package org.tinos.deta.classification;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//task 20191219 daytime

//通过scale 距离来进行 坐标团切裂。

//Theory yaoguang.luo 20191219， 欧基里德

//Application yaoguang.luo

public class Fissile{

public static Map<Double, List<Position2D>> fissilePosition2D(List<Position2D> groups

, double scale) {

Map<Double, List<Position2D>> distanceGroups= new HashMap<>();

Map<Double, Position2D> distanceHeart= new HashMap<>();

Iterator<Position2D> iterator= groups.iterator();

double i= 0.0;

Here:

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

if(distanceGroups.isEmpty()) {

List<Position2D> list= new ArrayList<>();

list.add(position2D);

distanceGroups.put(i, list);

distanceHeart.put(i, position2D);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= distanceHeart.keySet().iterator();

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position2D currenctHeart= distanceHeart.get(doubleScale);

double distance= Distance.getDistance2D(currenctHeart, position2D);

if(distance< scale) {

//融入得到新的重心

Position2D newHeart

= Euclid.findCryptionPosition2D(currenctHeart, position2D);

//删除当前增加坐标集，更新坐标集

List<Position2D> list= distanceGroups.get(doubleScale);

list.add(position2D);

distanceGroups.put(doubleScale, list);

//删除当前重心数据，更新重心数据

distanceHeart.put(doubleScale, newHeart);

//找到

isFind= true;

//如果预测 坐标适应状态 可以把 continue 省略。

continue Here;

}

}

//新存

if(!isFind) {

List<Position2D> list= new ArrayList<>();

list.add(position2D);

distanceGroups.put(++i, list);

distanceHeart.put(i, position2D);

}

}

}

return distanceGroups;

}

public static Map<Double, List<Position3D>> fissilePosition3D(List<Position3D> groups

, double scale) {

Map<Double, List<Position3D>> distanceGroups= new HashMap<>();

Map<Double, Position3D> distanceHeart= new HashMap<>();

Iterator<Position3D> iterator= groups.iterator();

double i= 0.0;

Here:

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

if(distanceGroups.isEmpty()) {

List<Position3D> list= new ArrayList<>();

list.add(position3D);

distanceGroups.put(i, list);

distanceHeart.put(i, position3D);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= distanceHeart.keySet().iterator();

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position3D currenctHeart= distanceHeart.get(doubleScale);

double distance= Distance.getDistance3D(currenctHeart, position3D);

if(distance< scale) {

//融入得到新的重心

Position3D newHeart

= Euclid.findCryptionPosition3D(currenctHeart, position3D);

//删除当前增加坐标集，更新坐标集

List<Position3D> list= distanceGroups.get(doubleScale);

list.add(position3D);

distanceGroups.put(doubleScale, list);

//删除当前重心数据，更新重心数据

distanceHeart.put(doubleScale, newHeart);

//找到

isFind= true;

//如果预测 坐标适应状态 可以把 continue 省略。

continue Here;

}

}

//新存

if(!isFind) {

List<Position3D> list= new ArrayList<>();

list.add(position3D);

distanceGroups.put(++i, list);

distanceHeart.put(i, position3D);

}

}

}

return distanceGroups;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.classification;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//task 20191219 daytime

//通过scale 距离来进行 坐标团切裂, 并匹配最短最近重心域融入。

//算法 耗时更长，准确度再增加。

//Theory yaoguang.luo 20191219~23， 欧基里德

//Application yaoguang.luo

public class FissileWithMatch{

public static Map<Double, ArrayList<Position2D>>

fissilePosition2DWithMatch(ArrayList<Position2D> groups, double scale) {

Map<Double, ArrayList<Position2D>> distanceGroups= new HashMap<>();

Map<Double, Position2D> distanceHeart= new HashMap<>();

Iterator<Position2D> iterator= groups.iterator();

double i= 0.0;

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

if(distanceGroups.isEmpty()) {

ArrayList<Position2D> ArrayList= new ArrayList<>();

ArrayList.add(position2D);

distanceGroups.put(i, ArrayList);

distanceHeart.put(i, position2D);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= distanceHeart.keySet().iterator();

double shortest= 0;

double shortestDoubleScale= 0;

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position2D currenctHeart= distanceHeart.get(doubleScale);

double distance= Distance.getDistance2D(currenctHeart, position2D);

if(distance< scale) {

if(false== isFind) {

isFind= true;

shortest= distance;

shortestDoubleScale= doubleScale;

}else {

if(shortest< distance) {

shortest= distance;

shortestDoubleScale= doubleScale;

}

}

}

}

if(true== isFind) {

Position2D currenctHeart= distanceHeart.get(shortestDoubleScale);

//融入得到新的重心

Position2D newHeart= Euclid.findCryptionPosition2D(currenctHeart, position2D);

//删除当前增加坐标集，更新坐标集

ArrayList<Position2D> ArrayList= distanceGroups.get(shortestDoubleScale);

ArrayList.add(position2D);

distanceGroups.put(shortestDoubleScale, ArrayList);

//删除当前重心数据，更新重心数据

distanceHeart.put(shortestDoubleScale, newHeart);

}else {

ArrayList<Position2D> ArrayList= new ArrayList<>();

ArrayList.add(position2D);

distanceGroups.put(++i, ArrayList);

distanceHeart.put(i, position2D);

}

}

}

return distanceGroups;

}

public static Map<Double, ArrayList<Position3D>>

fissilePosition3DWithMatch(ArrayList<Position3D> groups, double scale) {

Map<Double, ArrayList<Position3D>> distanceGroups= new HashMap<>();

Map<Double, Position3D> distanceHeart= new HashMap<>();

Iterator<Position3D> iterator= groups.iterator();

double i= 0.0;

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

if(distanceGroups.isEmpty()) {

ArrayList<Position3D> ArrayList= new ArrayList<>();

ArrayList.add(position3D);

distanceGroups.put(i, ArrayList);

distanceHeart.put(i, position3D);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= distanceHeart.keySet().iterator();

double shortest= 0;

double shortestDoubleScale= 0;

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position3D currenctHeart= distanceHeart.get(doubleScale);

double distance= Distance.getDistance3D(currenctHeart, position3D);

if(distance< scale) {

if(false== isFind) {

isFind= true;

shortest= distance;

shortestDoubleScale= doubleScale;

}else {

if(shortest< distance) {

shortest= distance;

shortestDoubleScale= doubleScale;

}

}

}

}

if(true== isFind) {

Position3D currenctHeart= distanceHeart.get(shortestDoubleScale);

//融入得到新的重心

Position3D newHeart= Euclid.findCryptionPosition3D(currenctHeart, position3D);

//删除当前增加坐标集，更新坐标集

ArrayList<Position3D> ArrayList= distanceGroups.get(shortestDoubleScale);

ArrayList.add(position3D);

distanceGroups.put(shortestDoubleScale, ArrayList);

//删除当前重心数据，更新重心数据

distanceHeart.put(shortestDoubleScale, newHeart);

}else {

ArrayList<Position3D> ArrayList= new ArrayList<>();

ArrayList.add(position3D);

distanceGroups.put(++i, ArrayList);

distanceHeart.put(i, position3D);

}

}

}

return distanceGroups;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.classification;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.tinos.deta.basic.RatioMatrix;

//这个函数用于模糊概率集平均值采样分类

//思想：贝叶斯 模糊数学 统计于概率论， 任课教授 ：卡拉森。

//实现：罗瑶光

public class FuzzProbabailityClassification{

public static RatioMatrix getSimilarFuzzSet(RatioMatrix input, List<RatioMatrix> groups) {

double shortestSumRight=0;

boolean isFirst= true;

int key= 0;

int i= 0;

//成功集

Iterator<RatioMatrix> iterators= groups.iterator();

while(iterators.hasNext()) {

RatioMatrix ratio= iterators.next();

double rightRightMean= ratio.getRightRightRatio();

double rightErrorMean= ratio.getRightErrorRatio();

double errorRightMean= ratio.getErrorRightRatio();

double errorErrorMean= ratio.getErrorErrorRatio();

double predictionRightRight= Math.abs(input.getRightRightRatio()- rightRightMean);

double predictionRightError= Math.abs(input.getRightErrorRatio()- rightErrorMean);

double predictionErrorRight= Math.abs(input.getErrorRightRatio()- errorRightMean);

double predictionErrorError= Math.abs(input.getErrorErrorRatio()- errorErrorMean);

double tempSumRight= predictionRightRight+ predictionRightError

+ predictionErrorRight+ predictionErrorError;

if(true== isFirst) {

isFirst= false;

shortestSumRight= tempSumRight;

key=i;

}else {

if(shortestSumRight> tempSumRight) {

shortestSumRight= tempSumRight;

key= i;

}

}

i++;

}

return groups.get(key);

}

public static List<RatioMatrix> getSimilarFuzzSetWithScale(RatioMatrix input, List<RatioMatrix> groups, double scale) {

List<RatioMatrix> output= new ArrayList<>();

Iterator<RatioMatrix> iterators= groups.iterator();

while(iterators.hasNext()) {

RatioMatrix ratio= iterators.next();

double rightRightMean= ratio.getRightRightRatio();

double rightErrorMean= ratio.getRightErrorRatio();

double errorRightMean= ratio.getErrorRightRatio();

double errorErrorMean= ratio.getErrorErrorRatio();

double predictionRightRight= Math.abs(input.getRightRightRatio()- rightRightMean);

double predictionRightError= Math.abs(input.getRightErrorRatio()- rightErrorMean);

double predictionErrorRight= Math.abs(input.getErrorRightRatio()- errorRightMean);

double predictionErrorError= Math.abs(input.getErrorErrorRatio()- errorErrorMean);

double tempSumRight= predictionRightRight+ predictionRightError+ predictionErrorRight+ predictionErrorError;

if(tempSumRight< scale) {

output.add(ratio);

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.classification;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

//这个函数用于通过重心位移距离来进行坐标分类

//思想：欧基里德

//实现：罗瑶光

public class PositionClasification{

public static Map<Double, List<Position2D>>

addNewPositionWithoutHeart(Map<Double, List<Position2D>> groups, Position2D position2D

, double scaleDistance){

double groupKey= 0;

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

groupKey= iterator.next();

List<Position2D> group= groups.get(groupKey);

Position2D heart= Euclid.findHeartPosition2D(group);

double distance= Distance.getDistance2D(heart, position2D);

if(scaleDistance< distance) {

group.add(position2D);

groups.put(groupKey, group);

return groups;

}

}

List<Position2D> group= new ArrayList<Position2D>() ;

group.add(position2D);

groups.put(groupKey+1, group);

return groups;

}

public static Map<Double, List<Position2D>> addNewPositionWithHeart(Map<Double

, List<Position2D>> groups, Position2D position2D

, Map<Double, Position2D> hearts, double scaleDistance){

double groupKey= 0;

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

groupKey= iterator.next();

List<Position2D> group= groups.get(groupKey);

Position2D heart= hearts.get(groupKey);

double distance= Distance.getDistance2D(heart, position2D);

if(scaleDistance< distance) {

group.add(position2D);

groups.put(groupKey, group);

//hearts熵增

Position2D CryptHeart= Euclid.findCryptionPosition2D(heart, position2D);

hearts.put(groupKey, CryptHeart);

return groups;

}

}

List<Position2D> group= new ArrayList<Position2D>() ;

group.add(position2D);

groups.put(groupKey+1, group);

//heart

hearts.put(groupKey+1, position2D);

return groups;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.classification;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//这个函数用于坐标集团距离采样匹配输出

