Functions in adjclust

1)**COLSARRAY**

Data Members

double* Data;

int* ColIndex;

int ArraySize;

PseudoMatrix *MyMatrix;

Functions

- ColsArray() sets default value of zero or NULL
- ColsArray(PseudoMatrix* M) uses Initialise to initialize the object
- void Initialize(PseudoMatrix* M) initializes MyMatrix as M and allocates ColIndex and Data space as array of sizes of M->h +1 elements
- void Set(int Col, double Value) finds col in ColIndex and inserts value in Data at the position found. If Col is not present inserts it at end in ColIndex and Value at end in Data
- void Set(int Col, double Value, int ForceIndex, bool Assumtion=False) if assumption is true and ColIndex[ForceIndex] = Col, assumption is set to false. If assumption is still true, call set(Col,Value). Else set Col in Colindex at ForceIndex and Value in Data at ForceIndex.
- void Restructure() deletes all those values from ColIndex and Data for which MyClassses in MyMatrix do not exist
- double Value(int Col) find Col in ColIndex and the return Data stores at that position
- double Value(int Col, int SupposedPlace) if Col is at SupposedPlace in ColIndex, return Data at SupposedPlace. Else if ColIndex at SupposedPlace is less than Col, searches it linearly after SupposedPlace and sets return data of the correct position. Else uses Value(Col) to find and return the data
- bool CheckMe() returns false if some MyClass[i] exist in MyMatrix but that i is not present in ColIndex

2)FUSIONNED CLASSES

Data Members

int NbFusions;

double FusionCost;

int MyIndex;

PseudoMatrix* MyMatrix;

int MyAvailableIndex;

int NextAvailableIndex;

int PrevAvailableIndex;

int WhoIAm;

Functions

- FusionnedClasses() sets indices and fusioncost to -1 and all others to 0 or NULL
- FusionnedClasses(PseudoMatrix* M, int ClassIndex) uses Initialise(M,ClassIndex) to initialize
- void Initialize(PseudoMatrix* M, int ClassIndex) sets MyMatrix to M, MyIndex and MyAvailableIndex to ClassIndex, PrevAvailableIndex to ClassIndex-1, NextAvailableIndex to ClassIndex+1 or if ClassIndex is greater than MyMatrix->p-1, then set it to -1. WhoIAm is set to –(ClassIndex+1) to depict that the cluster right now consist of individual elements.
- double MyValue() returns Value of MyMatrix at MyAvailableIndex, MyAvailableIndex
- void Swallow(int &NumFusionnedClasses) combines the fusionned class with the fusionnedClass in MyMatrix(i.e., MyClasses) at NextAvailableIndex by changing value of AvailableIndex of Next and NextAvailableIndex of current class. Also if this class after fusion has a NextAvailableIndex>-1, it changes the PrevAvailableIndex of this FusionnedClass. Also it updates WhoIAm to NumFusionnedClasses+1 which is positive depincting more than one element in the cluster. It also increments NumFusionnedClasses.
- bool Exist() return true if MyIndex is equal to MyAvailableIndex
- int MyCardinal() returns NbFusions+1
- void InitializeFusionCost() uses the following formula to initialize FusionCost 0.5*(M[MyIndex,MyIndex] + M[MyIndex+1, MyIndex+1] – 2*M[MyIndex, MyIndex+1]). Where M[i.i] denotes the value of MyMatrix at i,i
- void ComputeMyFusionCost() uses following formula to initialize fusioncost.

(a=MyCardinal(), b = MyCardinal() of NextAvailable Fusionned class, Coefficient = (a*b)/(a+b))

coefficient * (M[MyIndex,MyIndex]/(a*a) – 2*M[MyIndex,NextAvailableIndex]/(a*b) +

M[NextAvailableIndex, NextAvailableIndex]/(b*b))

3)CLASSESHEAP

Data members

int HeapSize;

int MaxSIze;

int CurrentNbClasses;

int NbWantedClasses;

int NumFusionnedClasses;

double* Output;

ostream* OutputStream;

int* Heap;

int* Which;

PseudoMatrix* MyMatrix;

Functions

- ClassesHeap() initializes member variables to 0 or NULL
- ClassesHeap(PseudoMatrix* M, int NbWC, string OutputPath=" ") uses initialize(M,NbWC,OutputPath) to initialize variables
- void Initialize(PseudoMatrix* M, int NbWC, string OutputPath) this function creates a min heap containing numbers from 1 to p-1 and taking fusioncost of MyClasses corresponding to that index to decide priority. After this while currentClasses is greater than NbWC or wanted classes, element from the top of the stack is fused with the NextAvailable class.
- ClassesHeap(double *V, int p, int h, int NbWC, string OutputPath="") calls initialize of PseudoMatrix to initialize elements
- ~ClassesHeap() frees the dynamic memory allocated to the various arrays
- void AddNode() adds the element corresponding to the current HeapSize in the minHeap and heapifies the heap based on its fusioncost
- bool CheckMe() it checks that MyClass exist for all elements in the Heap and also takes care that the Heap satisfies the MinHeap structure

- void MakeAFusion() Fuses the very first element in the heap with its NextAvailable Element. During the process, in the first two rows of Output, the elements which are merged are stored (merge) and the third row stores the fusion cost(gains)
- void Swap(int I, int j) swaps the elements in Heap at position I and j, and elements in Which at positions Heap[i] and Heap[j]
- bool RebalanceToDown(int index==0) starting from index it heapifies in the downward direction pushing the smaller child upward until it is inserted in its proper location.
- bool RebalanceToUp(int index=-1) starting from index it heapifies in the upward direction pushing the bigger parent downwards until it is at end so that it can be deleted easily
- bool FullRebalance(int index) starting from index it calls the required Rabalance Function to Heapfiy the heap properly

4)PSEUDOMATRIX

Data Members

int NbFusions;

int h;

int p;

ColsArray* MyData;

FusionnedClasses* MyClasses;

int NbClasses;

Functions

- PseudoMatrix() Initializes everything to 0 or NULL
- PseudoMatrix(double* V, int P, int H) uses Initialize(V,P,H) to initialize the members
- void Initialize(double* V, int P, int H) initializes MyClasses as an array of p and computes fusioncost for each. Also initializes MyData as an array of size p and setting the values from V for each row in MyData
- void Fusion(int LineIndex, int &NumFusionnedClasses) Fuses the MyClass at LineIndex while updating the values in the Matrix corresponding to that row and column
- double Value(int I, int j, int SupposedPlace) returns the element corresponding to row i and column j
- double Value(int I, int j) returns the element corresponding to row I and column j
- void set(int LineInA, int ColumnInA, double v) sets the element corresponding to row LineInA and column ColumInA to v by finding proper location of ColumnInA
- void set(int LineInA, int ColumnInA, double v, int ForceIndex, int Assumption = false) sets the element corresponding to row LineInA and column ColumnInA to v wher location is given by ForceIndex
- void DisplayMatrixA(ostream OutputPath) displays the data stored in matrix form
- ~PseudoMatrix() deletes the dynamically allocated space to all the arrays.

HeapHop function in PseudoMatrixRCpp.cpp

NumericVector HeapHop(NumericVector Input, int p, int h, int NbClasses) Intializes a vector V of size p*(h+1) by setting all the elements not available in the pencil shape extracted to zero. It then

contructs a ClassesHeap object which takes care of making clusters and returns merge and gains of each step.

Dependency Graph

library(igraph)

g = graph.formula(ClassesHeap-+PseudoMatrix, PseudoMatrix-+ColsArray, PseudoMatrix-+FusionnedClasses)

plot(g)

The output plot has been attached.