Affinity Propagation

The C++ implmentation for Affinity Propagation

Generated by Doxygen 1.8.11

Contents

Chapter 1

Decription of Affinity Propagation

The implementation is an O(n^3) with OpenMP and we have tested the result on the point cloud data set and compre it to the `Frey Lab webpage` linux binary version.

Two critical parameters are to be set

- **Preference value** s(i,i)
  - The preference value in `affinity propagation` and `Frey Lab webpage` is set to be the **median** of negative squared Euclidean distance between points
  - However, in flow visualization, it is set to be the **minimal similarity value** among streamlines
- **Relaxation factor** lambda
  - It controls the update rate and the default value is 0.5
- **Max iteration**
  - Due to that distance matrix for the streamline data sets is often large size ($>3000*3000$), the default value is 20

A two-level affinity propagation

Besides the conventional affinity propagation, the two-level affinity propagation is also included for user selection.

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:

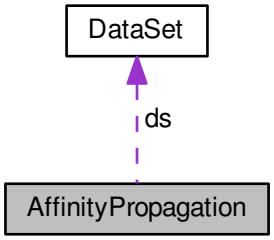
Chapter 4

Class Documentation

4.1 AffinityPropagation Class Reference

`#include <AffinityPropagation.h>`

Collaboration diagram for AffinityPropagation:



Public Member Functions

- AffinityPropagation ()
- AffinityPropagation (const int &argc, char **argv, const Para &p, bool &automatic)
- ~AffinityPropagation ()
- 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)
- void getParameterUserInput ()
- void setParameterAutomatic (const Para &p)

- void clusterByNorm (const int &norm)
- void setLabel (vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid, std::vector< int > &groupTag)
- void getEntropyRatio (const std::vector< int > &storage, float &EntropyRatio)
- void performAPClustering (Eigen::MatrixXf &matrixS, Eigen::MatrixXf &matrixR, Eigen::MatrixXf &matrixA, float **distMatrix, const Eigen::MatrixXf &coordinates)
- void getMatrixS (Eigen::MatrixXf &matrixS, float **distMatrix, const Eigen::MatrixXf &coordinates)
- void initializeMatrices (Eigen::MatrixXf &matrixS, Eigen::MatrixXf &matrixR, Eigen::MatrixXf &matrixA, const int &rows)
- void updateResponsibility (Eigen::MatrixXf &matrixR, const Eigen::MatrixXf &matrixA, const Eigen::MatrixXf &matrixS)
- void updateAvailability (Eigen::MatrixXf &matrixA, const Eigen::MatrixXf &matrixR)
- void getGroupAssignment (const Eigen::MatrixXf &matrixR, const Eigen::MatrixXf &matrixA, const Eigen↩ ::MatrixXf &matrixS, std::vector< std::vector< int > > &neighborVec, std::vector< int > &storage, std↩ ::vector< int > &groupTag)
- void getDistMatrixForCentroids (float ***centroidDistMatrix, const int &norm, const Eigen::MatrixXf &centroid)
- void getDistanceMatrixFromFile (const int &norm)
- void getHierarchicalClusters (std::vector< int > &storage, std::vector< std::vector< int > > &neighbor↩ Vec, Eigen::MatrixXf &centroid, std::vector< int > &group, const std::vector< int > &centroidGroup, const int &groupSize)

Private Attributes

- MetricPreparation object
- int normOption = -1
- std::vector< int > group
- std::vector< string > activityList
- std::vector< string > timeList
- DataSet ds
- int numberOfClusters = -1
- int extractOption = -1
- int maxIteration = -1
- bool isPBF
- bool isPathlines
- int initialOption
- bool useTwoStage

4.1.1 Detailed Description

Definition at line 54 of file AffinityPropagation.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AffinityPropagation::AffinityPropagation ( )

Definition at line 11 of file AffinityPropagation.cpp.

```
12 {
13
14 }
```

4.1.2.2 AffinityPropagation::AffinityPropagation (const int & *argc*, char ** *argv*, const Para & *p*, bool & *automatic*)

Definition at line 29 of file AffinityPropagation.cpp.

```
30 {
31     // set the data set information from the provided data set string name
32     setDataset(argc, argv);
33 
34     if(automatic)   // automate the parameter setting
35         setParameterAutomatic(p);
36 
37     else    // manually input the parameter
38         getParameterUserInput();
39 
40     /* select how to initialize the matrixS elements with preference value */
41     std::cout << "Please select a MatrixS initialization? 1.median value, 2.minimal value (recommended!)."
      << std::endl;
42     std::cin >> initialOption;
43     assert(initialOption==1||initialOption==2);
44 }
```

4.1.2.3 AffinityPropagation::~AffinityPropagation ( )

Definition at line 53 of file AffinityPropagation.cpp.

```
54 {
55     // clear the cache memory for distance matrix
56     deleteDistanceMatrix(ds.dataMatrix.rows());
57 }
```

4.1.3 Member Function Documentation

4.1.3.1 void AffinityPropagation::clusterByNorm (const int & *norm*) [private]

