Agglomerative Hierarchical Clustering

The C++ implmentation for AHC with OpenMP

Generated by Doxygen 1.8.11

Contents

Chapter 1

Agglomerative Hierarachical Clustering (AHC)

The program includes basically two aspects of AHC

- AHC of three linkages (will generate cluster result information)
  - Single linkage
  - Complete linkage
  - Average linkage
- The hierarchical L method to find optimal number of clusters (only generate optimal cluster number)
  - It is a global search of knee point along the clusters

Number of clusters as input

The program supports two kinds of input for number of clusters

- Direct input after the query information $>$ Input cluster number among [0, 1000] for norm X:
- Read the cluster numbers from a txt file
  - The txt file is called 'cluster_number' in the /dataset/ folder
  - The 'cluster number' has the following format
    0:10 // for similarity measure 0, the input of cluster number is 10 1:10 // for similarity measure 1, the input of cluster number is 10 2:10 4:10 12:10 13:10 14:10 15:10 17:10
    for better batch processing especially in our experiment when the code is automatically run on the server

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:

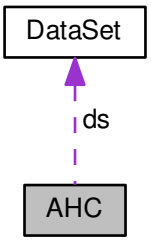
Chapter 4

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 > > &neighbor↩ Vec, const Eigen::MatrixXf &centroid)
- void setDataset (const int &argc, char **argv)
- const float getDistAtNodes (const vector< int > &firstList, const vector< int > &secondList, const int &Link- age)

- void hierarchicalMerging (std::unordered_map< int, Ensemble > &node_map, std::vector< DistNode > &dNodeVec, std::vector< Ensemble > &nodeVec)
- void setLabel (const std::vector< Ensemble > &nodeVec, vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid)
- string getLinkageStr ()
- void getEntropyRatio (const std::vector< int > &storage, float &EntropyRatio)
- string getNormStr ()
- string getEntropyStr (const float &EntropyRatio)
- void setValue_merge (std::vector< DistNode > &dNodeVec, std::unordered_map< int, Ensemble > &node_map)
- void setValue (std::vector< DistNode > &dNodeVec, std::unordered_map< int, Ensemble > &node_map)
- void performClustering_by_norm ()

Private Attributes

- MetricPreparation object
- int normOption
- bool isPBF
- std::vector< int > group
- std::vector< string > activityList
- std::vector< string > timeList
- DataSet ds
- int numberOfClusters
- int initializationOption
- int linkageOption
- bool lMethod
- bool readCluster
- bool isPathlines
- std::vector< float > curveValue [4]

4.1.1 Detailed Description

Definition at line 36 of file AHC.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AHC::AHC ( )

Definition at line 12 of file AHC.cpp.

```
13 {
14
15 }
```

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

Definition at line 26 of file AHC.cpp.

```
27 {
28     setDataset(argc, argv);
29 }
```

4.1.2.3 AHC::~AHC ( )

Definition at line 38 of file AHC.cpp.

```
39 {
40     deleteDistanceMatrix(ds.dataMatrix.rows());
41 }
```