//思想: 欧基里德

//实现：罗瑶光

public class PositionHeartsSample{

public static List<Position2D> getShortestSamplePosition2DGroup(Position2D inputHeart

, Map<Double, List<Position2D>> groups) {

double shortestDistance= 0;

boolean isFirst= true;

double key= 0;

Iterator<Double> iterators= groups.keySet().iterator();

while(iterators.hasNext()) {

double mapKey= iterators.next();

List<Position2D> positions= groups.get(mapKey);

Position2D heart= Euclid.findHeartPosition2D(positions);

double distance= Distance.getDistance2D(inputHeart, heart);

if(true== isFirst) {

isFirst= false;

shortestDistance= distance;

key= mapKey;

}else {

if(shortestDistance> distance) {

shortestDistance= distance;

key= mapKey;

}

}

}

return groups.get(key);

}

public static Map<Double, List<Position2D>> getShorterSamplePosition2DGroupsWithScale(Position2D

inputHeart, Map<Double, List<Position2D>> groups, double scale) {

Map<Double, List<Position2D>> output= new HashMap<>();

Iterator<Double> iterators= groups.keySet().iterator();

while(iterators.hasNext()) {

double mapKey= iterators.next();

List<Position2D> positions= groups.get(mapKey);

Position2D heart= Euclid.findHeartPosition2D(positions);

double distance= Distance.getDistance2D(inputHeart, heart);

if(scale> distance) {

output.put(mapKey, positions);

}

}

return output;

}

public static List<Position3D> getShortestSamplePosition3DGroup(Position3D inputHeart

, Map<Double, List<Position3D>> groups) {

double shortestDistance= 0;

boolean isFirst= true;

double key= 0;

Iterator<Double> iterators= groups.keySet().iterator();

while(iterators.hasNext()) {

double mapKey= iterators.next();

List<Position3D> positions= groups.get(mapKey);

Position3D heart= Euclid.findHeartPosition3D(positions);

double distance= Distance.getDistance3D(inputHeart, heart);

if(true== isFirst) {

isFirst= false;

shortestDistance= distance;

key= mapKey;

}else {

if(shortestDistance> distance) {

shortestDistance= distance;

key= mapKey;

}

}

}

return groups.get(key);

}

public static Map<Double, List<Position3D>> getShortestSamplePosition3DGroupsWithScale(Position3D

inputHeart, Map<Double, List<Position3D>> groups, double scale) {

Map<Double, List<Position3D>> output= new HashMap<>();

Iterator<Double> iterators= groups.keySet().iterator();

while(iterators.hasNext()) {

double mapKey= iterators.next();

List<Position3D> positions= groups.get(mapKey);

Position3D heart= Euclid.findHeartPosition3D(positions);

double distance= Distance.getDistance3D(inputHeart, heart);

if(scale> distance) {

output.put(mapKey, positions);

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.classification;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.RatioMatrix;

//这个函数用于通过概率轭相似度来进行坐标分类

//思想：贝叶斯

//实现：罗瑶光

public class ProbabilityClasification{

public static boolean predictionResult(RatioMatrix input, List<RatioMatrix> groups, double scale) {

double rightRightMean= 0;

double rightErrorMean= 0;

double errorRightMean= 0;

double errorErrorMean= 0;

//成功集

Iterator<RatioMatrix> iterators= groups.iterator();

while(iterators.hasNext()) {

RatioMatrix ratio= iterators.next();

rightRightMean+= ratio.getRightRightRatio();

rightErrorMean+= ratio.getRightErrorRatio();

errorRightMean+= ratio.getErrorRightRatio();

errorErrorMean+= ratio.getErrorErrorRatio();

}

rightRightMean= rightRightMean/ groups.size();

rightErrorMean= rightErrorMean/ groups.size();

errorRightMean= errorRightMean/ groups.size();

errorErrorMean= errorErrorMean/ groups.size();

//决策轭

double predictionRightRight= input.getRightRightRatio()- rightRightMean;

double predictionRightError= input.getRightErrorRatio()- rightErrorMean;

double predictionErrorRight= input.getErrorRightRatio()- errorRightMean;

double predictionErrorError= input.getErrorErrorRatio()- errorErrorMean;

//迪摩根轭集 因为考虑到在质量检测项目中的适用性，进行了乘积修改如下。

if(!(predictionRightRight< 0|| predictionRightError> 0|| predictionErrorRight< 0||predictionErrorError> 0)) {

return true;

}

return true;

}

public static String predictionMatrixResult(RatioMatrix input, Map<String, RatioMatrix> groups

, double scale) {

String groupKey= null;

double shortestDistance= 0;

boolean isFirst= true;

//轭

double esyn= input.getRightRightRatio()+ input.getErrorErrorRatio();

//double esny= input.getErrorRightRatio()+ input.getRightErrorRatio();若使用该行 轭 误差集合请自行校正。罗瑶光20191217

double yesyn= input.getRightRightRatio()/ esyn;

double nesyn= input.getErrorErrorRatio()/ esyn;

double totalRatio= -yesyn\* Math.log(yesyn)- nesyn\* Math.log(nesyn);

//本征

double esyy= input.getRightRightRatio()+ input.getRightErrorRatio();

double yesyy= input.getRightRightRatio()/ esyy;

double nesyy= input.getRightErrorRatio()/ esyy;

double meany= -yesyy\* Math.log(yesyy)- nesyy\* Math.log(nesyy);

double esnn= input.getErrorRightRatio()+ input.getErrorErrorRatio();

double yesnn= input.getErrorRightRatio()/ esnn;

double nesnn= input.getErrorErrorRatio()/ esnn;

double meann=-yesnn\* Math.log(yesnn)- nesnn\* Math.log(nesnn);

double deta= totalRatio- meany- meann;

Iterator<String> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

String matrixKey=iterator.next();

RatioMatrix ratioMatrix= groups.get(matrixKey);

//采样

double tesyy= ratioMatrix.getRightRightRatio()+ ratioMatrix.getRightErrorRatio();

double tyesyy= ratioMatrix.getRightRightRatio()/ tesyy;

double tnesyy= ratioMatrix.getRightErrorRatio()/ tesyy;

double tmeany= -tyesyy\* Math.log(tyesyy)- tnesyy\* Math.log(tnesyy);

double tesnn= ratioMatrix.getErrorRightRatio()+ ratioMatrix.getErrorErrorRatio();

double tyesnn= ratioMatrix.getErrorRightRatio()/ tesnn;

double tnesnn= ratioMatrix.getErrorErrorRatio()/ tesnn;

double tmeann= -tyesnn\* Math.log(tyesnn)- tnesnn\* Math.log(tnesnn);

double tdeta= totalRatio- tmeany- tmeann;

//取值

if(isFirst) {

isFirst= !isFirst;

shortestDistance= Math.abs(deta- tdeta);

groupKey= matrixKey;

}else {

if(Math.abs(deta- tdeta)< shortestDistance) {

shortestDistance= Math.abs(deta- tdeta);

groupKey= matrixKey;

}

}

}

//输出

return groupKey;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.cluster;

import java.util.List;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class ClusterAttraction{

//临近2个坐标团的相互引力比算法建模观测，小于1 吸引，大于1 排斥。

//思想： 罗瑶光，欧基里德， 立体几何，20191227

//实现： 罗瑶光

public static double getTwoPosition2DClusterAttraction(List<Position2D> clusterOne

, List<Position2D> clusterTwo) {

Position2D midOne= Euclid.findMidPosition2D(clusterOne);

Position2D midTwo= Euclid.findMidPosition2D(clusterTwo);

Position2D heartOne= Euclid.findHeartPosition2D(clusterOne);

Position2D heartTwo= Euclid.findHeartPosition2D(clusterTwo);

double midDistance= Distance.getDistance2D(midOne, midTwo);

double heartDistance= Distance.getDistance2D(heartOne, heartTwo);

return heartDistance/ midDistance;

}

public static double getTwoPosition3DClusterAttraction(List<Position3D> clusterOne

, List<Position3D> clusterTwo) {

Position3D midOne= Euclid.findMidPosition3D(clusterOne);

Position3D midTwo= Euclid.findMidPosition3D(clusterTwo);

Position3D heartOne= Euclid.findHeartPosition3D(clusterOne);

Position3D heartTwo= Euclid.findHeartPosition3D(clusterTwo);

double midDistance= Distance.getDistance3D(midOne, midTwo);

double heartDistance= Distance.getDistance3D(heartOne, heartTwo);

return heartDistance/ midDistance;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.cluster;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//task 20191220-20191222 daytime

//通过scale 距离来进行坐标团集合 融聚。

//Theory yaoguang.luo 20191219， 欧基里德

//Application yaoguang.luo

//注意：做完计算可以删除冗余map数据来加速运算，但是考虑到 java对象入参是指针形式，于是取消删除思想，避免破坏函数上层逻辑。

//小伙伴有加速需要，可以自行修改。

public class Fusion{

public static Map<Double, List<Position2D>> fusionPosition2DwithHeart

(Map<Double, List<Position2D>> groups, Map<Double, Position2D> groupsHeart, double scale){

//初始

Map<Double, List<Position2D>> output= new HashMap<>();

Map<Double, Position2D> outputHeart= new HashMap<>();

//逐团比较重心距离

Iterator<Double> outLoop= groupsHeart.keySet().iterator();

Map<Double, Double> isDelete= new HashMap<>();

//小于精度内融聚

//HereOut:

while(outLoop.hasNext()) {

double out= outLoop.next();

Iterator<Double> inLoop= groupsHeart.keySet().iterator();

HereIn:

while(inLoop.hasNext()) {

double in= inLoop.next();

if(out== in|| output.containsKey(in)|| isDelete.containsKey(in)) {

continue HereIn;//out做融聚参照物，in做计算算子。output做观测物。

}

Position2D inHeart= groupsHeart.get(in);

//Position2D outHeart= groupsHeart.get(out);

//如下因为java的指针被对象化，直接修改入参会产生问题于是新做了

//outputHeart变量来处理。

Position2D outHeart= outputHeart.containsKey(out)

? outputHeart.get(out): groupsHeart.get(out);

double distance= Distance.getDistance2D(inHeart, outHeart);

//比较 是融合

if(distance< scale) {

List<Position2D> outList;

//比较有融媒

if(output.containsKey(out)) {

outList= output.get(out);

}else {//比较无融媒

//加融媒in to out 加out，删除 in

outList= groups.get(out);

}

//加融媒in to out 删除 in

List<Position2D> inList= groups.get(in);

Iterator<Position2D> iterator= inList.iterator();

while(iterator.hasNext()) {

outList.add(iterator.next());

}

output.put(out, outList);

//更新heart

Position2D newHeart

= Euclid.findCryptionPosition2D(outHeart, inHeart);

outputHeart.put(out, newHeart);

isDelete.put(in, in);

}else {//比较 否融合）

//比较有融媒

if(!output.containsKey(out)) {//比较无融媒

//加融媒 out，删除 out，加融媒 in 删除 in

if(!output.containsKey(out)) {

List<Position2D> outList= groups.get(out);

output.put(out, outList);

//更新heart

outputHeart.put(out, outHeart);

isDelete.put(out, out);

}

}

if(!output.containsKey(in)) {

List<Position2D> inList= groups.get(in);

output.put(in, inList);

//更新heart

outputHeart.put(in, inHeart);

isDelete.put(in, in);

}

}

}

}

return output;

}

public static Map<Double, List<Position3D>> fusionPosition3DwithHeart

(Map<Double, List<Position3D>> groups, Map<Double, Position3D> groupsHeart, double scale){