Definition at line 129 of file AffinityPropagation.cpp.

```
130 {
131     // The parameters to record time needed for calculation
132     struct timeval start, end;
133     double timeTemp;
134 
135     // calculate the distance matrix given the similarity measure type
136     getDistanceMatrixFromFile(norm);
137 
138     Eigen::MatrixXf matrixR, matrixA, matrixS;
139 
140     gettimeofday(&start, NULL);
141 
142     /*-------------------------First-level Affinity Propagation----------------------------*/
143 
144     // perform the AP clustering based on given distance matrix and matrix S, R and A
145     performAPClustering(matrixS, matrixR, matrixA, distanceMatrix,
      ds.dataMatrix);
146 
147     // calculate and record the time for first-level AP clustering
148     gettimeofday(&end, NULL);
149     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u+ end.tv_usec - start.tv_usec) / 1.e6;
150 
151     activityList.push_back("First-level affinity propagation takes: ");
152     timeList.push_back(to_string(timeTemp)+" s");
153 
154     // some parameters for two-level AP clustering algorithm
155     std::vector<std::vector<int> > neighborVec;
156     std::vector<int> storage;
157     Eigen::MatrixXf centroid;
158 
159     // get exemplary examples from the first-level AP
160     getGroupAssignment(matrixR, matrixA, matrixS, neighborVec, storage,
```

```
    group);
161
162     // set the labels of initial samples by first-level AP
163     setLabel(neighborVec, storage, centroid, group);
164
165     activityList.push_back("First-level affinity propagation generates: ");
166     timeList.push_back(to_string(storage.size())+" groups");
167
168     if(useTwoStage)  // two-staged AP is activated
169     {
170
171     /*----------------------Second-level Affinity Propagation ---------------------------------------
172      * Use the centroid of the first level and then apply affinipty propagation once again ----------
173      */
174         gettimeofday(&start, NULL);
175
176         /* get distance matrix for the centroids */
177         float** centroidDistMatrix = NULL;
178         getDistMatrixForCentroids(&centroidDistMatrix,
    normOption, centroid);
179
180         // perform second-level Affinity Propagation on centroids of the streamlines/pathlines
181         performAPClustering(matrixS, matrixR, matrixA, centroidDistMatrix, centroid);
182
183         // release the memory of centroidDistMatrix
184     #pragma omp parallel for schedule(static) num_threads(8)
185         for(int i=0; i<centroid.rows(); ++i)
186         {
187             delete[] centroidDistMatrix[i];
188             centroidDistMatrix[i] = NULL;
189         }
190         delete[] centroidDistMatrix;
191         centroidDistMatrix = NULL;
192
193         // record the time into the README
194         gettimeofday(&end, NULL);
195         timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u+ end.tv_usec - start.tv_usec) / 1.e6;
196         activityList.push_back("Second-level affinity propagation takes: ");
197         timeList.push_back(to_string(timeTemp)+" s");
198
199         /* extract the group information */
200         std::vector<std::vector<int> > secondNeighborVec;
201         std::vector<int> secondStorage;
202         Eigen::MatrixXf secondCentroid;
203
204         std::vector<int> centroidGroup(centroid.rows());
205
206         /* get exemplary examples */
207         getGroupAssignment(matrixR, matrixA, matrixS, secondNeighborVec, secondStorage,
    centroidGroup);
208
209         // get the label of each candidate lines by two-level AP clustering
210         setLabel(secondNeighborVec, secondStorage, secondCentroid, centroidGroup);
211
212         secondNeighborVec.clear();
213
214         // record the consumed time
215         activityList.push_back("Second-level affinity propagation generates: ");
216         timeList.push_back(to_string(secondStorage.size())+" groups");
217
218     /*------------------------ Get the true group id by hierarchical affinity propagation -----------------
    */
219         // should re-calculate the centroid, storage and neighborVec for new clusters
220         getHierarchicalClusters(storage, neighborVec, centroid,
    group, centroidGroup, secondStorage.size());
221     }
222
223     // begin to calculate the evaluation metrics and cluster representatives
224     extractFeatures(storage, neighborVec, centroid);
225
226 }
```

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

Definition at line 301 of file AffinityPropagation.cpp.

```
303 {
304     const int& Row = ds.dataMatrix.rows();
```

```
305     const int& Column = ds.dataMatrix.cols();
306
307     /* record labeling information */
308     // IOHandler::generateGroups(neighborVec);
309
310     // Output the number of candidates inside each streamline cluster
311     std::cout << "Final group number information: " << std::endl;
312     for (int i = 0; i < storage.size(); ++i)
313     {
314         std::cout << storage[i] << " ";
315     }
316     std::cout << std::endl;
317
318     // calculate the normalized entropy to check the balance of cluster size
319     float EntropyRatio;
320     getEntropyRatio(storage, EntropyRatio);
321
322     // print the cluster labels in the primary .vtk file
323     IOHandler::printClusters(ds.dataVec,group,storage,"AP_norm"+to_string(
    normOption),ds.fullName,ds.dimension);
324
325     struct timeval start, end;
326     double timeTemp;
327
328     /* compute the centroid coordinates of each clustered group */
329
330     gettimeofday(&start, NULL);
331
332     vector<vector<float> > closest(numberOfClusters);
333     vector<vector<float> > furthest(numberOfClusters);
334
335     /* extract the closest and furthest streamlines to centroid */
336 #pragma omp parallel for schedule(static) num_threads(8)
337     for (int i=0;i<numberOfClusters;++i)
338     {
339         float minDist = FLT_MAX;
340         float maxDist = -10;
341         int minIndex = -1, maxIndex = -1;
342         const std::vector<int>& groupRow = neighborVec[i];
343         const Eigen::VectorXf& eachCentroid = centroid.row(i);
344         for (int j = 0; j < groupRow.size(); ++j)
345         {
346             float distance = getDisimilarity(eachCentroid,ds.dataMatrix,groupRow[j],
    normOption,object);
347             if(minDist>distance)
348             {
349                 minDist = distance;
350                 minIndex = groupRow[j];
351             }
352             if(maxDist<distance)
353             {
354                 maxDist = distance;
355                 maxIndex = groupRow[j];
356             }
357         }
358         closest[i] = ds.dataVec[minIndex];
359         furthest[i] = ds.dataVec[maxIndex];
360     }
361
362     // convert the centroid matrix into vector<vector<float>> type. It is not necessary actually
363     std::vector<std::vector<float> > center_vec(numberOfClusters, vector<float>(Column));
364 #pragma omp parallel for schedule(static) num_threads(8)
365     for (int i = 0; i < center_vec.size(); ++i)
366     {
367         for (int j = 0; j < Column; ++j)
368         {
369             center_vec[i][j] = centroid(i,j);
370         }
371     }
372
373     // Record the time for extracting the cluster representative lines
374     gettimeofday(&end, NULL);
375     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
376                 + end.tv_usec - start.tv_usec) / 1.e6;
377     activityList.push_back("Feature extraction takes: ");
378     timeList.push_back(to_string(timeTemp)+" s");
379
380     // calculate the normalized validity measurement metric for clustering evaluation
381     ValidityMeasurement vm;
382     vm.computeValue(normOption, ds.dataMatrix, group, object,
    isPBF);
383     activityList.push_back("Validity measure is: ");
384     stringstream fc_ss;
385     fc_ss << vm.f_c;
386     timeList.push_back(fc_ss.str());
387
388     std::cout << "Finishing extracting features!" << std::endl;
```

```
389
390     // calculate silhouette, the Gamma statistics and DB index for clustering evaluation
391     gettimeofday(&start, NULL);
392     Silhouette sil;
393     sil.computeValue(normOption,ds.dataMatrix,ds.
    dataMatrix.rows(),ds.dataMatrix.cols(),group,object,
394                      numberOfClusters, isPBF, neighborVec);
395     gettimeofday(&end, NULL);
396     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
397               + end.tv_usec - start.tv_usec) / 1.e6;
398     activityList.push_back("Silhouette calculation takes: ");
399     timeList.push_back(to_string(timeTemp)+" s");
400
401     stringstream ss;
402     ss << "norm_" << normOption;
403
404     /* measure closest and furthest rotation */
405     std::vector<float> closestRotation, furthestRotation;
406     const float& closestAverage = getRotation(closest, closestRotation);
407     const float& furthestAverage = getRotation(furthest, furthestRotation);
408
409     /* save closest, furthest and centroid representative streamlines */
410     IOHandler::printFeature(ds.dataName+"_AP_closest_"+ss.str()+".vtk", closest, sil.sCluster,
411             closestRotation, ds.dimension);
412     IOHandler::printFeature(ds.dataName+"_AP_furthest_"+ss.str()+".vtk", furthest, sil.sCluster,
413             furthestRotation, ds.dimension);
414     IOHandler::printFeature(ds.dataName+"_AP_centroid_"+ss.str()+".vtk", center_vec, sil.sCluster
    ,ds.dimension);
415
416     IOHandler::printToFull(ds.dataVec, sil.sData, "AP_SValueLine_"+ss.str(),
    ds.fullName, ds.dimension);
417     IOHandler::printToFull(ds.dataVec, group, sil.sCluster, "AP_SValueCluster_"+ss.str(),
    ds.fullName, ds.dimension);
418
419     // record the clustering evaluation metric values in the txt file
420     activityList.push_back("numCluster is: ");
421     timeList.push_back(to_string(numberOfClusters));
422
423     activityList.push_back("Norm option is: ");
424     timeList.push_back(to_string(normOption));
425
426     IOHandler::generateReadme(activityList,timeList);
427
428     /* print entropy value for the clustering algorithm */
429     IOHandler::writeReadme(EntropyRatio, sil, "For norm "+to_string(normOption));
430
431     IOHandler::writeReadme(closestAverage, furthestAverage);
432 }
```