4.1.3 Member Function Documentation

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

Definition at line 473 of file AHC.cpp.

```
475 {
476     const int& Row = ds.dataMatrix.rows();
477     const int& Column = ds.dataMatrix.cols();
478
479     std::cout << "Final group number information: " << std::endl;
480     for (int i = 0; i < storage.size(); ++i)
481     {
482         std::cout << storage[i] << " ";
483     }
484     std::cout << std::endl;
485
486     /* record labeling information */
487     // IOHandler::generateGroups(neighborVec);
488
489     IOHandler::printClusters(ds.dataVec,group,storage,"norm"+to_string(
    normOption),ds.fullName,ds.dimension);
490
491     struct timeval start, end;
492     double timeTemp;
493
494     /* compute the centroid coordinates of each clustered group */
495
496     gettimeofday(&start, NULL);
497
498     vector<vector<float> > closest(numberOfClusters);
499     vector<vector<float> > furthest(numberOfClusters);
500
501     /* extract the closest and furthest streamlines to centroid */
502 #pragma omp parallel for schedule(static) num_threads(8)
503     for (int i=0;i<numberOfClusters;++i)
504     {
505         float minDist = FLT_MAX;
506         float maxDist = -10;
507         int minIndex = -1, maxIndex = -1;
508         const std::vector<int>& groupRow = neighborVec[i];
509         const Eigen::VectorXf& eachCentroid = centroid.row(i);
510         for (int j = 0; j < groupRow.size(); ++j)
511         {
512             float distance = getDisimilarity(eachCentroid,ds.dataMatrix,groupRow[j],
    normOption,object);
513             if(minDist>distance)
514             {
515                 minDist = distance;
516                 minIndex = groupRow[j];
517             }
518             if(maxDist<distance)
519             {
520                 maxDist = distance;
521                 maxIndex = groupRow[j];
522             }
523         }
524         closest[i] = ds.dataVec[minIndex];
525         furthest[i] = ds.dataVec[maxIndex];
526     }
527
528     // re-assign centroid coordinates to the vector<vector<float>>
529     std::vector<std::vector<float> > center_vec(numberOfClusters, vector<float>(Column));
530 #pragma omp parallel for schedule(static) num_threads(8)
531     for (int i = 0; i < center_vec.size(); ++i)
532     {
```

```
533         for (int j = 0; j < Column; ++j)
534         {
535             center_vec[i][j] = centroid(i,j);
536         }
537     }
538 
539     // calculate the normalized entropy
540     float EntropyRatio;
541     getEntropyRatio(storage, EntropyRatio);
542 
543     std::cout << "Entropy ratio is: " << EntropyRatio << std::endl;
544 
545     // record the time for feature extraction
546     gettimeofday(&end, NULL);
547     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
548                 + end.tv_usec - start.tv_usec) / 1.e6;
549     activityList.push_back("Feature extraction for norm "+to_string(
    normOption)+ " takes: ");
550     timeList.push_back(to_string(timeTemp)+" s");
551 
552     // calculate the normalized validity measurement
553     ValidityMeasurement vm;
554     vm.computeValue(normOption, ds.dataMatrix, group, object,
    isPBF);
555     activityList.push_back("AHC Validity measure is: ");
556     stringstream fc_ss;
557     fc_ss << vm.f_c;
558     timeList.push_back(fc_ss.str());
559 
560     // calculate the silhouette, the Gamma statistics and DB index
561     gettimeofday(&start, NULL);
562     Silhouette sil;
563     sil.computeValue(normOption,ds.dataMatrix,ds.
    dataMatrix.rows(),ds.dataMatrix.cols(),
564                      group,object,numberOfClusters,
    isPBF);
565     gettimeofday(&end, NULL);
566     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
567                 + end.tv_usec - start.tv_usec) / 1.e6;
568     activityList.push_back("Silhouette calculation for norm " +to_string(
    normOption)+" takes: ");
569     timeList.push_back(to_string(timeTemp)+" s");
570 
571     std::cout << "Finishing extracting features!" << std::endl;
572 
573     stringstream ss;
574     ss << "norm_" << normOption;
575 
576     /* measure closest and furthest rotation */
577     std::vector<float> closestRotation, furthestRotation;
578     const float& closestAverage = getRotation(closest, closestRotation);
579     const float& furthestAverage = getRotation(furthest, furthestRotation);
580 
581     // record the linkage type, norm option and normalized entropy
582     string linkage = getLinkageStr();
583     string normStr = getNormStr();
584     string entropyStr = getEntropyStr(EntropyRatio);
585 
586     // create the .vtk for streamline labels and cluster representatives
587     IOHandler::printFeature(ds.dataName+"_AHC_"+linkage+"_closest_"+ss.str()+".vtk", closest, sil
    .sCluster,
588                             closestRotation, ds.dimension);
589     IOHandler::printFeature(ds.dataName+"_AHC_"+linkage+"_furthest_"+ss.str()+".vtk", furthest,
    sil.sCluster,
590                             furthestRotation, ds.dimension);
591     IOHandler::printFeature(ds.dataName+"_AHC_"+linkage+"_centroid_"+ss.str()+".vtk", center_vec,
     sil.sCluster,ds.dimension);
592 
593     IOHandler::printToFull(ds.dataVec, sil.sData, "AHC_SValueLine_"+ss.str(),
    ds.fullName, ds.dimension);
594     IOHandler::printToFull(ds.dataVec, group, sil.sCluster, "AHC_SValueCluster_"+ss.str(),
    ds.fullName, ds.dimension);
595 
596     // generate README for evaluation metrics
597     activityList.push_back("numCluster is: ");
598     timeList.push_back(to_string(numberOfClusters));
599 
600     activityList.push_back("Average Silhouette is: ");
601     timeList.push_back(to_string(sil.sAverage));
602 
603     activityList.push_back("Average rotation of closest is: ");
604     timeList.push_back(to_string(closestAverage));
605 
606     activityList.push_back("Average rotation of furthest is: ");
607     timeList.push_back(to_string(furthestAverage));
608 
609     IOHandler::generateReadme(activityList,timeList);
```

```
610     IOHandler::writeReadme("Linkage: "+linkage+", "+"norm option is "+normStr+", ");
611     IOHandler::writeGroupSize(storage);
612 
613     /* print entropy value for the clustering algorithm */
614     IOHandler::writeReadme(EntropyRatio, sil, "For norm "+to_string(normOption));
615     IOHandler::writeReadme(closestAverage, furthestAverage);
616 
617     //curveValue[0].push_back(sil.sAverage);
618     //curveValue[1].push_back(sil.gammaStatistic);
619     //curveValue[2].push_back(sil.dbIndex);
620     //curveValue[3].push_back(vm.f_c);
621 }
```