//初始

Map<Double, List<Position3D>> output= new HashMap<>();

Map<Double, Position3D> outputHeart= new HashMap<>();

//逐团比较重心距离

Iterator<Double> outLoop= groupsHeart.keySet().iterator();

Map<Double, Double> isDelete= new HashMap<>();

//小于精度内融聚

//HereOut:

while(outLoop.hasNext()) {

double out= outLoop.next();

Iterator<Double> inLoop= groupsHeart.keySet().iterator();

HereIn:

while(inLoop.hasNext()) {

double in= inLoop.next();

if(out== in|| output.containsKey(in)|| isDelete.containsKey(in)) {

continue HereIn;//out做融聚参照物，in做计算算子。output做观测物。

}

Position3D inHeart= groupsHeart.get(in);

//Position3D outHeart= groupsHeart.get(out);

//如下因为java的指针被对象化，直接修改入参会产生问题于是新做了

//outputHeart变量来处理。

Position3D outHeart= outputHeart.containsKey(out)

? outputHeart.get(out): groupsHeart.get(out);

double distance= Distance.getDistance3D(inHeart, outHeart);

//比较 是融合

if(distance< scale) {

List<Position3D> outList;

//比较有融媒

if(output.containsKey(out)) {

outList= output.get(out);

}else {//比较无融媒

//加融媒in to out 加out，删除 in

outList= groups.get(out);

}

//加融媒in to out 删除 in

List<Position3D> inList= groups.get(in);

Iterator<Position3D> iterator= inList.iterator();

while(iterator.hasNext()) {

outList.add(iterator.next());

}

output.put(out, outList);

//更新heart

Position3D newHeart

= Euclid.findCryptionPosition3D(outHeart, inHeart);

outputHeart.put(out, newHeart);

isDelete.put(in, in);

}else {//比较 否融合）

//比较有融媒

if(!output.containsKey(out)) {//比较无融媒

//加融媒 out，删除 out，加融媒 in 删除 in

if(!output.containsKey(out)) {

List<Position3D> outList= groups.get(out);

output.put(out, outList);

//更新heart

outputHeart.put(out, outHeart);

isDelete.put(out, out);

}

}

if(!output.containsKey(in)) {

List<Position3D> inList= groups.get(in);

output.put(in, inList);

//更新heart

outputHeart.put(in, inHeart);

isDelete.put(in, in);

}

}

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.cluster;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//求融聚团宇宙的重心

//Theory yaoguang.luo 20191219， 欧基里德

//Application yaoguang.luo

public class FusionHeart{

public static Position2D fusionPosition2DHeart(Map<Double, List<Position2D>> groups){

List<Position2D> hearts= new ArrayList<>();

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

double value= iterator.next();

Position2D positionHeart= Euclid.findHeartPosition2D(groups.get(value));

hearts.add(positionHeart);

}

return Euclid.findHeartPosition2D(hearts);

}

public static Position3D fusionPosition3DHeart(Map<Double, List<Position3D>> groups){

List<Position3D> hearts= new ArrayList<>();

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

double value= iterator.next();

Position3D positionHeart= Euclid.findHeartPosition3D(groups.get(value));

hearts.add(positionHeart);

}

return Euclid.findHeartPosition3D(hearts);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.cluster;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

import org.tinos.deta.ratio.DistanceRatio;

//Theory: Yaoguang.luo 20191216：12：06

//一种仅仅通过坐标重心来描绘射极边缘面探测使用方法。

//Application: Yaoguang.luo

public class SideEnd{

public static List<Position2D> getSideEnd2D(List<Position2D> list, double scale) {

Position2D heart= Euclid.findHeartPosition2D(list);

Map<Double, Position2D> ratioSide= new HashMap<>();

Iterator<Position2D> iterator= list.iterator();

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

double ratio= DistanceRatio.getDistanceRatio2D(heart, position2D);

if(ratioSide.containsKey(ratio)) {

double newDistance= Distance.getDistance2D(heart, position2D);

double oldDistance= Distance.getDistance2D(heart, ratioSide.get(ratio));

if(newDistance> oldDistance) {

if(newDistance> scale) {

ratioSide.put(ratio, position2D);

}

}

}else {

double newDistance= Distance.getDistance2D(heart, position2D);

if(newDistance> scale) {

ratioSide.put(ratio, position2D);

}

}

}

//转换

List<Position2D> output= new ArrayList<>();

Iterator<Double> iteratorKeys= ratioSide.keySet().iterator();

while(iteratorKeys.hasNext()) {

output.add(ratioSide.get(iteratorKeys.next()));

}

return output;

}

public static List<Position3D> getSideEnd3D(List<Position3D> list, double scale) {

Position3D heart= Euclid.findHeartPosition3D(list);

Map<Double, Position3D> ratioSide= new HashMap<>();

Iterator<Position3D> iterator= list.iterator();

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

double ratio= DistanceRatio.getDistanceRatio3D(heart, position3D);

if(ratioSide.containsKey(ratio)) {

double newDistance= Distance.getDistance3D(heart, position3D);

double oldDistance= Distance.getDistance3D(heart, ratioSide.get(ratio));

if(newDistance> oldDistance) {

if(newDistance> scale) {

ratioSide.put(ratio, position3D);

}

}

}else {

double newDistance= Distance.getDistance3D(heart, position3D);

if(newDistance> scale) {

ratioSide.put(ratio, position3D);

}

}

}

//转换

List<Position3D> output= new ArrayList<>();

Iterator<Double> iteratorKeys= ratioSide.keySet().iterator();

while(iteratorKeys.hasNext()) {

output.add(ratioSide.get(iteratorKeys.next()));

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.demension;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.tinos.deta.basic.Euclid;

public class FindHeartPositions{

//求坐标团的重心。

public static Map<Double, Position2D> getPosition2DGroupsHearts(Map<Double

, ArrayList<Position2D>> groups){

Map<Double, Position2D> output= new HashMap<>();

Iterator<Double> iterator= output.keySet().iterator();

while(iterator.hasNext()){

double value= iterator.next();

output.put(value, Euclid.findHeartPosition2D(groups.get(value)));

}

return output;

}

public static Map<Double, Position3D> getPosition3DGroupsHearts(Map<Double

, ArrayList<Position3D>> groups){

Map<Double, Position3D> output= new HashMap<>();

Iterator<Double> iterator= output.keySet().iterator();

while(iterator.hasNext()){

double value= iterator.next();

output.put(value, Euclid.findHeartPosition3D(groups.get(value)));

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.demension;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.tinos.deta.basic.Euclid;

public class FindMidPositions{

//求坐标团的中心。

public static Map<Double, Position2D> getPosition2DGroupsMids(Map<Double

, ArrayList<Position2D>> groups){

Map<Double, Position2D> output= new HashMap<>();

Iterator<Double> iterator= output.keySet().iterator();

while(iterator.hasNext()){

double value= iterator.next();

output.put(value, Euclid.findMidPosition2D(groups.get(value)));

}

return output;

}

public static Map<Double, Position3D> getPosition3DGroupsMids(Map<Double

, ArrayList<Position3D>> groups){

Map<Double, Position3D> output= new HashMap<>();

Iterator<Double> iterator= output.keySet().iterator();

while(iterator.hasNext()){

double value= iterator.next();

output.put(value, Euclid.findMidPosition3D(groups.get(value)));

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.desolation;

**import** org.tinos.deta.statistic.LYG4DWithDoubleQuickSort4D;

**public** **class** ErrorAsserts{

//这个函数用于进行精度误差匹配 获取质量是否属于合格状态。

//inputValue 代表观测成份。

//matchValues 代表观测参照样本。

//scale代表误差精度。

//sortStackRange 根据样本的相同项多少来确定堆栈溢出的适合减少递归树数。

**public** **boolean** getErrorAsserts(**double** inputValue, **double**[] matchValues, **double** scale) {

**double** max= inputValue+ scale;

**double** min= inputValue- scale;

**for**(**int** i= 0; i< matchValues.length; i++) {

**if**(!(matchValues[i]> max|| matchValues[i]< min)) {

**return** **true**;

}

}

**return** **false**;

}

**public** **boolean** getBinaryErrorAsserts(**double** inputValue, **double**[] matchValues

, **double** scale, **int** sortStackRange, **boolean** isSort) {

**if**(!isSort) {

matchValues= **new** LYG4DWithDoubleQuickSort4D().sort(matchValues, sortStackRange);

}

**double** max= inputValue+ scale;

**double** min= inputValue- scale;

**int** big= matchValues.length;

**int** mid= matchValues.length/ 2;

**while**(big> 0) {

**if**(!(matchValues[mid]> max|| matchValues[mid]< min)) {

**return** **true**;

}

**if**(inputValue> matchValues[mid]) {

mid= (mid+ big)/ 2;

}**else** {

big= mid;

mid= mid/ 2;

}

}

**return** **false**;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.hint;

import java.util.List;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Line2D;

import org.tinos.deta.demension.Line3D;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class PositionsHintDirection{

//坐标集隐藏运动趋势预测算法。

//这个函数适用于在坐标团中观测 中心 与重心的 长度 来预测坐标团的稳定状态和轨迹预判。

//适用于 游戏，数据建模，化学，物理等领域。

//思想：流体内部分子力 统计

//实现：罗瑶光

public static Line2D getHintDirectionTrendFromPosition2Ds(List<Position2D> inputs) {

Position2D mid= Euclid.findMidPosition2D(inputs);

Position2D heart= Euclid.findHeartPosition2D(inputs);

Line2D line2D= new Line2D();

line2D.setBegin(mid);

line2D.setEnd(heart);

return line2D;

}

public static Line3D getHintDirectionTrendFromPosition3Ds(List<Position3D> inputs) {

Position3D mid= Euclid.findMidPosition3D(inputs);

Position3D heart= Euclid.findHeartPosition3D(inputs);

Line3D line3D= new Line3D();

line3D.setBegin(mid);

line3D.setEnd(heart);

return line3D;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.ICA;

**public** **class** CorrelationICA{

//比较简单的鸡尾酒频率盲分割

**public** **static** **double**[] frequencyUpSplit(**double**[] originFrequency, **double**[] compareFrequency) {

**double**[] output= **new** **double**[originFrequency.length];

**for**(**int** i= 0; i< originFrequency.length; i++) {

output[i]= originFrequency[i]- compareFrequency[i]> 0? originFrequency[i]: 0;

}

**return** output;

}

**public** **static** **double**[] frequencyDownSplit(**double**[] originFrequency, **double**[] compareFrequency) {

**double**[] output= **new** **double**[originFrequency.length];

**for**(**int** i= 0; i< originFrequency.length; i++) {

output[i]= originFrequency[i]- compareFrequency[i]< 0? compareFrequency[i]: 0;

}

**return** output;

}

**public** **static** **double**[] frequencyUpSplitWithScale(**double**[] originFrequency

, **double**[] compareFrequency, **double** scale) {

**double**[] output= **new** **double**[originFrequency.length];

**for**(**int** i= 0; i< originFrequency.length; i++) {

output[i]= originFrequency[i]- compareFrequency[i]> scale? originFrequency[i]: 0;

}

**return** output;

}

**public** **static** **double**[] frequencyDownSplitWithScale(**double**[] originFrequency

, **double**[] compareFrequency, **double** scale) {

**double**[] output= **new** **double**[originFrequency.length];

**for**(**int** i= 0; i< originFrequency.length; i++) {

output[i]= originFrequency[i]- compareFrequency[i]< scale? compareFrequency[i]: 0;

}

**return** output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.image;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class ImagePixClassification{

//思想：统计与概率论

//作者：罗瑶光

//将shareholder分层处理的图片像素数据进行 分类归类统计输出

public static Map<Integer, ArrayList<Position2D>> getImagePixClassificationMap(int[][] pixMap){

Map<Integer, ArrayList<Position2D>> output= new HashMap<>();

for(int i= 0; i< pixMap.length; i++) {

for(int j= 0; j< pixMap[0].length; j++) {

if(pixMap[i][j]> 0) {

ArrayList<Position2D> temp;

if(output.containsKey(pixMap[i][j])) {

temp= output.get(pixMap[i][j]);

}else {

temp= new ArrayList<>();

}

Position2D position2D= new Position2D(i, j);

temp.add(position2D);

output.put(pixMap[i][j], temp);

}

}

}

return output;

}

public static Map<Integer, ArrayList<Position3D>> getImagePixClassificationMap(int[][][] pixMap){

Map<Integer, ArrayList<Position3D>> output= new HashMap<>();

for(int i= 0; i< pixMap.length; i++) {

for(int j= 0; j< pixMap[0].length; j++) {

for(int k= 0; k< pixMap[0][0].length; k++) {

if(pixMap[i][j][k]> 0) {

ArrayList<Position3D> temp;

if(output.containsKey(pixMap[i][j][k])) {

temp= output.get(pixMap[i][j][k]);

}else {

temp= new ArrayList<>();

}

Position3D position3D= new Position3D(i, j, k);

temp.add(position3D);

output.put(pixMap[i][j][k], temp);

}

}}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.image;

**public** **class** ImagePixExtract{

//思想：平面几何， 模式识别

//作者：罗瑶光

//适用于波形图变换数列.

