition at line 98 of file AHC.h.

4.1.4.7 bool AHC::isPBF [private]

Definition at line 88 of file AHC.h.

4.1.4.8 int AHC::linkageOption [private]

Definition at line 138 of file AHC.h.

4.1.4.9 int AHC::normOption [private]

Definition at line 93 of file AHC.h.

4.1.4.10 int AHC::numberOfClusters [private]

Definition at line 123 of file AHC.h.

4.1.4.11 MetricPreparation AHC::object [private]

Definition at line 83 of file AHC.h.

4.1.4.12 std::vector<string> AHC::timeList [private]

Definition at line 108 of file AHC.h.

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

- AHC.h
- AHC.cpp

4.2 DataSet Struct Reference

`#include <Predefined.h>`

Public Attributes

- vector< vector< float > > dataVec
- Eigen::MatrixXf dataMatrix
- int maxElements = -1
- int vertexCount = -1
- int dimension = -1
- string strName
- string fullName
- string dataName

4.2.1 Detailed Description

Definition at line 21 of file Predefined.h.

4.2.2 Member Data Documentation

4.2.2.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 24 of file Predefined.h.

4.2.2.2 string DataSet::dataName

Definition at line 31 of file Predefined.h.

4.2.2.3 vector<vector<float> > DataSet::dataVec

Definition at line 23 of file Predefined.h.

4.2.2.4 int DataSet::dimension = -1

Definition at line 27 of file Predefined.h.

4.2.2.5 string DataSet::fullName

Definition at line 30 of file Predefined.h.

4.2.2.6 int DataSet::maxElements = -1

Definition at line 25 of file Predefined.h.

4.2.2.7 string DataSet::strName

Definition at line 29 of file Predefined.h.

4.2.2.8 int DataSet::vertexCount = -1

Definition at line 26 of file Predefined.h.

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

- Predefined.h

4.3 Ensemble Struct Reference

`#include <Predefined.h>`

Public Member Functions

- Ensemble (const int &index)
- Ensemble ()

Public Attributes

- int index = -1
- bool merged = false
- std::vector< int > element

4.3.1 Detailed Description

Definition at line 39 of file Predefined.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Ensemble::Ensemble (const int & *index*) [inline]

Definition at line 45 of file Predefined.h.

```
45                                  : index(index)
46     {}
```

4.3.2.2 Ensemble::Ensemble ( ) [inline]

Definition at line 48 of file Predefined.h.

```
49     {}
```

4.3.3 Member Data Documentation

4.3.3.1 std::vector<int> Ensemble::element

Definition at line 43 of file Predefined.h.

4.3.3.2 int Ensemble::index = -1

Definition at line 41 of file Predefined.h.

4.3.3.3 bool Ensemble::merged = false

Definition at line 42 of file Predefined.h.

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

- Predefined.h

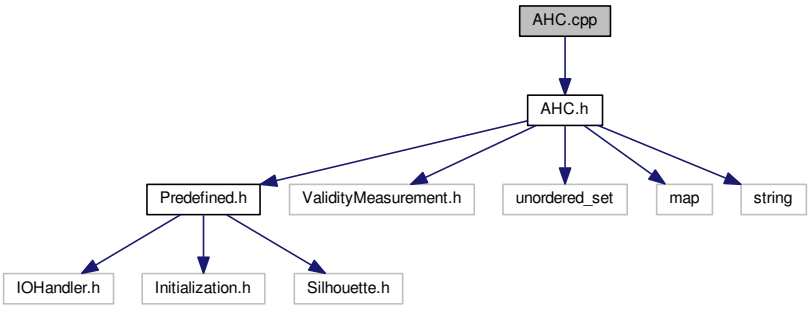
Chapter 5

File Documentation

5.1 AHC.cpp File Reference

```
#include "AHC.h"
```

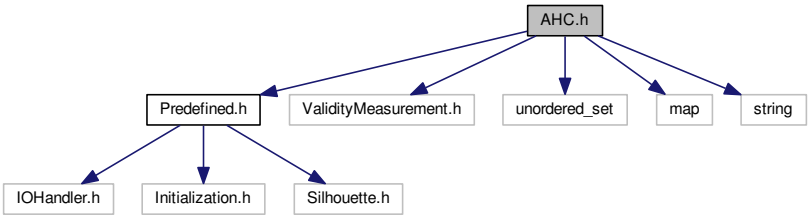
Include dependency graph for AHC.cpp:



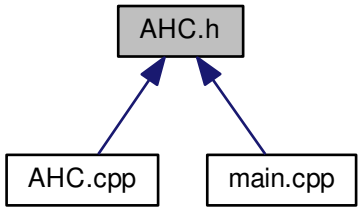
5.2 AHC.h File Reference

```
#include "Predefined.h"
#include "ValidityMeasurement.h"
#include <unordered_set>
#include <map>
#include <string>
```

Include dependency graph for AHC.h:



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



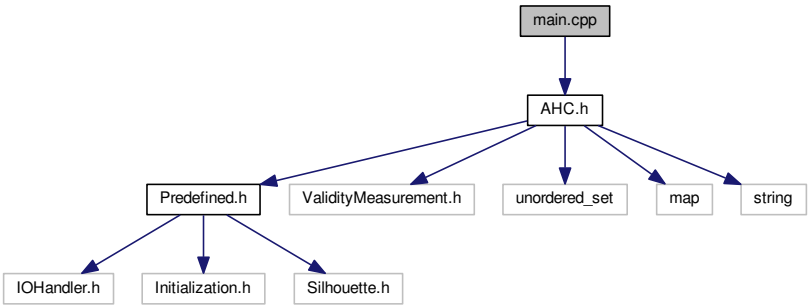
Classes

- class AHC

5.3 main.cpp File Reference

```
#include "AHC.h"
```

Include dependency graph for main.cpp:



Functions

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

5.3.1 Function Documentation

5.3.1.1 int main (int *argc*, char ** *argv*)

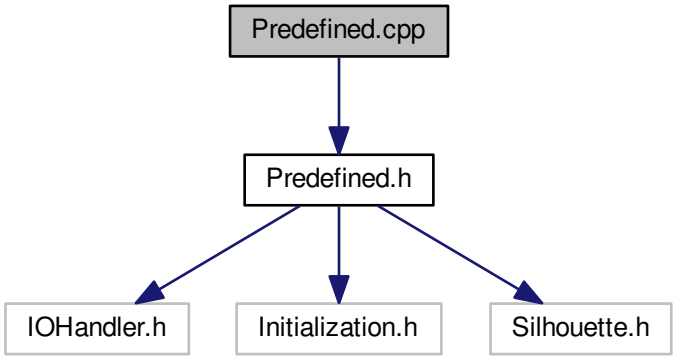
Definition at line 16 of file main.cpp.

```
17 {
18     AHC ahc(argc, argv);
19     ahc.performClustering();
20     return 0;
21 }
```

5.4 Predefined.cpp File Reference

```
#include "Predefined.h"
```

Include dependency graph for Predefined.cpp:



Functions

- template<class T >
  void deleteVecElements (std::vector< T > &original, const T &first, const T &second)

5.4.1 Function Documentation

5.4.1.1 template<class T > void deleteVecElements (std::vector< T > & *original*, const T & *first*, const T & *second*)

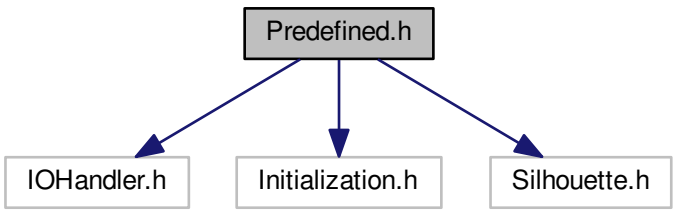
Definition at line 19 of file Predefined.cpp.

```
20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result(size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert(tag==size-2);
33     original = result;
34 }
```

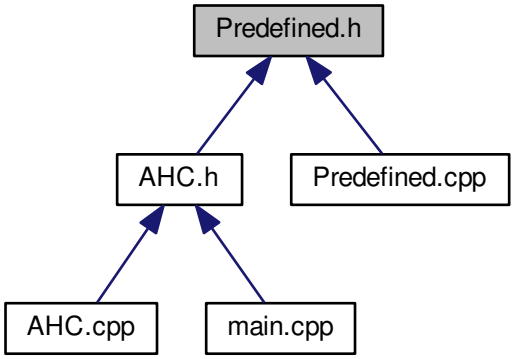
5.5 Predefined.h File Reference

```
#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 DataSet
- struct Ensemble

Functions

- template<class T >
  void deleteVecElements (std::vector< T > &origine, const T &first, const T &second)

5.5.1 Function Documentation

5.5.1.1 template<class T > void deleteVecElements (std::vector< T > & *origine,* const T & *first,* const T & *second*)

Definition at line 19 of file Predefined.cpp.

```
20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result(size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert(tag==size-2);
33     original = result;
34 }
```

5.6 README.md File Reference

Index