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

Definition at line 722 of file AHC.cpp.

```
723 {
724     const int& m = firstList.size();
725     const int& n = secondList.size();
726     assert(m!=0);
727     assert(n!=0);
728     /* 0: single linkage, min(x_i,y_j)
729      * 1: complete linkdage, max(x_i,y_j)
730      * 2: average linkage, sum/x_i/y_j
731      */
732     float result, value;
733     switch(Linkage)
734     {
735     case 0: //single linkage
736         {
737             result = FLT_MAX;
738         #pragma omp parallel for reduction(min:result) num_threads(8)
739             for(int i=0;i<m;++i)
740             {
741                 for(int j=0;j<n;++j)
742                 {
743                     if(distanceMatrix)
744                         value = distanceMatrix[firstList[i]][secondList[j]];
745                     else
746                         value = getDisimilarity(ds.dataMatrix, firstList[i], secondList[j],
    normOption, object);
747                     result = std::min(result, value);
748                 }
749             }
750         }
751         break;
752 
753     case 1: //complete linkage
754         {
755             result = -1.0;
756         #pragma omp parallel for reduction(max:result) num_threads(8)
757             for(int i=0;i<m;++i)
758             {
759                 for(int j=0;j<n;++j)
760                 {
761                     if(distanceMatrix)
762                         value = distanceMatrix[firstList[i]][secondList[j]];
763                     else
764                         value = getDisimilarity(ds.dataMatrix, firstList[i], secondList[j],
    normOption, object);
765                     result = std::max(result, value);
766                 }
767             }
768         }
769         break;
770 
771     case 2: // average linkage
772         {
773             result = 0;
774         #pragma omp parallel for reduction(+:result) num_threads(8)
775             for(int i=0;i<m;++i)
776             {
777                 for(int j=0;j<n;++j)
778                 {
779                     if(distanceMatrix)
780                         value = distanceMatrix[firstList[i]][secondList[j]];
781                     else
```

```
782                         value = getDisimilarity(ds.dataMatrix, firstList[i], secondList[j],
    normOption, object);
783                     result+=value;
784                 }
785             }
786             result/=m*n;
787         }
788         break;
789 
790     default:    // no linkage option, should print "Error" information and exit the program
791         std::cout << "error!" << std::endl;
792         exit(1);
793     }
794     return result;
795 }
```

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

Definition at line 829 of file AHC.cpp.

```
830 {
831     EntropyRatio = 0;
832     const int& Row = ds.dataMatrix.rows();
833     for (int i = 0; i < storage.size(); ++i)
834     {
835         float ratio = float(storage[i])/float(Row);
836         EntropyRatio-=ratio*log2f(ratio);
837     }
838     EntropyRatio/=log2f(storage.size());
839 }
```

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

Definition at line 860 of file AHC.cpp.

```
861 {
862     stringstream ss;
863     ss << EntropyRatio;
864     return ss.str();
865 }
```

4.1.3.5 string AHC::getLinkageStr ( ) [private]

Definition at line 802 of file AHC.cpp.

```
803 {
804     string result;
805     switch(linkageOption)
806     {
807     case 0:
808         result = "single";
809         break;
810 
811     case 1:
812         result = "complete";
813         break;
814 
815     case 2:
816         result = "average";
817         break;
818     }
819     return result;
820 }
```

4.1.3.6 string AHC::getNormStr ( ) [private]

Definition at line 846 of file AHC.cpp.

```
847 {
848     stringstream ss;
849     ss << normOption;
850     return ss.str();
851 }
```

4.1.3.7 void AHC::hierarchicalMerging (std::unordered_map< int, Ensemble > & *node_map*, std::vector< DistNode > & *dNodeVec*, std::vector< Ensemble > & *nodeVec*) [private]