//将像素矩阵指定的RBG颜色值拿出来形成波形数列输出

**public** **static** **int**[] getVWaveFromImagePix(**int**[][] pixMap, **int** RGB){

**int**[] output= **new** **int**[pixMap.length];

Next:

**for**(**int** i= 0; i< pixMap.length; i++) {

**for**(**int** j= 0; j< pixMap[0].length; j++) {

**if**(RGB== pixMap[i][j]) {

output[i]= j;

**continue** Next;

}

}

}

**return** output;

}

**public** **static** **int**[] getHWaveFromImagePix(**int**[][] pixMap, **int** RGB){

**int**[] output= **new** **int**[pixMap[0].length];

Next:

**for**(**int** i= 0; i< pixMap[0].length; i++) {

**for**(**int** j= 0; j< pixMap.length; j++) {

**if**(RGB== pixMap[j][i]) {

output[i]= j;

**continue** Next;

}

}

}

**return** output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.image;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.Map;

import org.tinos.deta.classification.FissileWithMatch;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class ImagePixGroupFilter{

//思想：罗瑶光

//作者：罗瑶光

//将shareholder分层处理的图片像素数据 按指定的精度和对应像素进行按像素团的大小进行特定过滤。

public int[][] getImagePix2DGroupFilter(int[][] inputPixMatrix

, int RBG, int distanceScale,int max, int min){

//像素分类

Map<Integer, ArrayList<Position2D>> map= ImagePixClassification

.getImagePixClassificationMap(inputPixMatrix);

//获取分类后的团簇

Map<Double, ArrayList<Position2D>> mid= FissileWithMatch

.fissilePosition2DWithMatch(map.get(RBG), distanceScale);

//筛选团簇，过滤团簇

Iterator<Double> iteratorMid= mid.keySet().iterator();

while(iteratorMid.hasNext()) {

ArrayList<Position2D> list= mid.get(iteratorMid.next());

if(null!= list) {

if(list.size()< min|| list.size()> max) {

Iterator< Position2D> iterator= list.iterator();

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

inputPixMatrix[(int) position2D.getX()]

[(int) position2D.getY()]= 0;

}

}

}

}

return inputPixMatrix;

}

public int[][][] getImagePix3DGroupFilter(int[][][] inputPixMatrix

, int RBG, int distanceScale,int max, int min){

//像素分类

Map<Integer, ArrayList<Position3D>> map= ImagePixClassification

.getImagePixClassificationMap(inputPixMatrix);

//获取分类后的团簇

Map<Double, ArrayList<Position3D>> mid= FissileWithMatch

.fissilePosition3DWithMatch(map.get(RBG), distanceScale);

//筛选团簇，过滤团簇

Iterator<Double> iteratorMid= mid.keySet().iterator();

while(iteratorMid.hasNext()) {

ArrayList<Position3D> list= mid.get(iteratorMid.next());

if(null!= list) {

if(list.size()< min|| list.size()> max) {

Iterator< Position3D> iterator= list.iterator();

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

inputPixMatrix[(int) position3D.getX()][(int) position3D.getY()]

[(int) position3D.getZ()]= 0;

}

}

}

}

return inputPixMatrix;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.isolation;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class ForestIsolation{

//带精度 2维 商旅路径团簇 森林单元 隔离 算法

//Theory 《神经网络: 权距》，欧基里德， Yaoguang.Luo 20191220

//Application Yaoguang.Luo

//适用于 最短路径，最小距离，商旅分析预测，等项目中

public static Map<Double, List<Position2D>> getTSPForestIsolationGroups2D(List<Position2D> groups, double scale) {

Map<Double, List<Position2D>> output= new HashMap<>();

Iterator<Position2D> iterator= groups.iterator();

double i= 0;

while(iterator.hasNext()) {

i++;

Position2D position2D= iterator.next();

Iterator<Position2D> inIterator= groups.iterator();

double j= 0;

Here:

while(inIterator.hasNext()) {

j++;

Position2D inPosition2D= inIterator.next();

//计算

if(i== j) {

continue Here;

}

double distance= Distance.getDistance2D(position2D, inPosition2D);

if(distance> scale) {

continue Here;

}

//添加

List<Position2D> list;

if(output.containsKey(i)) {

list= output.get(i);

}else {

list= new ArrayList<>();

list.add(position2D);

}

list.add(inPosition2D);

output.put(i, list);

}

}

return output;

}

//带精度 3维 商旅路径团簇 森林单元 隔离 算法

//Theory 《神经网络: 权距》，欧基里德， Yaoguang.Luo 20191220

//Application Yaoguang.Luo

//适用于 最短路径，最小距离，商旅分析预测，等项目中

public static Map<Double, List<Position3D>> getTSPIsolationGroups3D(List<Position3D> groups

, double scale) {

Map<Double, List<Position3D>> output= new HashMap<>();

Iterator<Position3D> iterator= groups.iterator();

double i= 0;

while(iterator.hasNext()) {

i++;

Position3D position3D= iterator.next();

Iterator<Position3D> inIterator= groups.iterator();

double j= 0;

Here:

while(inIterator.hasNext()) {

j++;

Position3D inPosition3D= inIterator.next();

//计算

if(i== j) {

continue Here;

}

double distance= Distance.getDistance3D(position3D, inPosition3D);

if(distance> scale) {

continue Here;

}

//添加

List<Position3D> list;

if(output.containsKey(i)) {

list= output.get(i);

}else {

list= new ArrayList<>();

list.add(position3D);

}

list.add(inPosition3D);

output.put(i, list);

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.isolation;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class Isolation{

public static double[] getCorrelation(double[] firstArray, double[] secondArray) {

double[] output= new double [firstArray.length];

for(int i= 0; i< firstArray.length; i++) {

output[i]= firstArray[i]/ secondArray[i];

}

return output;

}

//带精度 2维(非欧拉权距)商旅路径团簇 隔离 算法

//Theory 《神经网络: 权距》，欧基里德， Yaoguang.Luo 20191220

//Application Yaoguang.Luo

//适用于 最短路径，最小距离，商旅分析预测，等项目中

public static Map<Double, List<Position2D>> getTSPIsolationGroups2D(List<Position2D> groups

, double scale) {

boolean[][] isDelete= new boolean[groups.size()][groups.size()];

Map<Double, List<Position2D>> output= new HashMap<>();

Iterator<Position2D> iterator= groups.iterator();

double i= 0;

while(iterator.hasNext()) {

i++;

Position2D position2D= iterator.next();

Iterator<Position2D> inIterator= groups.iterator();

double j= 0;

Here:

while(inIterator.hasNext()) {

j++;

Position2D inPosition2D= inIterator.next();

//计算

if(isDelete[(int)i- 1][(int)j- 1]|| i== j) {

continue Here;

}

//轭消

isDelete[(int)i- 1][(int)j- 1]= true;

isDelete[(int)j- 1][(int)i- 1]= true;

double distance= Distance.getDistance2D(position2D, inPosition2D);

if(distance> scale) {

continue Here;

}

//添加

List<Position2D> list;

if(output.containsKey(i)) {

list= output.get(i);

}else {

list= new ArrayList<>();

list.add(position2D);

}

list.add(inPosition2D);

output.put(i, list);

}

}

return output;

}

//带精度 3维(非欧拉权距)商旅路径团簇 隔离 算法

//Theory 《神经网络: 权距》，欧基里德， Yaoguang.Luo 20191220

//Application Yaoguang.Luo

//适用于 最短路径，最小距离，商旅分析预测，等项目中

public static Map<Double, List<Position3D>> getTSPIsolationGroups3D(List<Position3D> groups

, double scale) {

boolean[][] isDelete= new boolean[groups.size()][groups.size()];

Map<Double, List<Position3D>> output= new HashMap<>();

Iterator<Position3D> iterator= groups.iterator();

double i= 0;

while(iterator.hasNext()) {

i++;

Position3D position3D= iterator.next();

Iterator<Position3D> inIterator= groups.iterator();

double j= 0;

Here:

while(inIterator.hasNext()) {

j++;

Position3D inPosition3D= inIterator.next();

//计算

if(isDelete[(int)i- 1][(int)j- 1]|| i== j) {

continue Here;

}

//轭消

isDelete[(int)i- 1][(int)j- 1]= true;

isDelete[(int)j- 1][(int)i- 1]= true;

double distance= Distance.getDistance3D(position3D, inPosition3D);

if(distance> scale) {

continue Here;

}

//添加

List<Position3D> list;

if(output.containsKey(i)) {

list= output.get(i);

}else {

list= new ArrayList<>();

list.add(position3D);

}

list.add(inPosition3D);

output.put(i, list);

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.isolation;

**public** **class** MatrixIsolationFilter{

//带精度 多维矩阵中非有效成份筛选过滤算法

//Theory，模式识别，专家系统， 罗瑶光Yaoguang.Luo

//Application 罗瑶光

//scaleDistance 为 卷积筛选的精度， PCARatio为有效邻接总数的量比

//适用于 索贝尔 emboss等卷积处理过滤后，进行噪声过滤。增加精准度。

**public** **int**[][] getIsolationFilterMartix2D(**int**[][] input, **int** scaleDistance, **double** PCARatio){

**double** compareRate= Math.*pow*(1+ scaleDistance\* 2, 2)\* PCARatio;

**int**[][] output= **new** **int**[input.length][input[0].length];

**for**(**int** i= scaleDistance; i< input.length- scaleDistance; i++){

**for**(**int** j= scaleDistance; j< input[0].length- scaleDistance; j++){

**if**(0< input[i][j]) {

**int** PCA= 0;

**for**(**int** p= -scaleDistance; p< scaleDistance; p++) {

**for**(**int** q= -scaleDistance; q< scaleDistance; q++) {

**if**(input[p][q]== input[i][j]) {

PCA++;

}

}

}

//筛选过滤

**if**(PCA> compareRate) {

output[i][j]= input[i][j];