4.1.3.3 void AffinityPropagation::getDistanceMatrixFromFile (const int & *norm*) [private]

Definition at line 870 of file AffinityPropagation.cpp.

```
871 {
872     normOption = norm;
873
874     /* very hard to decide whether needed to perform such pre-processing, but recommended
875      * to create a cached object for further pair-wise distance matrix calculation
876      */
877     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
878     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
    ds.dataMatrix.cols(), normOption);
879
880     /* would store distance matrix instead because it would save massive time */
881     struct timeval start, end;
882     double timeTemp;
883     gettimeofday(&start, NULL);
884
885     // in case the distance matrix already exists for other similarity, will clean it first
886     deleteDistanceMatrix(ds.dataMatrix.rows());
887
888     // read distance matrix from the local file in ../dataset/
889     std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
890
891     // the local file of distance matrix does not exist, then will create the file
892     if(distFile.fail())
893     {
894         distFile.close();
895         // calculate the distance matrix from norm option
896         getDistanceMatrix(ds.dataMatrix, normOption, object);
```

```
897         std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
898         for(int i=0;i<ds.dataMatrix.rows();++i)
899         {
900             for(int j=0;j<ds.dataMatrix.rows();++j)
901             {
902                 distFileOut << distanceMatrix[i][j] << " ";
903             }
904             distFileOut << std::endl;
905         }
906         distFileOut.close();
907     }
908     else // the local file for distance matrix computation exists, then directly read in
909     {
910         std::cout << "read distance matrix..." << std::endl;
911 
912         // create the distance matrix and read in the content
913         distanceMatrix = new float*[ds.dataMatrix.rows()];
914     #pragma omp parallel for schedule(static) num_threads(8)
915         for (int i = 0; i < ds.dataMatrix.rows(); ++i)
916         {
917             distanceMatrix[i] = new float[ds.dataMatrix.rows()];
918         }
919         int i=0, j;
920         string line;
921         stringstream ss;
922         // extract the distance values from the file
923         while(getline(distFile, line))
924         {
925             j=0;
926             ss.str(line);
927             while(ss>>line)
928             {
929                 if(i==j)
930                     distanceMatrix[i][j]=0;
931                 else
932                     distanceMatrix[i][j] = std::atof(line.c_str());
933                 ++j;
934             }
935             ++i;
936             ss.str("");
937             ss.clear();
938         }
939         distFile.close();
940     }
941 
942     gettimeofday(&end, NULL);
943     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
944                 + end.tv_usec - start.tv_usec) / 1.e6;
945     activityList.push_back("Distance matrix computing for norm "+to_string(
      normOption)+" takes: ");
946     timeList.push_back(to_string(timeTemp)+" s");
947 }
```

4.1.3.4 void AffinityPropagation::getDistMatrixForCentroids (float *** *centroidDistMatrix*, const int & *norm*, const Eigen::MatrixXf & *centroid*) [private]

Definition at line 838 of file AffinityPropagation.cpp.

```
840 {
841     const int& rows = centroid.rows();
842     *centroidDistMatrix = new float*[rows];
843 
844     /* in order to calculate the distance matrix given norm, we need to calculate the object first. This
      object
845      * is to pre-calculate some preliminary stuff for distance matrix computation. I know it is redundant
      but in
846      * practice it can help to accelerate the performance a little bit
847      */
848 
849     MetricPreparation centroidObj = MetricPreparation(centroid.rows(), centroid.cols());
850     centroidObj.preprocessing(centroid, centroid.rows(), centroid.cols(), norm);
851 
852     // calculate the distance matrix among centroid matrix coordinates
853 #pragma omp parallel for schedule(static) num_threads(8)
854     for(int i=0; i<rows; ++i)
855     {
856         (*centroidDistMatrix)[i] = new float[rows];
857         for(int j=0; j<rows; ++j)
858         {
```

```
859             (*centroidDistMatrix)[i][j] = getDisimilarity(centroid, i, j, norm, centroidObj);
860         }
861     }
862 }
```