Definition at line 270 of file AHC.cpp.

```
272 {
273     std::map<int, float> dist_map;
274
275     /* would store distance matrix instead because it would save massive time */
276     struct timeval start, end;
277     double timeTemp;
278     gettimeofday(&start, NULL);
279
280     const int Row = ds.dataMatrix.rows();
281
282     DistNode poped;
283
284     /* find node-pair with minimal distance */
285     float minDist = FLT_MAX;
286     int target = -1;
287     for (int i = 0; i < dNodeVec.size(); ++i)
288     {
289         if(dNodeVec[i].distance<minDist)
290         {
291             target = i;
292             minDist = dNodeVec[i].distance;
293         }
294     }
295
296     // find which distNode is to be popped
297     poped = dNodeVec[target];
298
299     int index = Row, currentNumber;
300     do  // perform iterative hierarchical merging until the final cluster reaches the given input number
301     {
302         if(lMethod)  // if the l-method is enabled, record the number of clusters and merged
        distance
303         {
304             dist_map.insert(std::make_pair(node_map.size(), poped.distance));
305         }
306         //create new node merged and input it into hash unordered_map
307         vector<int> first = (node_map[poped.first]).element;
308         vector<int> second = (node_map[poped.second]).element;
309
310         /* index would be starting from Row */
311         Ensemble newNode(index);
312         newNode.element = first;
313         newNode.element.insert(newNode.element.end(), second.begin(), second.end());
314         node_map.insert(make_pair(index, newNode));
315
316         //delete two original nodes
317         node_map.erase(poped.first);
318         node_map.erase(poped.second);
319
320         /* the difficulty lies how to update the min-heap with linkage
321          * This would take 2NlogN.
322          * Copy all node-pairs that are not relevant to merged nodes to new vec.
323          * For relevant, would update the mutual distance by linkage
324          */
325
326         /* how many clusters exist */
327         currentNumber = node_map.size();
328
329         target = -1, minDist = FLT_MAX;
330
331         // create new distNode vector
```

```
332         std::vector<DistNode> tempVec(currentNumber*(currentNumber-1)/2);
333 
334         // update and find the minimal distance for next merging
335         int current = 0, i_first, i_second;
336         for(int i=0;i<dNodeVec.size();++i)
337         {
338             i_first=dNodeVec[i].first, i_second=dNodeVec[i].second;
339             /* not relevant, directly copied to new vec */
340             if(i_first!=poped.first&&i_first!=poped.second&&i_second!=poped.
    first&&i_second!=poped.second)
341             {
342                 tempVec[current]=dNodeVec[i];
343                 if(tempVec[current].distance<minDist)
344                 {
345                     target = current;
346                     minDist = tempVec[current].distance;
347                 }
348                 ++current;
349             }
350         }
351 
352         // merge two nodes and update the node-distance relative to these two nodes
353         for (auto iter=node_map.begin();iter!=node_map.end();++iter)
354         {
355             if((*iter).first!=newNode.index)
356             {
357                 tempVec[current].first = (*iter).first;
358                 tempVec[current].second = newNode.index;
359                 tempVec[current].distance=getDistAtNodes(newNode.element,(*iter).second.
    element, linkageOption);
360                 if(tempVec[current].distance<minDist)
361                 {
362                     target = current;
363                     minDist = tempVec[current].distance;
364                 }
365                 ++current;
366             }
367         }
368 
369         if(target>=0 && tempVec.size()>=1)
370         {
371             poped = tempVec[target];
372 
373             /* judge whether current is assigned to right value */
374             assert(current==tempVec.size());
375             dNodeVec.clear();
376             dNodeVec = tempVec;
377             tempVec.clear();
378             ++index;
379         }
380 
381       }while(node_map.size()!=numberOfClusters);  //merging happens whenever requested
     cluster is not met
382 
383       if(lMethod)  // invoke the l-method to find the optimal number of clusters
384       {
385           /* perform L-method computation to detect optimal number of AHC */
386           DetermClusterNum dcn;
387           dcn.iterativeRefinement(dist_map);
388           std::cout << "Otimal number of clusters by L-Method is " << dcn.getFinalNumOfClusters() <<
    std::endl;
389           dcn.recordLMethodResult(normOption);
390       }
391 
392       else    // otherwise, just perform the AHC clustering
393       {
394           nodeVec=std::vector<Ensemble>(node_map.size());
395           int tag = 0;
396           for(auto iter=node_map.begin();iter!=node_map.end();++iter)
397               nodeVec[tag++]=(*iter).second;
398 
399           gettimeofday(&end, NULL);
400           timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
401 
402           activityList.push_back("Hirarchical clustering of norm "+to_string(
    normOption)+" for "+
403                                  to_string(numberOfClusters)+" groups takes: ");
404           timeList.push_back(to_string(timeTemp)+" s");
405           /* task completed, would delete memory contents */
406           dNodeVec.clear();
407           node_map.clear();
408           /* use alpha function to sort the group by its size in ascending order */
409           std::sort(nodeVec.begin(), nodeVec.end(), [](const Ensemble& e1, const
    Ensemble& e2)
410           {return e1.element.size()<e2.element.size()||(e1.element.size()==e2.element.size()&&
    e1.index<e2.index);});
411       }
```

```
412 }
```

4.1.3.8 void AHC::performClustering ( )

Definition at line 165 of file AHC.cpp.

```
166 {
167     /*  0: Euclidean Norm
168         1: Fraction Distance Metric
169         2: piece-wise angle average
170         3: Bhattacharyya metric for rotation
171         4: average rotation
172         5: signed-angle intersection
173         6: normal-direction multivariate distribution
174         7: Bhattacharyya metric with angle to a fixed direction
175         8: Piece-wise angle average \times standard deviation
176         9: normal-direction multivariate un-normalized distribution
177         10: x*y/|x||y| borrowed from machine learning
178         11: cosine similarity
179         12: Mean-of-closest point distance (MCP)
180         13: Hausdorff distance min_max(x_i,y_i)
181         14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
182         15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
183         16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
184         17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
    S0097849318300128
185             for pathlines only
186     */
187
188     // read the input number for different similarity measures from the file "cluster_number"
189     std::unordered_map<int,int> clusterMap;
190     if(readCluster)
191     {
192         IOHandler::readClusteringNumber(clusterMap, "cluster_number");
193     }
194
195     // std::vector<int> cluster_array;
196     //for(int i=2; i<=100; ++i)
197     //{
198         //cluster_array.push_back(i);
199         for(normOption=0;normOption<=17;++normOption)
200         {
201             if(isPathlines)  // for pathlines, will consider d_T (17)
202             {
203                 if(normOption!=0 && normOption!=1 &&
    normOption!=2 && normOption!=4 && normOption!=12
204                    && normOption!=13 && normOption!=14 &&
    normOption!=15 && normOption!=17)
205                     continue;
206             }
207             else    // for streamlines, will not consider d_T (17)
208             {
209                 if(normOption!=0 && normOption!=1 &&
    normOption!=2 && normOption!=4 && normOption!=12
210                 && normOption!=13 && normOption!=14 &&
    normOption!=15)
211                     continue;
212             }
213             std::cout << "------------------------------------------------------------" << std::endl;
214             std::cout << "norm " << normOption << " starts......" << std::endl;
215             timeList.clear();
216             activityList.clear();
217
218             /* L-method is not performed. It's a normal AHC procedure */
219             if(!lMethod)
220             {
221                 const int& Row = ds.dataMatrix.rows();
222                 if(readCluster)
223                 {
224                     numberOfClusters = clusterMap[normOption];
225                 }
226                 else
227                 {
228                     std::cout << "---------------------------------------" << std::endl;
229                     std::cout << "Input cluster number among [0, " << Row << "] for norm " <<
    normOption << ": ";
230                     std::cin >> numberOfClusters;
231                     assert(numberOfClusters>0 && numberOfClusters<Row);
232                 }
233                 //numberOfClusters = i;
```

```
234                 assert(numberOfClusters>0 && numberOfClusters<Row);
235             }
236             /* perform L-method for detecting optimal num of clusters */
237             else if(lMethod)
238             {
239                 numberOfClusters = 1;
240             }
241             // perform clustering by given input of norm option
242             performClustering_by_norm();
243         }
244     //}
245
246     /*std::ofstream curve("../dataset/curveValue.txt", ios::out);
247     for(int i=0; i<4; ++i)
248     {
249         for(int j=0; j<curveValue[0].size(); ++j)
250         {
251             curve << curveValue[i][j] << " ";
252         }
253         curve << std::endl;
254     }
255     curve.close();*/
256 }
```