}

}

}

}

**return** output;

}

**public** **int**[][][] getIsolationFilterMartix3D(**int**[][][] input, **int** scaleDistance, **double** PCARatio){

**double** compareRate= Math.*pow*(1+ scaleDistance\* 2, 3)\* PCARatio;

**int**[][][] output= **new** **int**[input.length][input[0].length][input[0][0].length];

**for**(**int** i= scaleDistance; i< input.length- scaleDistance; i++){

**for**(**int** j= scaleDistance; j< input[0].length- scaleDistance; j++){

**for**(**int** k= scaleDistance; k< input[0].length- scaleDistance; k++){

**if**(0< input[i][j][k]) {

**int** PCA= 0;

**for**(**int** p= -scaleDistance; p< scaleDistance; p++) {

**for**(**int** q= -scaleDistance; q< scaleDistance; q++) {

**for**(**int** r= -scaleDistance; r< scaleDistance; r++) {

**if**(input[p][q][r]== input[i][j][r]) {

PCA++;

}

}

}

}

//筛选过滤

**if**(PCA> compareRate) {

output[i][j][k]= input[i][j][k];

}

}

}

}

}

**return** output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.issueDate;

**import** java.util.Date;

//Theory: Euclid

//Application: Yaoguang.luo

**public** **class** IsIssueDate{

**public** **boolean** isIssueDate(Date issueOut) {

**long** issueOutLong= issueOut.getTime();

**long** currentLong= **new** Date().getTime();

**return** currentLong< issueOutLong? **true**: **false**;

}

**public** **boolean** isIssueDatewithSwapGTC(Date issueOutGTC, **long** offsetUTC8) {

**long** issueOutLong= issueOutGTC.getTime();

**long** currentLong= **new** Date().getTime()+ offsetUTC8;

**return** currentLong< issueOutLong? **true**: **false**;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.NLP;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.deta.tinos.list.ListSwap;

import org.tinos.deta.statistic.LYG4DWithDoubleQuickSort4D;

import org.tinos.engine.analysis.Analyzer;

import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;

import org.tinos.view.obj.WordFrequency;

import org.tinos.view.stable.StableData;

public class NLPTopicMatch{

//<<NLP Algorithm of Matching The POS Scored Sentence>>.

//This prediction algorithm mostly used for matching the best sample sentence by using score method.

//Theory: Yaoguang.Luo 20191228

//Application: Yaoguang.Luo

//Attention: need Deta Parser API OSS

public static String NLPBestSentenceMatch(String searchString

, String[] sampleSentences) throws IOException {

//init the deta mixed parser engine.

Analyzer analyzer= new CogsBinaryForestAnalyzerImp();

analyzer.initMixed();

//init the nlp POS(part of speech) functions.

Map<String, String> nlp= analyzer.getPosCnToCn();

List<String> keys= analyzer.parserMixedString(searchString);

//find a appear frequency from the keys of search string.

Map<String, WordFrequency> keyMap= analyzer.getWordFrequencyMap(keys);

//get a POS score rights array from keyMap.

String[] stringKeys= ListSwap.listToArray(keys);

// I create a new algorithm of 'list to array' in my Data Swap Project. 20191228 Yaoguang. Luo

double[] scoreRights= getNLPBestSentencesMatchScoreRights(searchString

, sampleSentences, analyzer, nlp, keyMap);

//loop score array

double[] matchScore= new double[sampleSentences.length];

double max= 0;

int maxPoint= 0;

for(int i= 0; i< sampleSentences.length; i++) {

List<String> matchList= analyzer.parserMixedString(sampleSentences[i]);

Map<String, WordFrequency> matchMap= analyzer.getWordFrequencyMap(matchList);

for(int j= 0; j< stringKeys.length; j++) {

if(matchMap.containsKey(stringKeys[j])) {

matchScore[i]+= scoreRights[j]\* matchMap.get(stringKeys[j]).getFrequency();

}

}

if(max< matchScore[i]) {

max= matchScore[i];

maxPoint= i;

}

}

//output

return sampleSentences[maxPoint];

}

//<<NLP Algorithm of Matching The POS Scored Sentences>>.

//This prediction algorithm mostly used for matching the ranged sample sentences by using score method.

//Theory: Yaoguang.Luo 20191229

//Application: Yaoguang.Luo

//Attention: need Deta Parser API OSS

//sortStackRange: for the sort stacks filter scale

//filterRate: for how many sets need to delete

public static List<String> NLPBestSentencesMatch(String searchString

, String[] sampleSentences, int sortStackRange, double filterRate) throws IOException {

double[] matchScore= getNLPBestSentencesMatchScore(searchString, sampleSentences);

//get iden key

Map<Double, List<String>> tempBase= new HashMap<>();

for(int i= 0; i< matchScore.length; i++) {

List<String> tempList;

if(tempBase.containsKey(matchScore[i])) {

tempList= tempBase.get(matchScore[i]);

}else {

tempList= new ArrayList<>();

}

tempList.add(sampleSentences[i]);

tempBase.put(matchScore[i], tempList);

}

//sort

matchScore= new LYG4DWithDoubleQuickSort4D().sort(matchScore, sortStackRange);

double filterCount= filterRate\* matchScore.length;

//filter

List<String> output= new ArrayList<>();

for(int i= (int)filterCount; i< matchScore.length; i++) {

if(tempBase.containsKey(matchScore[i])) {

Iterator<String> iterator= tempBase.get(matchScore[i]).iterator();

while(iterator.hasNext()) {

output.add(iterator.next());

}

tempBase.remove(matchScore[i]);

}

}

//output

return output;

}

private static double[] getNLPBestSentencesMatchScore(String searchString

, String[] sampleSentences) throws IOException {

//init the deta mixed parser engine.

Analyzer analyzer= new CogsBinaryForestAnalyzerImp();

analyzer.initMixed();

//init the nlp POS(part of speech) functions.

Map<String, String> nlp= analyzer.getPosCnToCn();

List<String> keys= analyzer.parserMixedString(searchString);

//find a appear frequency from the keys of search string.

Map<String, WordFrequency> keyMap= analyzer.getWordFrequencyMap(keys);

//get a POS score rights array from keyMap.

String[] stringKeys= ListSwap.listToArray(keys);

double[] scoreRights= getNLPBestSentencesMatchScoreRights(searchString

, sampleSentences, analyzer, nlp, keyMap);

//loop score array

double[] matchScore= new double[sampleSentences.length];

for(int i= 0; i< sampleSentences.length; i++) {

List<String> matchList= analyzer.parserMixedString(sampleSentences[i]);

Map<String, WordFrequency> matchMap= analyzer.getWordFrequencyMap(matchList);

for(int j= 0; j< stringKeys.length; j++) {

if(matchMap.containsKey(stringKeys[j])) {

matchScore[i]+= scoreRights[j]\* matchMap.get(stringKeys[j]).getFrequency();

}

}

}

return matchScore;

}

private static double[] getNLPBestSentencesMatchScoreRights(String searchString

, String[] sampleSentences, Analyzer analyzer, Map<String, String> nlp

, Map<String, WordFrequency> keyMap) throws IOException {

double[] scoreRights= new double[keyMap.size()];

int scoreRightsPoint= 0;

Iterator<String> keyIterator= keyMap.keySet().iterator();

while(keyIterator.hasNext()) {

String key= keyIterator.next();

if(nlp.containsKey(key)) {

String pos= nlp.get(key);

//init rights of POS {30,20,10,3,1}

if(pos.contains(StableData.NLP\_CI\_MING)) {// n.

scoreRights[scoreRightsPoint]= 30\* keyMap.get(key).getFrequency();

}else if(pos.contains(StableData.NLP\_CI\_DONG)) {//v

scoreRights[scoreRightsPoint]= 20\* keyMap.get(key).getFrequency();

}else if(pos.contains(StableData.NLP\_CI\_XING\_RONG)) {//adj

scoreRights[scoreRightsPoint]= 10\* keyMap.get(key).getFrequency();

}else {

scoreRights[scoreRightsPoint]= 3\* keyMap.get(key).getFrequency();

}

}else {

scoreRights[scoreRightsPoint]= 1\* keyMap.get(key).getFrequency();

}

scoreRightsPoint++;

}

return scoreRights;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.PCA;

import java.util.Iterator;

import java.util.List;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

import org.tinos.deta.statistic.LYG4DWithDoubleQuickSort4D;

public class FindPCAMeanDistance{

//oberserverPCAScale：确定主要有效最短路径的观测数

//sortRangeScale：确定坐标距离排序时的相似成份多少来确定堆栈冗余比，避免堆栈溢出。

//求坐标团的主要有效距离成份集的平均压强算法

//适用于 观测/预测坐标集中某精度邻接坐标集的 平均距离 来确定 紧凑度，压力，压强，斥力等。

//思想：罗瑶光 20191225

//实现：罗瑶光

public static double findMeanDistanceFromPositions2D(List<Position2D> position2Ds

, double oberserverPCAScale, int sortRangeScale) {

Iterator<Position2D> outPosition2DIterator= position2Ds.iterator();

double outMean= 0;

while(outPosition2DIterator.hasNext()) {

Position2D outPosition2D= outPosition2DIterator.next();

double[] distance= new double[position2Ds.size()];

int i= 0;

Iterator<Position2D> inPosition2DIterator= position2Ds.iterator();

//取坐标点所有距离集合先

while(inPosition2DIterator.hasNext()) {

Position2D inPosition2D= inPosition2DIterator.next();

distance[i++]= Distance.getDistance2D(outPosition2D, inPosition2D);

}

//距离非对称缺陷小高峰过滤极速快排 从小到大

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//仅仅取精度内坐标点距离求平均值，0 为本身所以从 1 开始

double inMean= 0;

oberserverPCAScale= oberserverPCAScale>= position2Ds.size()

? position2Ds.size()- 1: oberserverPCAScale;

oberserverPCAScale= oberserverPCAScale< 0? 0: oberserverPCAScale;

for(i= 1; i<= oberserverPCAScale; i++) {

inMean+= distance[i];

}

inMean/= oberserverPCAScale;

outMean+= inMean;

}

return outMean/ position2Ds.size();

}

public static double findMeanDistanceFromPositions3D(List<Position3D> position3Ds

, double oberserverPCAScale, int sortRangeScale) {

Iterator<Position3D> outPosition3DIterator= position3Ds.iterator();

double outMean= 0;

while(outPosition3DIterator.hasNext()) {

Position3D outPosition3D= outPosition3DIterator.next();

double[] distance= new double[position3Ds.size()];

int i= 0;

Iterator<Position3D> inPosition3DIterator= position3Ds.iterator();

//取坐标点所有距离集合先

while(inPosition3DIterator.hasNext()) {

Position3D inPosition3D= inPosition3DIterator.next();

distance[i++]= Distance.getDistance3D(outPosition3D, inPosition3D);

}

//距离非对称缺陷小高峰过滤极速快排 从小到大

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//仅仅取精度内坐标点距离求平均值，0 为本身所以从 1 开始

double inMean= 0;

oberserverPCAScale= oberserverPCAScale>= position3Ds.size()

? position3Ds.size()- 1: oberserverPCAScale;

oberserverPCAScale= oberserverPCAScale< 0? 0: oberserverPCAScale;

for(i= 1; i<= oberserverPCAScale; i++) {

inMean+= distance[i];

}

inMean/= oberserverPCAScale;

outMean+= inMean;

}

return outMean/ position3Ds.size();

}

//2D坐标团中 每一个坐标的斥力预测算法。

public static double[] findPascalMeanDistanceByEachPositions2D(List<Position2D> position2Ds