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

Definition at line 485 of file AffinityPropagation.cpp.

```
486 {
487     // the formula is -s[i]/S * log(s[i]/S), and then normalized by log(numOfClusters)
488     EntropyRatio = 0;
489     const int& Row = ds.dataMatrix.rows();
490     for (int i = 0; i < storage.size(); ++i)
491     {
492         float ratio = float(storage[i])/float(Row);
493         EntropyRatio-=ratio*log2f(ratio);
494     }
495     /* the higher value shows that the final clusters are balanced and almost equal sized, while the
496         low value shows the contrary
497     */
498     EntropyRatio/=log2f(storage.size());
499 }
```

4.1.3.6 void AffinityPropagation::getGroupAssignment (const Eigen::MatrixXf & *matrixR*, const Eigen::MatrixXf & *matrixA*, const Eigen::MatrixXf & *matrixS*, std::vector< std::vector< int > > & *neighborVec*, std::vector< int > & *storage*, std::vector< int > & *groupTag*) [private]

Definition at line 757 of file AffinityPropagation.cpp.

```
760 {
761     std::vector<int> centerVec;
762     const int& rows = matrixR.rows();
763
764     /* store the candidate whose diagonal summation is positive */
765     float diagonalSum;
766     for(int i=0;i<rows;++i)
767     {
768         diagonalSum=matrixR(i,i)+matrixA(i,i);
769         if(diagonalSum>0)
770         {
771             centerVec.push_back(i);
772         }
773     }
774
775     const int& centerSize = centerVec.size();
776     /* get group tag information for each candidate streamline */
777 #pragma omp parallel for schedule(static) num_threads(8)
778     for(int i=0;i<rows;++i)
779     {
780         int index, element;
781         float maxSim = -FLT_MAX;
782         for(int j=0;j<centerSize;++j)
783         {
784             element = centerVec[j];
785             if(matrixS(i,element)>maxSim)
786             {
787                 maxSim = matrixS(i,element);
788                 index = element;
789             }
790         }
791         groupTag[i]=index;
792     }
793
794     /* output group information and cluster size */
795     std::map<int,int> groupMap;
796     for(int i=0;i<rows;++i)
797     {
798         /* group tag not int the hash map */
799         if(groupMap.find(groupTag[i])==groupMap.end())
```

```
800         {
801             groupMap.insert(make_pair(groupTag[i],0));
802         }
803     }
804 
805     /* give them new index starting from 0 */
806     int count = 0;
807     for(auto iter = groupMap.begin();iter!=groupMap.end();++iter)
808     {
809         iter->second = count++;
810     }
811 
812     numberOfClusters = groupMap.size();
813 
814     /* assign contained element and size */
815     neighborVec = std::vector<std::vector<int> >(numberOfClusters);
816     storage = std::vector<int>(numberOfClusters);
817     for(int i=0;i<rows;++i)
818     {
819         count = groupMap[group[i]];
820         neighborVec[count].push_back(i);
821     }
822 
823     /* assign the storage vector */
824     for(int i=0;i<storage.size();++i)
825     {
826         storage[i] = neighborVec[i].size();
827     }
828 }
```

4.1.3.7 void AffinityPropagation::getHierarchicalClusters (std::vector< int > & *storage*, std::vector< std::vector< int > > & *neighborVec*, Eigen::MatrixXf & *centroid*, std::vector< int > & *group*, const std::vector< int > & *centroidGroup*, const int & *groupSize*) [private]

Definition at line 993 of file AffinityPropagation.cpp.

```
996 {
997     neighborVec.clear();
998     neighborVec.resize(groupSize);
999     storage.resize(groupSize);
1000     centroid = Eigen::MatrixXf::Zero(groupSize, centroid.cols());
1001 
1002     int groupID;
1003     for(int i=0; i<groupTag.size(); ++i)
1004     {
1005         groupID = centroidGroup[groupTag[i]];
1006         groupTag[i] = groupID;
1007         neighborVec[groupID].push_back(i);
1008         centroid.row(groupID)+=ds.dataMatrix.row(i);
1009     }
1010 
1011 #pragma omp parallel for schedule(static) num_threads(8)
1012     for(int i=0; i<groupSize; ++i)
1013     {
1014         centroid.row(i)/=neighborVec[i].size();
1015         storage[i] = neighborVec[i].size();
1016     }
1017 }
```

4.1.3.8 void AffinityPropagation::getMatrixS (Eigen::MatrixXf & *matrixS*, float ** *distMatrix*, const Eigen::MatrixXf & *coordinates*) [private]