4.1.3.9 void AHC::performClustering_by_norm ( ) [private]

Definition at line 53 of file AHC.cpp.

```
54 {
55     /* very hard to decide whether needed to perform such pre-processing, but still create a
56      * MetricPreparation object in case some pre-calculation for similarity meaures can be ready
57      * before the pairwise distance matrix calculation
58      */
59     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
60     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
       ds.dataMatrix.cols(), normOption);
61
62     /* would store distance matrix instead because it would save massive time */
63     struct timeval start, end;
64     double timeTemp;
65     gettimeofday(&start, NULL);
66
67     // check whether the file for distance matrix exists or not
68     std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
69     if(distFile.fail()) // not exist, will calculate the distance matrix and store them in local files
70     {
71         distFile.close();
72         // calculate the distance matrix
73         getDistanceMatrix(ds.dataMatrix, normOption, object);
74         // store the distance matrix values in the local files
75         std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
76         for(int i=0;i<ds.dataMatrix.rows();++i)
77         {
78             for(int j=0;j<ds.dataMatrix.rows();++j)
79             {
80                 distFileOut << distanceMatrix[i][j] << " ";
81             }
82             distFileOut << std::endl;
83         }
84         distFileOut.close();
85     }
86     else    // the file for distance matrix exists, then directly reads in the pair-wise values
87     {
88         std::cout << "read distance matrix..." << std::endl;
89
90         distanceMatrix = new float*[ds.dataMatrix.rows()];
91     #pragma omp parallel for schedule(static) num_threads(8)
92         for (int i = 0; i < ds.dataMatrix.rows(); ++i)
93         {
94             distanceMatrix[i] = new float[ds.dataMatrix.rows()];
95         }
96
97         // read the distance values from the .txt file
98         int i=0, j;
99         string line;
100         stringstream ss;
101         while(getline(distFile, line))
102         {
103             j=0;
104             ss.str(line);
```

```
105             while(ss>>line)
106             {
107                 if(i==j)
108                     distanceMatrix[i][j]=0;
109                 else
110                     distanceMatrix[i][j] = std::atof(line.c_str());
111                 ++j;
112             }
113             ++i;
114             ss.str("");
115             ss.clear();
116         }
117         distFile.close();
118     }
119 
120     // record the time for distance matrix computation time
121     gettimeofday(&end, NULL);
122     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
123                 + end.tv_usec - start.tv_usec) / 1.e6;
124     activityList.push_back("Distance matrix computing for norm "+to_string(
    normOption)+" takes: ");
125     timeList.push_back(to_string(timeTemp)+" s");
126 
127     // create node-related parameters for AHC clustering
128     std::unordered_map<int, Ensemble> node_map;
129     std::vector<DistNode> dNodeVec;
130     std::vector<Ensemble> nodeVec;
131 
132     /* set the ditNode vector */
133     setValue_merge(dNodeVec, node_map);
134 
135     /* perform hiarchical clustering where within each step would merge two nodes */
136     hierarchicalMerging(node_map, dNodeVec, nodeVec);
137 
138     if(!lMethod) // perform the AHC clustering with lMethod not activated
139     {
140         vector<vector<int>> neighborVec(numberOfClusters);
141         // element size for all groups
142         vector<int> storage(numberOfClusters);
143 
144         // geometric center
145         Eigen::MatrixXf centroid = Eigen::MatrixXf::Zero(numberOfClusters,
    ds.dataMatrix.cols());
146 
147         // set label information
148         setLabel(nodeVec, neighborVec, storage, centroid);
149 
150         nodeVec.clear();
151 
152         extractFeatures(storage, neighborVec, centroid);
153     }
154 }
```