, double oberserverPCAScale, int sortRangeScale) {

double[] pascal= new double[position2Ds.size()];

Iterator<Position2D> outPosition2DIterator= position2Ds.iterator();

double outMean= 0;

int positionPoint=0;

while(outPosition2DIterator.hasNext()) {

Position2D outPosition2D= outPosition2DIterator.next();

double[] distance= new double[position2Ds.size()];

int i= 0;

Iterator<Position2D> inPosition2DIterator= position2Ds.iterator();

//取坐标点所有距离集合先

while(inPosition2DIterator.hasNext()) {

Position2D inPosition2D= inPosition2DIterator.next();

distance[i++]= Distance.getDistance2D(outPosition2D, inPosition2D);

}

//距离非对称缺陷小高峰过滤极速快排 从小到大

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//仅仅取精度内坐标点距离求平均值，0 为本身所以从 1 开始

double inMean= 0;

oberserverPCAScale= oberserverPCAScale>= position2Ds.size()

? position2Ds.size()- 1: oberserverPCAScale;

oberserverPCAScale= oberserverPCAScale< 0? 0: oberserverPCAScale;

for(i= 1; i<= oberserverPCAScale; i++) {

inMean+= distance[i];

}

inMean/= oberserverPCAScale;

outMean+= inMean;

//采集坐标点形成斥力参照的主要最短路径成份。

pascal[positionPoint++]= inMean;

}

outMean/= position2Ds.size();

for(int i= 0; i< pascal.length; i++) {

//比值获取 不稳定 观测数据。

pascal[i]/= outMean;

}

return pascal;

}

//3D坐标团中 每一个坐标的斥力预测算法。

public static double[] findPascalMeanDistanceByEachPositions3D(List<Position3D> position3Ds

, double oberserverPCAScale, int sortRangeScale) {

double[] pascal= new double[position3Ds.size()];

Iterator<Position3D> outPosition3DIterator= position3Ds.iterator();

double outMean= 0;

int positionPoint=0;

while(outPosition3DIterator.hasNext()) {

Position3D outPosition3D= outPosition3DIterator.next();

double[] distance= new double[position3Ds.size()];

int i= 0;

Iterator<Position3D> inPosition3DIterator= position3Ds.iterator();

//取坐标点所有距离集合先

while(inPosition3DIterator.hasNext()) {

Position3D inPosition3D= inPosition3DIterator.next();

distance[i++]= Distance.getDistance3D(outPosition3D, inPosition3D);

}

//距离非对称缺陷小高峰过滤极速快排 从小到大

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//仅仅取精度内坐标点距离求平均值，0 为本身所以从 1 开始

double inMean= 0;

oberserverPCAScale= oberserverPCAScale>= position3Ds.size()

? position3Ds.size()- 1: oberserverPCAScale;

oberserverPCAScale= oberserverPCAScale< 0? 0: oberserverPCAScale;

for(i= 1; i<= oberserverPCAScale; i++) {

inMean+= distance[i];

}

inMean/= oberserverPCAScale;

outMean+= inMean;

//采集坐标点形成斥力参照的主要最短路径成份。

pascal[positionPoint++]= inMean;

}

outMean/= position3Ds.size();

for(int i= 0; i< pascal.length; i++) {

//比值获取 不稳定 观测数据。

pascal[i]/= outMean;

}

return pascal;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.PCA;

import java.util.HashMap;

import java.util.Iterator;

import java.util.ArrayList;

import java.util.Map;

import org.tinos.deta.classification.FissileWithMatch;

import org.tinos.deta.demension.FindHeartPositions;

import org.tinos.deta.demension.FindMidPositions;

import org.tinos.deta.demension.Line2D;

import org.tinos.deta.demension.Line3D;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class FindPositionsGroupPascalHearts{

//通过坐标团的 精度匹配分割的内部坐标聚类团 进行 每个聚类团的 重心和中心距离 求解 获取有效的团稳定观测数据模型

//思想 帕斯卡，欧基里德，数据挖掘，贝叶斯，立体几何，流体力学，量化拓扑力学，离散数学，统计与概率论 20191227

//实现 罗瑶光

public static Map<Double, Position2D> getPosition2DsGroupPascalHearts(ArrayList<Position2D> groups, double scale) {

Map<Double, ArrayList<Position2D>> pascalGroups

= FissileWithMatch.fissilePosition2DWithMatch(groups, scale);

Map<Double, Position2D> pascalHearts

= FindHeartPositions.getPosition2DGroupsHearts(pascalGroups);

return pascalHearts;

}

public static Map<Double, Position3D> getPosition3DsGroupPascalHearts(ArrayList<Position3D> groups, double scale) {

Map<Double, ArrayList<Position3D>> pascalGroups

= FissileWithMatch.fissilePosition3DWithMatch(groups, scale);

Map<Double, Position3D> pascalHearts

= FindHeartPositions.getPosition3DGroupsHearts(pascalGroups);

return pascalHearts;

}

public static Map<Double, Position2D> getPosition2DsGroupPascalMids(ArrayList<Position2D> groups, double scale) {

Map<Double, ArrayList<Position2D>> pascalGroups

= FissileWithMatch.fissilePosition2DWithMatch(groups, scale);

Map<Double, Position2D> pascalMids= FindMidPositions.getPosition2DGroupsMids(pascalGroups);

return pascalMids;

}

public static Map<Double, Position3D> getPosition3DsGroupPascalMids(ArrayList<Position3D> groups, double scale) {

Map<Double, ArrayList<Position3D>> pascalGroups

= FissileWithMatch.fissilePosition3DWithMatch(groups, scale);

Map<Double, Position3D> pascalMids= FindMidPositions.getPosition3DGroupsMids(pascalGroups);

return pascalMids;

}

public static Map<Double, Line2D> getPosition2DsGroupPascalDirection(Map<Double, Position2D> pascalHearts

, Map<Double, Position2D> pascalMids) {

Map<Double, Line2D> pascalDirections= new HashMap<>();

Iterator<Double> iterator= pascalHearts.keySet().iterator();

while(iterator.hasNext()) {

double key= iterator.next();

Line2D value= new Line2D();

value.setBegin(pascalMids.get(key));

value.setEnd(pascalHearts.get(key));

pascalDirections.put(key, value);

}

return pascalDirections;

}

public static Map<Double, Line3D> getPosition3DsGroupPascalDirection(Map<Double, Position3D> pascalHearts

, Map<Double, Position3D> pascalMids) {

Map<Double, Line3D> pascalDirections= new HashMap<>();

Iterator<Double> iterator= pascalHearts.keySet().iterator();

while(iterator.hasNext()) {

double key= iterator.next();

Line3D value= new Line3D();

value.setBegin(pascalMids.get(key));

value.setEnd(pascalHearts.get(key));

pascalDirections.put(key, value);

}

return pascalDirections;

}

public static Map<Double, Line3D> getPosition3DsGroupPascalDirection(ArrayList<Position3D> groups, double scale){

return getPosition3DsGroupPascalDirection(getPosition3DsGroupPascalHearts(groups, scale)

,getPosition3DsGroupPascalMids(groups, scale));

}

public static Map<Double, Line2D> getPosition2DsGroupPascalDirection(ArrayList<Position2D> groups, double scale){

return getPosition2DsGroupPascalDirection(getPosition2DsGroupPascalHearts(groups, scale)

,getPosition2DsGroupPascalMids(groups, scale));

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.PCA;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//这个函数用于坐标融聚团的主要条件过滤。

//思想：统计与概率论，立体几何，数据挖掘

//实现：罗瑶光

public class FusionPCAFilter{

public static Map<Double, List<Position2D>> filterFusion2DSetsWithCountScale

(Map<Double, List<Position2D>> groups, double countScale){

Map<Double, List<Position2D>> output= new HashMap<>();

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

double value= iterator.next();

if(groups.get(value).size()>= countScale) {

output.put(value, groups.get(value));

}

}

return output;

}

public static Map<Double, List<Position3D>> filterFusion3DSetsWithCountScale

(Map<Double, List<Position3D>> groups, double countScale){

Map<Double, List<Position3D>> output= new HashMap<>();

Iterator<Double> iterator= groups.keySet().iterator();

while(iterator.hasNext()) {

double value= iterator.next();

if(groups.get(value).size()>= countScale) {

output.put(value, groups.get(value));

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.PCA;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.tinos.deta.basic.RatioMatrix;

//这个函数用于求模糊概率集平均值采样主要成份分析

//思想：贝叶斯 模糊数学 统计于概率论

//实现：罗瑶光

public class PCAMeanOfFuzzPC{

public static RatioMatrix getSimilarFuzzSetWithScale(RatioMatrix input, List<RatioMatrix> groups, double scale) {

List<RatioMatrix> output= new ArrayList<>();

Iterator<RatioMatrix> iterators= groups.iterator();

while(iterators.hasNext()) {

RatioMatrix ratio= iterators.next();

double rightRightMean= ratio.getRightRightRatio();

double rightErrorMean= ratio.getRightErrorRatio();

double errorRightMean= ratio.getErrorRightRatio();

double errorErrorMean= ratio.getErrorErrorRatio();

double predictionRightRight= Math.abs(input.getRightRightRatio()- rightRightMean);

double predictionRightError= Math.abs(input.getRightErrorRatio()- rightErrorMean);

double predictionErrorRight= Math.abs(input.getErrorRightRatio()- errorRightMean);

double predictionErrorError= Math.abs(input.getErrorErrorRatio()- errorErrorMean);

double tempSumRight= predictionRightRight+ predictionRightError+ predictionErrorRight+ predictionErrorError;

if(tempSumRight< scale) {

output.add(ratio);

}

}

RatioMatrix outputMean= new RatioMatrix();

Iterator<RatioMatrix> iteratorsOutput= output.iterator();

while(iteratorsOutput.hasNext()) {

RatioMatrix ratio= iteratorsOutput.next();

outputMean.setErrorErrorRatio(ratio.getErrorErrorRatio()+ outputMean.getErrorErrorRatio());

outputMean.setErrorRightRatio(ratio.getErrorRightRatio()+ outputMean.getErrorRightRatio());

outputMean.setRightErrorRatio(ratio.getRightErrorRatio()+ outputMean.getRightErrorRatio());

outputMean.setRightRightRatio(ratio.getRightRightRatio()+ outputMean.getRightRightRatio());

}

outputMean.setErrorErrorRatio(outputMean.getErrorErrorRatio()/ output.size());

outputMean.setErrorRightRatio(outputMean.getErrorRightRatio()/ output.size());

outputMean.setRightErrorRatio(outputMean.getRightErrorRatio()/ output.size());

outputMean.setRightRightRatio(outputMean.getRightRightRatio()/ output.size());

return outputMean;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.PCA;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Map;

**import** org.tinos.deta.basic.Distance;

**import** org.tinos.deta.basic.Euclid;

**import** org.tinos.deta.demension.Position2D;

**import** org.tinos.deta.demension.Position3D;

//这个函数用于坐标类的 主要成份分析与提取

//思想：欧基里德 平面，立体几何

//实现：罗瑶光

**public** **class** PCAPositionFilter{

**public** **static** List<Position2D> filterPosition2DsWithScaledDistance(List<Position2D> input

, Position2D heart, **double** scaleDistacne){

List<Position2D> output= **new** ArrayList<>();

Iterator<Position2D> iterator= input.iterator();

**while**(iterator.hasNext()) {

Position2D position2D= iterator.next();

**double** distance= Distance.*getDistance2D*(heart, position2D);

**if**(distance< scaleDistacne) {

output.add(position2D);