Definition at line 584 of file AffinityPropagation.cpp.

```
585 {
586     std::cout << "Start initializing matrix S..." << std::endl;
587 
588     const int& rows = matrixS.rows();
589 
590     /* define a vector to store pair-wise distance vector and get the median */
591     const int& distVecSize = rows*(rows-1)/2;
```

```
592     std::vector<float> distVec(distVecSize);
593     int count = 0;
594 
595     /* find the minimal dissimilarity value from the distance matrix */
596     float minV = (float)FLT_MAX;
597     float tempDist;
598     for(int i=0;i<rows-1;++i)
599     {
600         for(int j=i+1;j<rows;++j)
601         {
602             if(distMatrix)  // if distance matrix exists, direct fetch the cached value
603                 tempDist = distMatrix[i][j];
604             else    // otherwise, has to calculate the distance matrix
605                 tempDist = getDisimilarity(coordinates, i, j, normOption, object);
606 
607             /* conventionally we assign -d*d as non-diagonal entries for matrix S */
608             matrixS(i,j) = -tempDist;
609             matrixS(j,i) = matrixS(i,j);
610 
611             minV = std::min(minV, matrixS(i,j));
612             distVec[count++] = matrixS(i,j);
613         }
614     }
615 
616     std::cout << "min Value is " << minV << std::endl;
617     assert(count==distVecSize);
618 
619     float initialValue;
620     if(initialOption==1)   // the initialization is by median of distance matrix values
621     {
622         /* get median value to be assigned for S(i,i) */
623         float medianValue, leftMedian, rightMedian;
624 
625         /* odd size, just pick mid index */
626         if(distVecSize%2==1)
627             medianValue = select(distVec, 0, distVecSize-1, distVecSize/2);
628         /* even size, choose average of left and right */
629         else if(distVecSize%2==0)
630         {
631             leftMedian = select(distVec, 0, distVecSize-1, (distVecSize-1)/2);
632             rightMedian = select(distVec, 0, distVecSize-1, distVecSize/2);
633             medianValue = (leftMedian+rightMedian)/2.0;
634         }
635         // assign the preference value as median of the distance matrix values
636         initialValue = medianValue;
637     }
638     else if(initialOption==2)  // the initialization is by minimal dissimilarity value
639     {
640         initialValue = minV;
641     }
642     std::cout << "Initial value is " << initialValue << std::endl;
643 
644     /* assign the initialValue to diagonal matrix element */
645 #pragma omp parallel for schedule(static) num_threads(8)
646     for(int i=0;i<rows;++i)
647         matrixS(i,i) = initialValue;
648 
649     std::cout << "Finish initializing matrix S..." << std::endl;
650 }
```

4.1.3.9 void AffinityPropagation::getParameterUserInput ( ) [private]

Definition at line 541 of file AffinityPropagation.cpp.

```
542 {
543     // User input for streamline/pathline sampleOption
544     int sampleOption;
545     std::cout << "choose a sampling method for the dataset?" << std::endl
546               << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
547     std::cin >> sampleOption;
548     assert(sampleOption==1||sampleOption==2);
549 
550     if(isPathlines)  // if is pathlines, directly repeat the last vertex of pathlines
551         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
552     else    // for streamlines, there are multiple options for that
553     {
554         if(sampleOption==1) // direct repeat the last vertex
555             IOHandler::expandArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
```

```
556         else if(sampleOption==2)    // sample the array on the intervals
557             IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
558         else if(sampleOption==3)    // sample the array with equal arc
559             IOHandler::uniformArcSampling(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
560     }
561
562     group = std::vector<int>(ds.dataMatrix.rows());
563
564     // select cluster represnetative strategy, and 1 is recommended
565     std::cout << "Select extraction method: 1.centroid, closest and furthest (recommended!), 2.median."
566             << std::endl;
567     std::cin >> extractOption;
568     assert(extractOption==1||extractOption==2);
569
570     // Input the maximal iteration for AP clustering algorithm
571     std::cout << "Input max iteration for affinity propagation: " << std::endl;
572     std::cin >> maxIteration;
573     assert(maxIteration>0);
574 }
```

4.1.3.10 void AffinityPropagation::initializeMatrices (Eigen::MatrixXf & *matrixS*, Eigen::MatrixXf & *matrixR*, Eigen::MatrixXf & *matrixA*, const int & *rows*) [private]

Definition at line 661 of file AffinityPropagation.cpp.

```
663 {
664     /* initialize all three matrices as zero entry */
665     matrixS = Eigen::MatrixXf::Zero(rows, rows);
666     matrixR = Eigen::MatrixXf::Zero(rows, rows);
667     matrixA = Eigen::MatrixXf::Zero(rows, rows);
668 }
```

4.1.3.11 void AffinityPropagation::performAPClustering (Eigen::MatrixXf & *matrixS*, Eigen::MatrixXf & *matrixR*, Eigen::MatrixXf & *matrixA*, float ** *distMatrix*, const Eigen::MatrixXf & *coordinates*) [private]

Definition at line 959 of file AffinityPropagation.cpp.

```
961 {
962     /* initialize S, R, A */
963     initializeMatrices(matrixS, matrixR, matrixA, coordinates.rows());
964
965     /* get S */
966     getMatrixS(matrixS, distMatrix, coordinates);
967
968     int current = 0;
969     while(current++<maxIteration)
970     {
971         std::cout << "Iteration " << current << std::endl;
972
973         /* update responsibility */
974         updateResponsibility(matrixR, matrixA, matrixS);
975
976         /* update availability */
977         updateAvailability(matrixA, matrixR);
978
979     }
980 }
```

4.1.3.12 void AffinityPropagation::performClustering ( )

Definition at line 67 of file AffinityPropagation.cpp.

```
68 {
69     //distance metric type
70     /*  0: Euclidean Norm, d(a,b) = (\sum_(a-b)^2)^(1/2).
71         1: Fraction Distance Metric, d(a,b) = (\sum_(a-b)^p)^(1/p), we choose p==0.5
72         2: piece-wise angle average, from http://www2.cs.uh.edu/~chengu/Publications/3DFlowVis/
    curveClustering.pdf
73         3: Bhattacharyya metric for rotation
74         4: average rotation
75         5: signed-angle intersection
76         6: normal-direction multivariate distribution
77         7: Bhattacharyya metric with angle to a fixed direction
78         8: Piece-wise angle average \times standard deviation
79         9: normal-direction multivariate un-normalized distribution
80         10: x*y/|x||y| borrowed from machine learning
81         11: cosine similarity
82         12: Mean-of-closest point distance (MCP)
83         13: Hausdorff distance min_max(x_i,y_i)
84         14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
85         15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
86         16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
87         17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
    S0097849318300128
88             for pathlines only
89     */
90
91     for(int i=0;i<=17;++i)
92     {
93         if(isPathlines)  // for pathlines, it will call similarity measure d_T (17)
94         {
95             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15 && i!=17)
96                 continue;
97         }
98         else    // for streamlines, d_T (17) will not be involved
99         {
100             /* don't want to deal with many too naive metrics */
101             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
102                 continue;
103         }
104
105         std::cout << "-----------------------------------------------------" << std::endl;
106         std::cout << "Experiment on norm " << i << " starts!--------------" << std::endl;
107
108         // clear out the recorded string information
109         activityList.clear();
110         timeList.clear();
111
112         // perform clustering on the selected similarity measure i
113         clusterByNorm(i);
114
115         std::cout << std::endl;
116     }
117 }
```