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

Definition at line 634 of file AHC.cpp.

```
635 {
636     if(argc!=3)
637     {
638         std::cout << "Input argument should have 3!" << endl
639                   << "./cluster inputFile_name(in dataset folder) "
640                   << "data_dimension(3)" << endl;
641         exit(1);
642     }
643     // get the attribute for data set
644     ds.strName = string("../dataset/")+string(argv[1]);
645     ds.dataName = string(argv[1]);
646     ds.dimension = atoi(argv[2]);
647 
648     /* get the bool tag for isPBF */
649     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
650     int PBFjudgement;
651     std::cin >> PBFjudgement;
652     assert(PBFjudgement==1||PBFjudgement==0);
653     isPBF = (PBFjudgement==1);
654 
655     /* get the bool tag for isPBF */
656     std::cout << "It is a pathline dataset? 1.Yes, 0.No" << std::endl;
657     std::cin >> PBFjudgement;
```

```
658     assert(PBFjudgement==1||PBFjudgement==0);
659     isPathlines = (PBFjudgement==1);
660 
661     // set up the sample option by user input and data set type (pathlines or streamlines)
662     int sampleOption;
663     if(isPathlines)  // default direct-repeating for pathlines to match the time steps
664         sampleOption = 1;
665     else    // streamline sample option can be versatile
666     {
667         std::cout << "choose a sampling method for the dataset?" << std::endl
668                   << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
669         std::cin >> sampleOption;
670         assert(sampleOption==1||sampleOption==2);
671     }
672 
673     // read the coordinates into the member variales
674     IOHandler::readFile(ds.strName,ds.dataVec,ds.vertexCount,
    ds.dimension,ds.maxElements);
675 
676     // get the path of full name and print the streamlines vtk
677     ds.fullName = ds.strName+"_full.vtk";
678     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
    vertexCount, ds.dimension);
679 
680     // perform sampling operation with user parameters
681     if(sampleOption==1)
682         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
683     else if(sampleOption==2)
684         IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
685 
686     // create the label index for each individual streamline/pathline
687     group = std::vector<int>(ds.dataMatrix.rows());
688 
689     // whether to activate the L-method or not
690     std::cout << "Perform L-method to detect optimal num of clusters? 0: No, 1: Yes! " << std::endl;
691     std::cin >> lMethod;
692     assert(lMethod==0 || lMethod==1);
693     lMethod = (lMethod==1);
694 
695     // which linkage type to be selected
696     std::cout << "----------------------------" << std::endl;
697     std::cout << "Input linkage option: 0.single linkage, 1.complete linkage, 2.average linkage" <<
    std::endl;
698     std::cin >> linkageOption;
699     assert(linkageOption==0||linkageOption==1||linkageOption==2);
700 
701     // lMethod is not activated, so will ask for number of cluster as input
702     if(!lMethod)
703     {
704         std::cout << "----------------------------" << std::endl;
705         std::cout << "Choose cluster number input method: 0.user input, 1.read from file: " << std::endl;
706         int clusterInput;
707         std::cin >> clusterInput;
708         assert(clusterInput==0||clusterInput==1);
709         readCluster = (clusterInput==1);
710     }
711 }
```

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

Definition at line 426 of file AHC.cpp.

```
428 {
429 // group tag by increasing order
430     int groupID = 0;
431 
432     // element list for each group
433     vector<int> eachContainment;
434 
435     // find group id and neighboring vec
436     for(auto iter = nodeVec.begin(); iter!=nodeVec.end();++iter)
437     {
438         eachContainment = (*iter).element;
439         neighborVec[groupID] = eachContainment;
440     #pragma omp parallel num_threads(8)
441         {
```

```
442         #pragma omp for nowait
443             for(int i=0;i<eachContainment.size();++i)
444             {
445                 // update the label index for each streamline candidates
446                 group[eachContainment[i]] = groupID;
447             #pragma omp critical
448                 // update the centroid coordinates of the cluster
449                 centroid.row(groupID) += ds.dataMatrix.row(eachContainment[i]);
450             }
451         }
452         storage[groupID] = (*iter).element.size();
453         centroid.row(groupID)/=eachContainment.size();
454         ++groupID;
455         eachContainment.clear();
456     }
457 }
```

4.1.3.12 void AHC::setValue (std::vector< DistNode > & *dNodeVec*, std::unordered_map< int, Ensemble > & *node_map*) [private]

Definition at line 955 of file AHC.cpp.

```
956 {
957     const int& Row = ds.dataMatrix.rows();
958     dNodeVec = std::vector<DistNode>(Row*(Row-1)/2);
959     int tag = 0;
960     // record the node i, node j and their distance into the vector
961     for(int i=0;i<Row-1;++i)
962     {
963         for(int j=i+1;j<Row;++j)
964         {
965             dNodeVec[tag].first = i;
966             dNodeVec[tag].second = j;
967             if(distanceMatrix)
968                 dNodeVec[tag].distance = distanceMatrix[i][j];
969             else
970                 dNodeVec[tag].distance = getDisimilarity(ds.dataMatrix, i, j,
    normOption, object);
971             ++tag;
972         }
973     }
974     assert(tag==dNodeVec.size());
975     // record the index of the node
976     for(int i=0;i<Row;++i)
977     {
978         node_map[i].element.push_back(i);
979     }
980 }
```

4.1.3.13 void AHC::setValue_merge (std::vector< DistNode > & *dNodeVec*, std::unordered_map< int, Ensemble > & *node_map*) [private]