}

}

**return** output;

}

**public** **static** List<Position3D> filterPosition3DsWithScaledDistance(List<Position3D> input

, Position3D heart, **double** scaleDistacne){

List<Position3D> output= **new** ArrayList<>();

Iterator<Position3D> iterator= input.iterator();

**while**(iterator.hasNext()) {

Position3D position3D= iterator.next();

**double** distance= Distance.*getDistance3D*(heart, position3D);

**if**(distance< scaleDistacne) {

output.add(position3D);

}

}

**return** output;

}

**public** **static** Map<Double, List<Position2D>> filterPosition2DsWithScaledDistance(Map<Double, List<Position2D>> input

, Position2D heart, **double** scaleDistacne){

Map<Double, List<Position2D>> output= **new** HashMap<>();

Iterator<Double> iterator= input.keySet().iterator();

**while**(iterator.hasNext()) {

**double** mapKey= iterator.next();

List<Position2D> position2DList= input.get(mapKey);

Position2D tempHeart= Euclid.*findHeartPosition2D*(position2DList);

**double** distance= Distance.*getDistance2D*(heart, tempHeart);

**if**(distance< scaleDistacne) {

output.put(mapKey, position2DList);

}

}

**return** output;

}

**public** **static** Map<Double, List<Position3D>> filterPosition3DsWithScaledDistance(Map<Double, List<Position3D>> input

, Position3D heart, **double** scaleDistacne){

Map<Double, List<Position3D>> output= **new** HashMap<>();

Iterator<Double> iterator= input.keySet().iterator();

**while**(iterator.hasNext()) {

**double** mapKey= iterator.next();

List<Position3D> position3DList= input.get(mapKey);

Position3D tempHeart= Euclid.*findHeartPosition3D*(position3DList);

**double** distance= Distance.*getDistance3D*(heart, tempHeart);

**if**(distance< scaleDistacne) {

output.put(mapKey, position3DList);

}

}

**return** output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.ratio;

import org.tinos.deta.basic.ComputeSets;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

//Theory: Yaoguang.luo

//一种仅仅通过坐标差值叠加来计算距离向量的使用方法。

//Application: Yaoguang.luo

public class DistanceRatio{

//获取斜率梯度

public static double getDistanceRatio2D(Position2D begin, Position2D end) {

double x= begin.getX()- end.getX();

double y= begin.getY()- end.getY();

//1象限

if(x>=0&& y>=0) {

return Math.abs(x)/ Math.abs(y);

}

//4象限

if(x>= 0 && y<0) {

return 1+ Math.abs(x)/ Math.abs(y);

}

//2象限

if(x< 0 && y>= 0) {

return Math.abs(x)/ Math.abs(y);

}

//3象限

if(x< 0&& y< 0) {

return -1 - Math.abs(x)/ Math.abs(y);

}

return 0;

}

//降低计算速度来获取高准确斜率梯度 (早期傅里叶思想)

public static double getARCDistanceRatio2D(Position2D begin, Position2D end) {

double x= begin.getX()- end.getX();

double y= begin.getY()- end.getY();

double z= Math.sqrt(x\* x+ y\* y);

return Math.asin(x/z)+ Math.acos(y/z);

}

//降低计算速度来获取两点间线的真实角度

public static double getTrueARCDistanceRatio2D(Position2D begin, Position2D end) {

double x= begin.getX()- end.getX();

double y= begin.getY()- end.getY();

double z= Math.sqrt(x\* x+ y\* y);

//1象限

if(x>=0&& y>=0) {

return Math.asin(Math.abs(x)/z);

}

//4象限

if(x>= 0 && y<0) {

return 2\* ComputeSets.getPi()- Math.asin(Math.abs(x)/z);

}

//2象限

if(x< 0 && y>= 0) {

return ComputeSets.getPi()- Math.asin(Math.abs(x)/z);

}

//3象限

if(x< 0&& y< 0) {

return ComputeSets.getPi()+ Math.asin(Math.abs(x)/z);

}

return 0;

}

//获取计算参照梯度

public static double getDistanceRatio3D(Position3D begin, Position3D end) {

double pi= 3.1415926;

double x= begin.getX()- end.getX();

double y= begin.getY()- end.getY();

double z= begin.getZ()- end.getZ();

//1象限

if(x>= 0&& y>= 0) {

if(z>= 0) {

return Math.abs(x)/ Math.abs(y);

}else {

return pi+ Math.abs(x)/ Math.abs(y);

}

}

//4象限

if(x>= 0&& y< 0) {

if(z>= 0) {

return pi\*6+ Math.abs(x)/ Math.abs(y);

}else {

return pi\*7+ Math.abs(x)/ Math.abs(y);

}

}

//2象限

if(x< 0&& y>= 0) {

if(z>= 0) {

return pi\*2+ Math.abs(x)/ Math.abs(y);

}else {

return pi\*3+ Math.abs(x)/ Math.abs(y);

}

}

//3象限

if(x< 0&& y< 0) {

if(z>= 0) {

return pi\*4+ Math.abs(x)/ Math.abs(y);

}else {

return pi\*5+ Math.abs(x)/ Math.abs(y);

}

}

return 0;

}

//获取真实三维夹角

public static double getTrueARCDistanceRatio3D(Position3D begin, Position3D end) {

return 0;

//球面参照无效。以后研究下有什么标准规范没。

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.tinos.deta.statistic;

//基于算法导论快排4衍生极速小高峰缺陷过滤理论快速排序第4代 线性数字数组排序法函数Java完整版本实现。

//思想：算法导论快排4理论，罗瑶光小高峰过滤理论。

//实现：罗瑶光

//时间：20140101~ 20191105

**public** **class** LYG4DWithDoubleQuickSort4D{

**int** range;

**public** **double**[] sort(**double**[] array, **int** range) {

**this**.range= range< 1? 1: range;

processDouble(array, 0, array.length- 1);

**return** array;

}

**private** **void** processDouble(**double**[] array, **int** leftPoint, **int** rightPoint) {

**if**(leftPoint< rightPoint){

**int** c= rightPoint- leftPoint;

**if**(c< **this**.range){

**int** j;

**for**(**int** i= 1+ leftPoint; i<= leftPoint+ c; i++){

j= i;

**while**(j>= 1+ leftPoint){

**if**(array[j]< array[j- 1]){

**double** temp= array[j];

array[j]= array[j- 1];

array[j- 1]= temp;

}

j--;

}

}

**return**;

}

**int** pos= partition(array, leftPoint, rightPoint);

processDouble(array, leftPoint, pos- 1);

processDouble(array, pos+ 1, rightPoint);

}

}

**private** **int** partition(**double**[] array, **int** leftPoint, **int** rightPoint) {

**double** x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];

**int** leftPointReflection= leftPoint;

**while**(leftPointReflection< rightPoint){

**while**(!(array[leftPointReflection]> x|| leftPointReflection>= rightPoint)) {

leftPointReflection++;

}

**while**(array[rightPoint]> x){

rightPoint--;

}

**if**(leftPointReflection< rightPoint){

**double** temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

**return** rightPoint;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.trace;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class TraceFissilePositionHearts{

//Source: 《2维 3维 坐标集 切裂 重心 轨迹 跟踪算法JAVA源码》

//出版日期2019 年 12 月 21 日 作品说明Gitee, Github, DetaOSS

//作品说明适用于 坐标团 动态分析，增量轨迹分析，熵增信息单元记录。

//作者 罗瑶光

public static Map<Double, List<Position2D>>

trackTracePosition2DHeartsWithFissileGroups(List<Position2D> coods, double distanceScale){

Map<Double, List<Position2D>> output= new HashMap<>();

Map<Double, List<Position2D>> distanceGroups= new HashMap<>();

Iterator<Position2D> iterator= coods.iterator();

double i= 0.0;

Here:

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

if(distanceGroups.isEmpty()) {

List<Position2D> list= new ArrayList<>();

list.add(position2D);

distanceGroups.put(i, list);

//

List<Position2D> listHeartsTrace= output.get(i);

listHeartsTrace.add(position2D);

output.put(i, listHeartsTrace);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= output.keySet().iterator();

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position2D currenctHeart= output.get(doubleScale).get(output.get(doubleScale).size()- 1);

double distance= Distance.getDistance2D(currenctHeart, position2D);

if(distance< distanceScale) {

//融入得到新的重心

Position2D newHeart= Euclid.findCryptionPosition2D(currenctHeart

, position2D);

//删除当前增加坐标集，更新坐标集

List<Position2D> list= distanceGroups.get(doubleScale);

list.add(position2D);

distanceGroups.put(doubleScale, list);

//删除当前重心数据，更新重心数据轨迹

List<Position2D> listHeartsTrace= output.get(doubleScale);

listHeartsTrace.add(newHeart);

output.put(doubleScale, listHeartsTrace);

//找到

isFind= true;

//如果预测 坐标适应状态 可以把 continue 省略。

continue Here;

}

}

//新存

if(!isFind) {

List<Position2D> list= new ArrayList<>();

list.add(position2D);

distanceGroups.put(++i, list);

//加新hearts

List<Position2D> listHeartsTrace= output.get(i);

listHeartsTrace.add(position2D);

output.put(i, listHeartsTrace);

}

}

}

return output;

}

public static Map<Double, List<Position3D>>

trackTracePosition3DHeartsWithFissileGroups(List<Position3D> coods, double distanceScale){

Map<Double, List<Position3D>> output= new HashMap<>();

Map<Double, List<Position3D>> distanceGroups= new HashMap<>();

Iterator<Position3D> iterator= coods.iterator();

double i= 0.0;

Here:

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

if(distanceGroups.isEmpty()) {

List<Position3D> list= new ArrayList<>();

list.add(position3D);

distanceGroups.put(i, list);

//

List<Position3D> listHeartsTrace= output.get(i);

listHeartsTrace.add(position3D);

output.put(i, listHeartsTrace);

}else {

//遍历所有团

//团重心匹配如果超精度新存，不是就融入。

Iterator<Double> iteratorScale= output.keySet().iterator();

boolean isFind= false;

while(iteratorScale.hasNext()) {

Double doubleScale= iteratorScale.next();

Position3D currenctHeart

= output.get(doubleScale).get(output.get(doubleScale).size()- 1);

double distance= Distance.getDistance3D(currenctHeart, position3D);

if(distance< distanceScale) {

//融入得到新的重心

Position3D newHeart

= Euclid.findCryptionPosition3D(currenctHeart, position3D);

//删除当前增加坐标集，更新坐标集

List<Position3D> list= distanceGroups.get(doubleScale);

list.add(position3D);

distanceGroups.put(doubleScale, list);

//删除当前重心数据，更新重心数据轨迹

List<Position3D> listHeartsTrace= output.get(doubleScale);

listHeartsTrace.add(newHeart);

output.put(doubleScale, listHeartsTrace);

//找到

isFind= true;

//如果预测 坐标适应状态 可以把 continue 省略。

continue Here;

}

}

//新存

if(!isFind) {

List<Position3D> list= new ArrayList<>();

list.add(position3D);

distanceGroups.put(++i, list);

//加新hearts

List<Position3D> listHeartsTrace= output.get(i);

listHeartsTrace.add(position3D);

output.put(i, listHeartsTrace);

}

}

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.trace;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.tinos.deta.basic.Euclid;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

public class TracePositionHearts{

//Source: 《2维 3维 坐标集 切裂 重心 轨迹 跟踪算法JAVA源码》

//出版日期2019 年 12 月 21 日 作品说明Gitee, Github, DetaOSS

//作品说明适用于 坐标团 动态分析，增量轨迹分析，熵增信息单元记录。

//作者 罗瑶光

public static List<Position2D> trackTracePosition2DHeartsWithSingerGroup(List<Position2D> coods){

List<Position2D> hearts= new ArrayList<>();

Iterator<Position2D> iterator= coods.iterator();

boolean isFirst= true;

while(iterator.hasNext()) {

if(isFirst) {

isFirst= !isFirst;

hearts.add(iterator.next());

}else {

Position2D position2D= iterator.next();

Position2D CryptHeart

= Euclid.findCryptionPosition2D(hearts.get(hearts.size()- 1), position2D);

hearts.add(CryptHeart);

}

}

return hearts;

}

public static List<Position3D> trackTracePosition3DHeartsWithSingerGroup(List<Position3D> coods){

List<Position3D> hearts= new ArrayList<>();

Iterator<Position3D> iterator= coods.iterator();

boolean isFirst= true;

while(iterator.hasNext()) {

if(isFirst) {

isFirst= !isFirst;

hearts.add(iterator.next());

}else {

Position3D position3D= iterator.next();

Position3D CryptHeart

= Euclid.findCryptionPosition3D(hearts.get(hearts.size()- 1), position3D);

hearts.add(CryptHeart);

}

}

return hearts;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.tsp;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.demension.Line2D;

import org.tinos.deta.demension.Line3D;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

import org.tinos.deta.statistic.LYG4DWithDoubleQuickSort4D;

public class YaoguangLuoEulerRingTSP{

//Before I left L.A and went to Folsom, I did a quick TSP version at that time

//at rosemead, it seems not fast. but now will be the best around this real world.