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

Definition at line 441 of file AffinityPropagation.cpp.

```
442 {
443     // the argc should be 3, e.g., ./ap cylinder 3
444     if(argc!=3)
445     {
446         std::cout << "Input argument should have 3!" << endl
447                   << "./cluster inputFile_name(in dataset folder) "
448                   << "data_dimension(3)" << endl;
449         exit(1);
450     }
451
452     // extract the required information from argument string
453     ds.strName = string("../dataset/")+string(argv[1]);
454     ds.dataName = string(argv[1]);
455     ds.dimension = atoi(argv[2]);
456
```

```
457     /* get the bool tag for variable isPBF */
458     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
459     int PBFjudgement;
460     std::cin >> PBFjudgement;
461     assert(PBFjudgement==1||PBFjudgement==0);
462     isPBF = (PBFjudgement==1);
463 
464     /* check whether it is a Pathline data set or not */
465     std::cout << "It is a Pathline? 1.Yes, 0. No" << std::endl;
466     std::cin >> PBFjudgement;
467     assert(PBFjudgement==1||PBFjudgement==0);
468     isPathlines = (PBFjudgement==1);
469 
470     // read from the file into the member variables
471     IOHandler::readFile(ds.strName,ds.dataVec,ds.vertexCount,
      ds.dimension,ds.maxElements);
472 
473     // print the streamline/pathline vtk file
474     ds.fullName = ds.strName+"_full.vtk";
475     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
      vertexCount, ds.dimension);
476 }
```

4.1.3.14 void AffinityPropagation::setLabel (vector< vector< int > > & *neighborVec*, vector< int > & *storage*, Eigen::MatrixXf & *centroid*, std::vector< int > & *groupTag*) [private]

Definition at line 241 of file AffinityPropagation.cpp.

```
243 {
244     // record the pair {cluster size, cluster candidate index}
245     std::vector<Ensemble> nodeVec;
246 
247     for(int i=0;i<storage.size();++i)
248     {
249         if(storage[i]==0)
250             continue;
251         nodeVec.push_back({storage[i], neighborVec[i]});
252     }
253 
254     numberOfClusters = nodeVec.size();
255 
256     std::cout << "Cluster label setting begins with " << nodeVec.size() << " clusters..." << std::endl;
257 
258     /* sort group index by size of elements containd inside to make sure that, 0 cluster has the
259      * smallest size of candidates
260      */
261     std::sort(nodeVec.begin(), nodeVec.end(), [](const Ensemble& first, const
      Ensemble& second)
262     {return first.size<second.size|| (first.size==second.size&&first.
      element[0]<second.element[0]);});
263 
264     // re-define the neighborVec, storage and centroid coordinates given the new cluster index
265     neighborVec = std::vector<std::vector<int> >(nodeVec.size());
266     storage = std::vector<int>(nodeVec.size());
267     centroid = Eigen::MatrixXf(nodeVec.size(), ds.dataMatrix.cols());
268 
269     // re-calculate the coordinates of the cluster centroids
270 #pragma omp parallel for schedule(static) num_threads(8)
271     for(int i=0;i<nodeVec.size();++i)
272     {
273         neighborVec[i] = nodeVec[i].element;
274         storage[i] = nodeVec[i].size;
275         Eigen::VectorXf tempVec = Eigen::VectorXf::Zero(ds.dataMatrix.cols());
276         for(int j=0;j<storage[i];++j)
277         {
278             tempVec+=ds.dataMatrix.row(i).transpose();
279             /* don't forget to re-compute the group tag */
280             groupTag[neighborVec[i][j]]=i;
281         }
282         centroid.row(i) = tempVec/storage[i];
283     }
284 
285     std::cout << "Cluster label setting ends..." << std::endl;
286 }
```

4.1.3.15 void AffinityPropagation::setParameterAutomatic (const Para & *p*) [private]

Definition at line 507 of file AffinityPropagation.cpp.

```
508 {
509     // if the data set is pathline, will direct expand the array on the back
510     if(isPathlines)
511         IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.
    dimension,ds.maxElements);
512     else    // it is streamline
513     {
514         if(p.sampled==1) // sampling is to directly expand the array from the back
515             IOHandler::expandArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
516         else if(p.sampled==2)    // sample the array on the intervals without change of geometric
     shape
517             IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
518         else if(p.sampled==3)    // sample the array with equal arcs such that
519             IOHandler::uniformArcSampling(ds.dataMatrix,ds.dataVec,
    ds.dimension,ds.maxElements);
520     }
521
522     // ceate a label vector for each candidate line
523     group = std::vector<int>(ds.dataMatrix.rows());
524
525     // assign the parameters for AP clustering
526     extractOption = p.extractOption;
527     maxIteration = p.maxIteration;
528
529     /* whether to activate two-staged AP or not, see Jun Tao FlowString TVCG 2016 paper for details */
530     std::cout << "Whether to activate two-staged AP or not? 1.Yes, 2.No," << std::endl;
531     int twoStageOption;
532     std::cin >> twoStageOption;
533     assert(twoStageOption==1 || twoStageOption==2);
534     useTwoStage = (twoStageOption==1);
535 }
```

4.1.3.16 void AffinityPropagation::updateAvailability (Eigen::MatrixXf & *matrixA*, const Eigen::MatrixXf & *matrixR*) [private]