Definition at line 874 of file AHC.cpp.

```
875 {
876     const int& Row = ds.dataMatrix.rows();
877
878     /* find the node of closest distance */
879     std::vector<int> miniNode(Row);
880 #pragma omp parallel for schedule(static) num_threads(8)
881     for(int i=0;i<Row;++i)
882     {
883         float miniDist = FLT_MAX, dist;
884         int index = -1;
885         for(int j=0;j<Row;++j)
886         {
887             if(i==j)
888                 continue;
889             if(distanceMatrix)
890                 dist = distanceMatrix[i][j];
891             else
```

```
892                 dist = getDisimilarity(ds.dataMatrix, i, j,
    normOption, object);
893
894             if(miniDist>dist)
895             {
896                 miniDist=dist;
897                 index=j;
898             }
899         }
900         miniNode[i]=index;
901     }
902
903     std::vector<bool> isIn(Row, false);
904
905     // update the map for node
906     int tag = 0;
907     for(int i=0;i<Row;++i)
908     {
909         if(!isIn[i])
910         {
911             Ensemble en;
912             if(miniNode[miniNode[i]]==i)
913             {
914                 en.element.push_back(i);
915                 en.element.push_back(miniNode[i]);
916                 isIn[i]=true;
917                 isIn[miniNode[i]]=true;
918                 node_map[tag] = en;
919             }
920             else
921             {
922                 en.element.push_back(i);
923                 isIn[i]=true;
924                 node_map[tag] = en;
925             }
926             ++tag;
927         }
928     }
929
930     // update the dNodeVec from the newly create nodes
931     const int& mapSize = node_map.size();
932     dNodeVec = std::vector<DistNode>(mapSize*(mapSize-1)/2);
933
934     tag = 0;
935     for(auto start = node_map.begin(); start!=node_map.end(); ++start)
936     {
937         for(auto end = node_map.begin(); end!=node_map.end() && end!=start; ++end)
938         {
939             dNodeVec[tag].first = start->first;
940             dNodeVec[tag].second = end->first;
941             dNodeVec[tag].distance = getDistAtNodes(start->second.element,end->second.element
    , linkageOption);
942             ++tag;
943         }
944     }
945     assert(tag==dNodeVec.size());
946 }
```

4.1.4 Member Data Documentation

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

Definition at line 122 of file AHC.h.

4.1.4.2 std::vector<float> AHC::curveValue[4] [private]

Definition at line 167 of file AHC.h.

4.1.4.3 DataSet AHC::ds [private]

Definition at line 132 of file AHC.h.

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

Definition at line 117 of file AHC.h.

4.1.4.5 int AHC::initializationOption [private]

Definition at line 142 of file AHC.h.

4.1.4.6 bool AHC::isPathlines [private]

Definition at line 162 of file AHC.h.

4.1.4.7 bool AHC::isPBF [private]

Definition at line 112 of file AHC.h.

4.1.4.8 int AHC::linkageOption [private]

Definition at line 147 of file AHC.h.

4.1.4.9 bool AHC::lMethod [private]

Definition at line 152 of file AHC.h.

4.1.4.10 int AHC::normOption [private]

Definition at line 107 of file AHC.h.

4.1.4.11 int AHC::numberOfClusters [private]

Definition at line 137 of file AHC.h.

4.1.4.12 MetricPreparation AHC::object [private]

Definition at line 102 of file AHC.h.

4.1.4.13 bool AHC::readCluster [private]

Definition at line 157 of file AHC.h.

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

Definition at line 127 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 20 of file Predefined.h.

4.2.2 Member Data Documentation

4.2.2.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 23 of file Predefined.h.

4.2.2.2 string DataSet::dataName

Definition at line 30 of file Predefined.h.

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

Definition at line 22 of file Predefined.h.

4.2.2.4 int DataSet::dimension = -1

Definition at line 26 of file Predefined.h.

4.2.2.5 string DataSet::fullName

Definition at line 29 of file Predefined.h.

4.2.2.6 int DataSet::maxElements = -1

Definition at line 24 of file Predefined.h.

4.2.2.7 string DataSet::strName

Definition at line 28 of file Predefined.h.

4.2.2.8 int DataSet::vertexCount = -1

Definition at line 25 of file Predefined.h.

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

- Predefined.h

4.3 DistNode Struct Reference

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

Public Member Functions

- DistNode (const int &first, const int &second, const float &dist)
- DistNode ()

Public Attributes

- int first = -1
- int second = -1
- float distance = -1.0

4.3.1 Detailed Description

Definition at line 69 of file Predefined.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DistNode::DistNode (const int & *first,* const int & *second,* const float & *dist*) [inline]

Definition at line 74 of file Predefined.h.

```
74                                                                  :first(
      first), second(second), distance(dist)
75     {}
```

4.3.2.2 DistNode::DistNode ( ) [inline]

Definition at line 77 of file Predefined.h.

```
78     {}
```

4.3.3 Member Data Documentation

4.3.3.1 float DistNode::distance = -1.0

Definition at line 72 of file Predefined.h.

4.3.3.2 int DistNode::first = -1

Definition at line 71 of file Predefined.h.

4.3.3.3 int DistNode::second = -1

Definition at line 71 of file Predefined.h.

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

- Predefined.h

4.4 Ensemble Struct Reference

`#include <Predefined.h>`

Public Member Functions

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

Public Attributes

- int index = -1
- std::vector< int > element

4.4.1 Detailed Description

Definition at line 38 of file Predefined.h.