//This Algorithm Theory as a new year gift to my CHRISTINA.

//Foundation: Euler

//Theory: Yaoguang.Luo

//Application: Yaoguang.Luo 20200112

public List<Line2D> getYaoguangLuo2DEulerRingTSP(List<Position2D> positions){

//1 annotations

List<Position2D> position2DTag= new ArrayList<>();

Iterator<Position2D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

position2D.setTag(tag+ i++);

position2DTag.add(position2D);

}

positions= position2DTag;

//2 get all lines

List<Line2D> linesMap= new ArrayList<>();

Iterator<Position2D> iteratorOuter= positions.iterator();

i= 0;

while(iteratorOuter.hasNext()) {

Position2D position2DOuter= iteratorOuter.next();

Iterator<Position2D> iteratorInner= positions.iterator();

while(iteratorInner.hasNext()) {

Position2D position2DInner= iteratorOuter.next();

Line2D line2D= new Line2D();

line2D.setBegin(position2DOuter);

line2D.setEnd(position2DInner);

linesMap.add(line2D);

}

}

//3 sort line2D

double[] distance= new double[positions.size()];

Iterator<Line2D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line2D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line2D line2D= linesKeySets.next();

double distanceDouble= Distance.getDistance2D(line2D.getBegin(), line2D.getEnd());

List<Line2D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line2D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

int sortRangeScale= 4; //my default is 4. you should change it as your want.

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//7 From small to big loop the distance and make a condition tree.

List<Line2D> outputLine2D= new ArrayList<>();

Map<String, Double> outputDouble2D= new HashMap<>();

for(i= 0; i< distance.length; i++) {

List<Line2D> list= uniqueLines.get(distance[i]);

Iterator<Line2D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line2D line2D= iteratorLines.next();

Position2D begin= line2D.getBegin();

Position2D end= line2D.getEnd();

//8 decision tree add rights line

if(outputDouble2D.containsKey(begin.getTag())) {

double beginTimes= outputDouble2D.get(begin.getTag()).doubleValue();

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), 1.0);

}

}

outputLine2D.add(line2D);

}

}

return outputLine2D;

}

public List<Line3D> getYaoguangLuo3DEulerRingTSP(List<Position3D> positions){

//1 annotations

List<Position3D> position3DTag= new ArrayList<>();

Iterator<Position3D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

position3D.setTag(tag+ i++);

position3DTag.add(position3D);

}

positions= position3DTag;

//2 get all lines

List<Line3D> linesMap= new ArrayList<>();

Iterator<Position3D> iteratorOuter= positions.iterator();

i= 0;

while(iteratorOuter.hasNext()) {

Position3D position3DOuter= iteratorOuter.next();

Iterator<Position3D> iteratorInner= positions.iterator();

while(iteratorInner.hasNext()) {

Position3D position3DInner= iteratorOuter.next();

Line3D line3D= new Line3D();

line3D.setBegin(position3DOuter);

line3D.setEnd(position3DInner);

linesMap.add(line3D);

}

}

//3 sort line3D

double[] distance= new double[positions.size()];

Iterator<Line3D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line3D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line3D line3D= linesKeySets.next();

double distanceDouble= Distance.getDistance3D(line3D.getBegin(), line3D.getEnd());

List<Line3D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line3D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

int sortRangeScale= 4; //my default is 4. you should change it as your want.

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//7 From small to big loop the distance and make a condition tree.

List<Line3D> outputLine3D= new ArrayList<>();

Map<String, Double> outputDouble3D= new HashMap<>();

for(i= 0; i< distance.length; i++) {

List<Line3D> list= uniqueLines.get(distance[i]);

Iterator<Line3D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line3D line3D= iteratorLines.next();

Position3D begin= line3D.getBegin();

Position3D end= line3D.getEnd();

//8 decision tree add rights line

if(outputDouble3D.containsKey(begin.getTag())) {

double beginTimes= outputDouble3D.get(begin.getTag()).doubleValue();

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), 1.0);

}

}

outputLine3D.add(line3D);

}

}

return outputLine3D;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.tinos.deta.tsp;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.tinos.deta.basic.Distance;

import org.tinos.deta.demension.Line2D;

import org.tinos.deta.demension.Line3D;

import org.tinos.deta.demension.Position2D;

import org.tinos.deta.demension.Position3D;

import org.tinos.deta.statistic.LYG4DWithDoubleQuickSort4D;

public class YaoguangLuoEulerRingTSP2D{

//Foundation: Euler

//Theory: Yaoguang.Luo

//Application: Yaoguang.Luo 20200114

public List<Line2D> getYaoguangLuo2DEulerRingTSP2D(List<Position2D> positions){

//1 annotations

List<Position2D> position2DTag= new ArrayList<>();

Iterator<Position2D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

Position2D position2D= iterator.next();

position2D.setTag(tag+ i++);

position2DTag.add(position2D);

}

positions= position2DTag;

//2 get all lines

List<Line2D> linesMap= new ArrayList<>();

Iterator<Position2D> iteratorOuter= positions.iterator();

Map<String, Map<String, String>> indexMap= new HashMap<>();

i= 0;

while(iteratorOuter.hasNext()) {

Position2D position2DOuter= iteratorOuter.next();

Iterator<Position2D> iteratorInner= positions.iterator();

Next:

while(iteratorInner.hasNext()) {

Position2D position2DInner= iteratorOuter.next();

Line2D line2D= new Line2D();

line2D.setBegin(position2DOuter);

line2D.setEnd(position2DInner);

//2.1 delete the De-reflection redundant lines

if(indexMap.containsKey(position2DInner.getTag())) {

continue Next;

}

//2.2 delete self positions lines

if(!(position2DOuter.getX()!=position2DInner.getX()

|| position2DOuter.getY()!=position2DInner.getY())) {

continue Next;

}

Map<String, String> map= new HashMap<>();

if(indexMap.containsKey(position2DOuter.getTag())) {

map= indexMap.get(position2DOuter.getTag());

}else {

map= new HashMap<>();

}

map.put(position2DInner.getTag(), "");

indexMap.put(position2DOuter.getTag(), map);

linesMap.add(line2D);

}

}

//3 sort line2D

double[] distance= new double[positions.size()];

Iterator<Line2D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line2D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line2D line2D= linesKeySets.next();

double distanceDouble= Distance.getDistance2D(line2D.getBegin(), line2D.getEnd());

List<Line2D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line2D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

int sortRangeScale= 4; //my default is 4. you should change it as your want.

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//7 From small to big loop the distance and make a condition tree.

List<Line2D> outputLine2D= new ArrayList<>();

Map<String, Double> outputDouble2D= new HashMap<>();

for(i= 0; i< distance.length; i++) {

List<Line2D> list= uniqueLines.get(distance[i]);

Iterator<Line2D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line2D line2D= iteratorLines.next();

Position2D begin= line2D.getBegin();

Position2D end= line2D.getEnd();

//8 decision tree add rights line

if(outputDouble2D.containsKey(begin.getTag())) {

double beginTimes= outputDouble2D.get(begin.getTag()).doubleValue();

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), 1.0);

}

}

outputLine2D.add(line2D);

}

}

return outputLine2D;

}

public List<Line3D> getYaoguangLuo3DEulerRingTSP2D(List<Position3D> positions){

//1 annotations

List<Position3D> position3DTag= new ArrayList<>();

Iterator<Position3D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

Position3D position3D= iterator.next();

position3D.setTag(tag+ i++);

position3DTag.add(position3D);

}

positions= position3DTag;

//2 get all lines

List<Line3D> linesMap= new ArrayList<>();

Iterator<Position3D> iteratorOuter= positions.iterator();

Map<String, Map<String, String>> indexMap= new HashMap<>();

i= 0;

while(iteratorOuter.hasNext()) {

Position3D position3DOuter= iteratorOuter.next();

Iterator<Position3D> iteratorInner= positions.iterator();

Next:

while(iteratorInner.hasNext()) {

Position3D position3DInner= iteratorOuter.next();

Line3D line3D= new Line3D();

line3D.setBegin(position3DOuter);

line3D.setEnd(position3DInner);

//2.1 delete the De-reflection redundant lines

if(indexMap.containsKey(position3DInner.getTag())) {

continue Next;

}

//2.2 delete self positions lines

if(!(position3DOuter.getX()!= position3DInner.getX()

|| position3DOuter.getY()!= position3DInner.getY()

|| position3DOuter.getZ()!= position3DInner.getZ())) {

continue Next;

}

Map<String, String> map= new HashMap<>();

if(indexMap.containsKey(position3DOuter.getTag())) {

map= indexMap.get(position3DOuter.getTag());

}else {

map= new HashMap<>();

}

map.put(position3DInner.getTag(), "");

indexMap.put(position3DOuter.getTag(), map);

linesMap.add(line3D);

}

}

//3 sort line3D

double[] distance= new double[positions.size()];

Iterator<Line3D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line3D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line3D line3D= linesKeySets.next();

double distanceDouble= Distance.getDistance3D(line3D.getBegin(), line3D.getEnd());

List<Line3D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line3D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

int sortRangeScale= 4; //my default is 4. you should change it as your want.

distance= new LYG4DWithDoubleQuickSort4D().sort(distance, sortRangeScale);

//7 From small to big loop the distance and make a condition tree.

List<Line3D> outputLine3D= new ArrayList<>();

Map<String, Double> outputDouble3D= new HashMap<>();

for(i= 0; i< distance.length; i++) {

List<Line3D> list= uniqueLines.get(distance[i]);

Iterator<Line3D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line3D line3D= iteratorLines.next();

Position3D begin= line3D.getBegin();

Position3D end= line3D.getEnd();

//8 decision tree add rights line

if(outputDouble3D.containsKey(begin.getTag())) {

double beginTimes= outputDouble3D.get(begin.getTag()).doubleValue();

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), 1.0);

}

}

outputLine3D.add(line3D);

}

}

return outputLine3D;

}

}