Definition at line 709 of file AffinityPropagation.cpp.

```
710 {
711     const int& rows = matrixR.rows();
712 #pragma omp parallel for schedule(static) num_threads(8)
713     for(int i=0;i<rows;++i)
714     {
715         for(int k=0;k<rows;++k)
716         {
717             /* for diagonal matrix, update by summation of non-diagonal entries in the row */
718             if(i==k)
719             {
720                 float summation = 0.0;
721                 for(int ii=0;ii<rows;++ii)
722                 {
723                     if(ii==i)
724                         continue;
725                     summation+=std::max((float)0.0, matrixR(ii,k));
726                 }
727
728                 /* smoothing update instead of direct assignment */
729                 matrixA(i,k)=(1-LAMBDA)*summation+LAMBDA*matrixA(i,k);
730             }
731             else
732             {
733                 float summation = 0.0;
734                 for(int ii=0;ii<rows;++ii)
735                 {
736                     if(ii==i||ii==k)
737                         continue;
738                     summation+=std::max((float)0.0, matrixR(ii,k));
739                 }
740                 matrixA(i,k)=(1-LAMBDA)*std::min((float)0.0, matrixR(k,k)+summation)+
    LAMBDA*matrixA(i,k);
741             }
742         }
743     }
744 }
```

4.1.3.17 void AffinityPropagation::updateResponsibility (Eigen::MatrixXf & *matrixR,* const Eigen::MatrixXf & *matrixA,* const Eigen::MatrixXf & *matrixS*) [private]

Definition at line 678 of file AffinityPropagation.cpp.

```
680 {
681     const int& rows = matrixR.rows();
682     // update the R with relaxed value of S and R
683 #pragma omp parallel for schedule(static) num_threads(8)
684     for(int i=0;i<rows;++i)
685     {
686         for(int k=0;k<rows;++k)
687         {
688             /* don't use FLT_MIN because FLT_MIN == 0.0 */
689             float maxValue = -FLT_MAX;
690             for(int kk=0;kk<rows;++kk)
691             {
692                 if(kk==k)
693                     continue;
694                 maxValue = std::max(maxValue, matrixS(i,kk)+matrixA(i,kk));
695             }
696             /* in wikipage it's update by R[i,k] = S[i][k]-maxValue, but here use a Laplace smoothor for
    convergence */
697             matrixR(i,k) = (1-LAMBDA)*(matrixS(i,k)-maxValue)+LAMBDA*matrixR(i,k);
698         }
699     }
700 }
```

4.1.4 Member Data Documentation

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

Definition at line 119 of file AffinityPropagation.h.

4.1.4.2 DataSet AffinityPropagation::ds [private]

Definition at line 129 of file AffinityPropagation.h.

4.1.4.3 int AffinityPropagation::extractOption = -1 [private]

Definition at line 139 of file AffinityPropagation.h.

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

Definition at line 114 of file AffinityPropagation.h.

4.1.4.5 int AffinityPropagation::initialOption [private]

Definition at line 159 of file AffinityPropagation.h.

4.1.4.6 bool AffinityPropagation::isPathlines [private]

Definition at line 154 of file AffinityPropagation.h.

4.1.4.7 bool AffinityPropagation::isPBF [private]

Definition at line 149 of file AffinityPropagation.h.

4.1.4.8 int AffinityPropagation::maxIteration = -1 [private]

Definition at line 144 of file AffinityPropagation.h.

4.1.4.9 int AffinityPropagation::normOption = -1 [private]

Definition at line 109 of file AffinityPropagation.h.

4.1.4.10 int AffinityPropagation::numberOfClusters = -1 [private]

Definition at line 134 of file AffinityPropagation.h.

4.1.4.11 MetricPreparation AffinityPropagation::object [private]

Definition at line 104 of file AffinityPropagation.h.

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

Definition at line 124 of file AffinityPropagation.h.

4.1.4.13 bool AffinityPropagation::useTwoStage [private]

Definition at line 164 of file AffinityPropagation.h.

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

- AffinityPropagation.h
- AffinityPropagation.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 17 of file Predefined.h.

4.2.2 Member Data Documentation

4.2.2.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 20 of file Predefined.h.

4.2.2.2 string DataSet::dataName

Definition at line 27 of file Predefined.h.

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

Definition at line 19 of file Predefined.h.

4.2.2.4 int DataSet::dimension = -1

Definition at line 23 of file Predefined.h.

4.2.2.5 string DataSet::fullName

Definition at line 26 of file Predefined.h.

4.2.2.6 int DataSet::maxElements = -1

Definition at line 21 of file Predefined.h.

4.2.2.7 string DataSet::strName

Definition at line 25 of file Predefined.h.

4.2.2.8 int DataSet::vertexCount = -1

Definition at line 22 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 Attributes

- int size
- std::vector< int > element

4.3.1 Detailed Description

Definition at line 35 of file Predefined.h.

4.3.2 Member Data Documentation

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

Definition at line 38 of file Predefined.h.

4.3.2.2 int Ensemble::size

Definition at line 37 of file Predefined.h.

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

- Predefined.h

4.4 Para Struct Reference

`#include <AffinityPropagation.h>`

Public Attributes

- int sampled
- int extractOption
- int maxIteration

4.4.1 Detailed Description

Definition at line 31 of file AffinityPropagation.h.

4.4.2 Member Data Documentation

4.4.2.1 int Para::extractOption

Definition at line 42 of file AffinityPropagation.h.

4.4.2.2 int Para::maxIteration

Definition at line 47 of file AffinityPropagation.h.

4.4.2.3 int Para::sampled

Definition at line 37 of file AffinityPropagation.h.

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

- AffinityPropagation.h

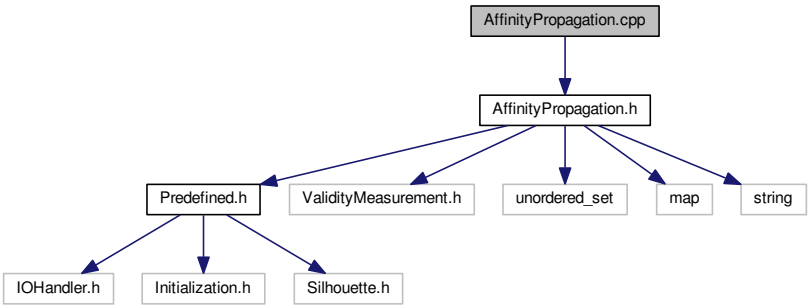
Chapter 5

File Documentation

5.1 AffinityPropagation.cpp File Reference

```
#include "AffinityPropagation.h"
```

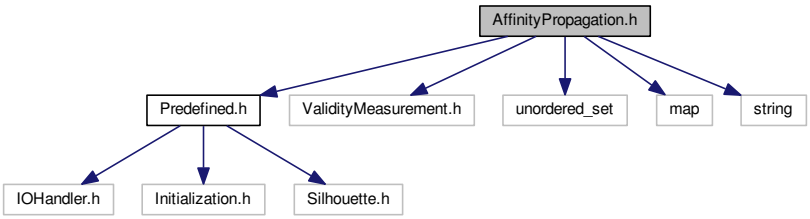
Include dependency graph for AffinityPropagation.cpp:



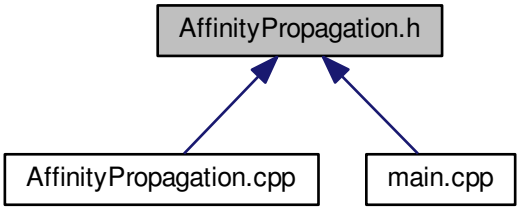
5.2 AffinityPropagation.h File Reference

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

Include dependency graph for AffinityPropagation.h:



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



Classes

- struct Para
- class AffinityPropagation

Macros

- #define LAMBDA 0.5

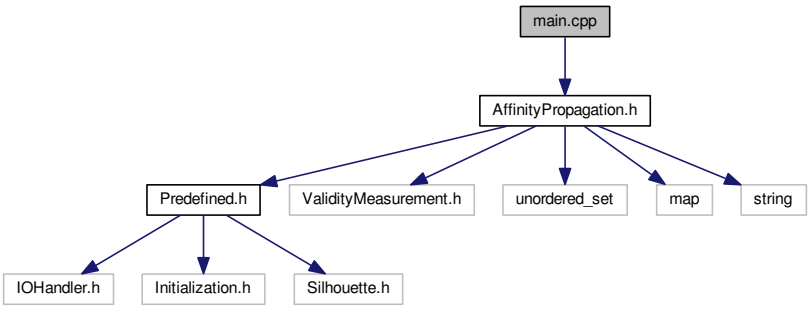
5.2.1 Macro Definition Documentation

5.2.1.1 #define LAMBDA 0.5

Definition at line 21 of file AffinityPropagation.h.

5.3 main.cpp File Reference

`#include "AffinityPropagation.h"`
Include dependency graph for main.cpp:



Functions

- void setPara (Para &p)
- int main (int argc, char **argv)

5.3.1 Function Documentation

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

Definition at line 21 of file main.cpp.

```
22 {
23     Para p;
24
25     setPara(p);
26
27     /* enable automatic option */
28     bool automatic = true;
29
30     AffinityPropagation ap(argc, argv, p, automatic);
31
32     ap.performClustering();
33
34     return 0;
35 }
```

5.3.1.2 void setPara (Para & *p*)

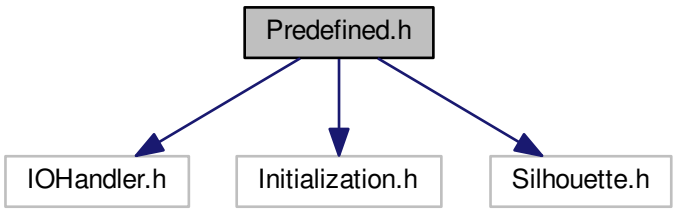
Definition at line 43 of file main.cpp.

```
44 {
45     /* 1.directly filling with last vertex; 2. uniform sampling, 3. equal-arc sampling */
46     p.sampled = 2;
47
48     /* extraction option, 1. centroid, closest and furthest, 2. median, 3. statistical representation */
49     p.extractOption = 1;
50
51     /* max iteration for AP clustering */
52     p.maxIteration = 20;
53
54 }
```

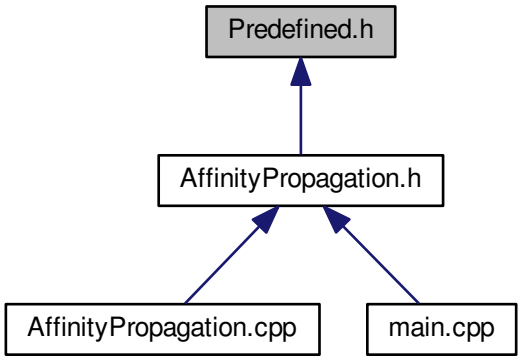
5.4 Predefined.h File Reference

```
#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 DataSet
- struct Ensemble

Functions

- template<class T >
  void mySwap (T &a, T &b)
- template<class T >
  int partition (std::vector< T > &array, const int &left, const int &right, const int &pivotIndex)
- template<class T >
  T select (std::vector< T > &array, int left, int right, const int &k)

5.4.1 Function Documentation

5.4.1.1 template<class T > void mySwap (T & *a,* T & *b*)

Definition at line 45 of file Predefined.h.

```
46 {
47     T temp = a;
48     a = b;
49     b = temp;
50 }
```

5.4.1.2 template<class T > int partition (std::vector< T > & *array,* const int & *left,* const int & *right,* const int & *pivotIndex*)

Definition at line 62 of file Predefined.h.

```
63 {
64     T pivotValue = array[pivotIndex];
65     mySwap(array[pivotIndex], array[right]);
66     int storeIndex = left;
67     for(int i=left; i<right;++i)
68     {
69         if(array[i]<pivotValue)
70         {
71             mySwap(array[storeIndex], array[i]);
72             ++storeIndex;
73         }
74     }
75     mySwap(array[right], array[storeIndex]);
76     return storeIndex;
77 }
```

5.4.1.3 template<class T > T select (std::vector< T > & *array,* int *left,* int *right,* const int & *k*)

Definition at line 91 of file Predefined.h.

```
92 {
93     int pivotIndex;
94     while(true)
95     {
96         if(left==right)
97             return array[left];
98         pivotIndex = (left+right)/2;
99         pivotIndex = partition(array, left, right, pivotIndex);
100         if(k==pivotIndex)
101             return array[k];
102         else if(k<pivotIndex)
103             right = pivotIndex-1;
104         else
105             left = pivotIndex+1;
106     }
107 }
```

5.5 README.md File Reference

Index