4.4.2 Constructor & Destructor Documentation

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

Definition at line 45 of file Predefined.h.

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

4.4.2.2 Ensemble::Ensemble ( ) [inline]

Definition at line 48 of file Predefined.h.

```
49     {}
```

4.4.3 Member Data Documentation

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

Definition at line 43 of file Predefined.h.

4.4.3.2 int Ensemble::index = -1

Definition at line 40 of file Predefined.h.

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

- Predefined.h

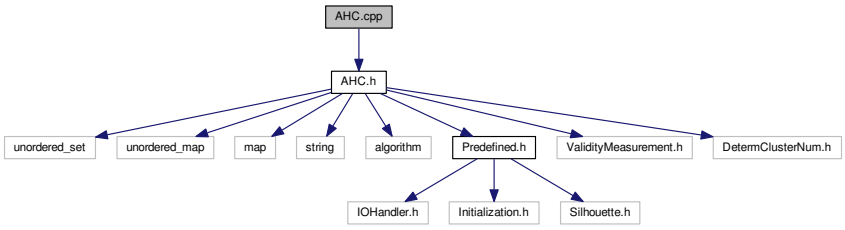
Chapter 5

File Documentation

5.1 AHC.cpp File Reference

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

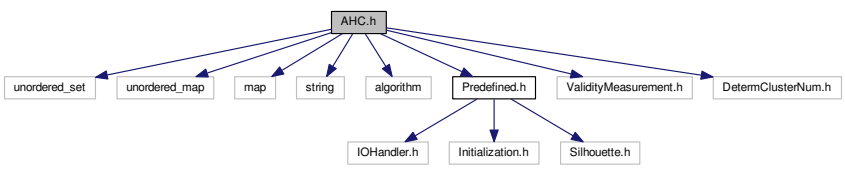
Include dependency graph for AHC.cpp:



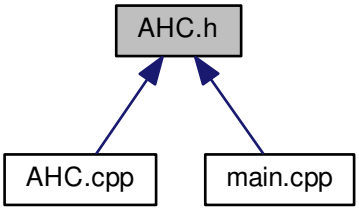
5.2 AHC.h File Reference

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

Include dependency graph for AHC.h:



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



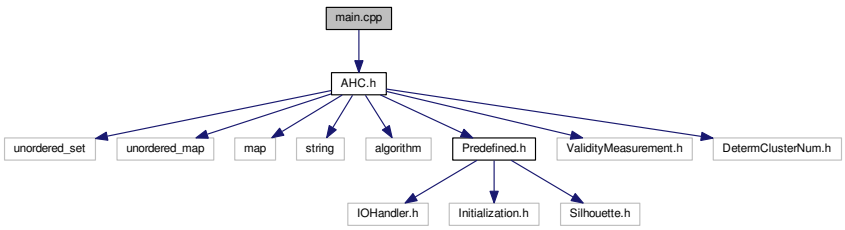
Classes

- 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*)

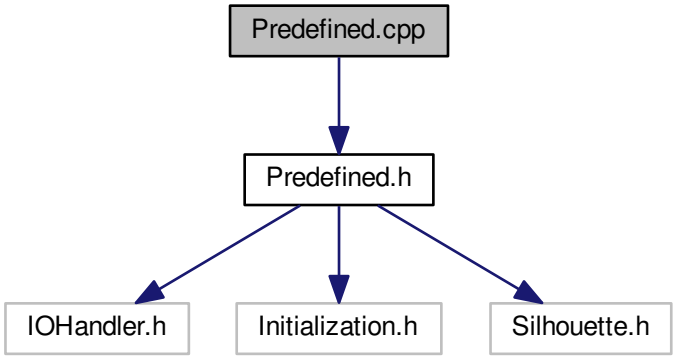
Definition at line 17 of file main.cpp.

```
18 {
19     AHC ahc(argc, argv);
20     ahc.performClustering();
21     return 0;
22 }
```

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*)

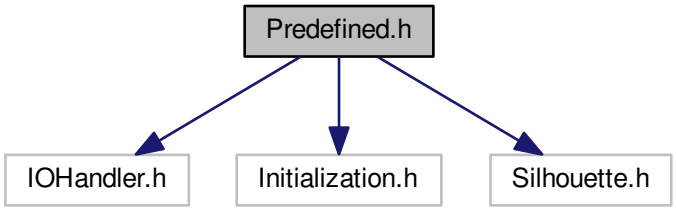
Definition at line 19 of file Predefined.cpp.

```
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 }
```

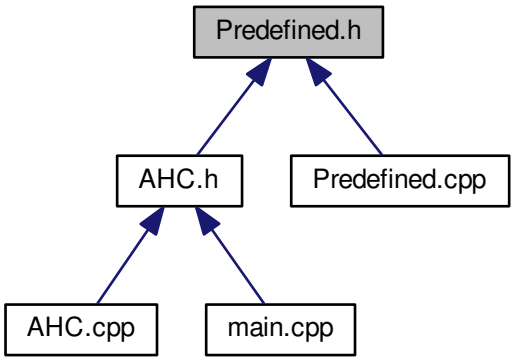
5.5 Predefined.h File Reference

```
#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
- struct DistNode

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