Topic 4: Topological Relationships

Summary

# Motivation

Topological predicates (overlap, meet, inside etc) between spatial objects (point, line, region) has been a focus of interdisciplinary research (cognitive sciences, artificial intelligence, linguistics, robotics, spatial reasoning) for spatial data handling, reasoning and query languages.The motivation behind this project is to provide support for the evaluation of topological relationships, which are valuable tools for understanding the relations between two spatial objects.Topological relationships between spatial objects helps in formulating query language to retrieve and analyze spatial database systems and Geographic Information systems as well as in many interdisciplinary fields.So this project aims at implementing efficient algorithm to quantify the topological relationships between complex spatial objects.

Goals

The goal of this project is

* To implement topological predicates and methods to prevent repetitive predicate executions.
* To present efficient implementation strategies for topological predicates between all combinations of the three complex spatial data types point2D, line2D, and region2D.
* To present systematic, correct, robust and efficient implementation strategies and optimized exploration and evaluation methods for topological predicates between all combinations of complex spatial data types for line, point and region objects.
* To come up with topological information which is specific and suitable for verification query and determination query
* To implement a universal solution for all spatial type systems offering simple/complex spatial data types and for all complete collections of mutually exclusive topological predicates based on 9-intersection model using matrix thinning algorithm.

# Problems to solve

1. Determine a representation of how two objects are related. Based on this representation, classify the relationship in a meaningful way.
2. Formulate a solution to implement meaningful topological predicates between combinations of complex spatial objects namely point2D, line2D and region2D.
3. The type of query which is posed is also a problem and needs to be addressed:

* Do A & B (spatial objects of type line, point or region) satisfy topological predicate p? => this is called a verification query
* What is the topological predicate p between A & B? - it returns a predicate name => this is called determination query
* For both type of queries dimension of intersection can also be included

# Interface:

Data types (classes and operations which other groups may need to provide to us):

We assume these are the following interface declarations.

1. Object data types/classes: point2D; line2D; region2D.
   1. enum predicates {collinear, equal, disjoint, meet};
   2. class object2D { … };
   3. class poi2D: public object2D { … };
   4. class point2D: public poi2D { … };
   5. class seg2D: public point2D { … };
   6. class line2D: public point2D { … };
   7. class region2D: public line2D { … };
2. Constructors:
   1. point2D();
   2. point2D(int x, int y);
   3. line2D();
   4. line2D(point2D p1, point2D p2);
   5. region2D(line2D \* line2DArrays);
3. Methods:
   1. point2D:
      1. void setPoint2D(int x, int y);
      2. int getX();
      3. int getY();
   2. line2D:
      1. void setP1(point2D p1)
      2. void setP2(point2D p2);
      3. point2D getP1();
      4. point2D getP2();
   3. region2D:
      1. void setRegion2D(seg2D \* seg2DArrays);
      2. seg2D[] getRegion2D();
   4. operations: // suggestions
      1. bool equals(poi2D, poi2D);
      2. bool notEquals(poi2D, poi2D);
      3. bool on(poi2D, seg2D);
      4. bool poiInRegion(poi2D, region2D);
      5. bool segInRegion(seg2D,region2D);
      6. bool poiIntersect(poi2D,seg2D);
      7. bool segIntersect(seg2D,seg2D);
      8. poi2D poiIntersection(poi2D);

# Internal Data Structures and Operations:

enum topological\_vector\_types { poi\_shared, poi\_disjoint, poi\_on\_interior, poi\_on\_bound, bound\_poi\_disjoint, poi\_inside, poi\_outside, seg\_shared, interior\_poi\_shared, seg\_unshared, bound\_on\_interior, bound\_shared, bound\_disjoint, bound\_poi\_shared, p01, p10, p12, p21, p02, p20, p11};

enum combination\_std { point2Dpoint2D, point2Dline2d, point2Dregion2D, line2Dline2D, line2Dregion2D, region2Dregion2D};

static len\_topo\_vector\_types=21;

struct topological\_vector

{

bool VF [len\_topo\_vector\_types];

bool VG [len\_topo\_vector\_types];

int FC; // Type of combination\_std

};

//The following needs to be discussed with the professor

// The following is for optimization when predicate verification happens (Matrix Thinning)

\*\*\*int Matrix\_Thinning( int \*im[3][3], int n) // \* stands for pointer, n stands for number of topological predicates existing in the type combination

// This is a place holder for Minimum Cost Decision Tree when Predicate Determination happens (optimization)

// This is a placeholder for Determination of Dimension Refined Predicates

Interface:

topological\_vector explorePoint2DPoint2D(point2D, point2D);

topological\_vector explorePoint2DLine2D(point2D, line2D);

topological\_vector explorePoint2DRegion2D(point2D, region2D);

topological\_vector exploreLine2DLine2D(line2D, line2D);

topological\_vector exploreLine2DRegion2D(line2D, region2D);

topological\_vector exploreRegion2DRegion2D(region2D, region2D);

# Operation/method semantics description:

Exploration phase to find topological relationship between complex spatial objects identifies 6 algorithms for different combination of spatial objects.

1.Exploration Algorithm for point2D|point2D

Description: To explore the topological information between two point2D objects

Input: Two point2D objects and two topological feature vectors corresponding to two point objects and initialized to false

Output: Updated topological feature vectors

2.Exploration Algorithm for point2D|line2D

Description: To explore the topological information between point2D object and line2D object

Input: point2D object and line2D object and two topological feature vectors corresponding to the objects and initialized to false

Output: Updated topological feature vectors

3.Exploration Algorithm for point2D|region2D

Description: To explore the topological information between point2D object and region2D object

Input: point2D object and region2D object and two topological feature vectors corresponding to the objects and initialized to false

Output: Updated topological feature vectors

4.Exploration Algorithm for line2D|line2D

Description: To explore the topological information between two line2D objects

Input: Two line2D objects and two topological feature vectors corresponding to the objects and initialized to false

Output: Updated topological feature vectors

5.Exploration Algorithm for line2D|region2D

Description: To explore the topological information between line2D object and region2D

Input: line2D object and region2D object and two topological feature vectors corresponding to the objects and initialized to false

Output: Updated topological feature vectors

6.Exploration Algorithm for region2D|region2D

Description: To explore the topological information between two region2D objects

Input: Two region2D objects and two topological feature vectors corresponding to the objects and initialized to false

Output: Updated topological feature vectors

Evaluation phase takes the topological vector which is the end result of exploration phase to create 9 intersection matrix.

Input: Topological feature for each of the 6 combinations

Output: Usually a bool (implementation differs slightly depending on if it is a predicate verification query or if it is a predicate determination query)

The Steps of evaluation phase are as follows:

Overview: The 9 intersection matrix is created based on the topological feature vector which was given by the exploration phase [The size of the 9 intersection matrix may vary depending on the combination type which is used]

Step 1. The output differs slightly depending on whether it is a predicate verification query or it is a predicate determination query. The topological feature vector is an input.

1.1. If it is a predicate verification query, then we require the topological predicate p as an input too. We evaluate the logical expression on the right side, then we compare the boolean result with the boolean value stored at the respective position in p's intersection matrix. If they don't match, we return a false. Otherwise we continue to the next matrix predicate. We return true if it matches for all 9 matrix predicates. This requires constant time.

1.2. If it is a predicate determination query, we evaluate all the logical predicates on the right side. This yields a Boolean 9-intersection matrix. This Boolean 9-intersection matrix is compared to all 9 intersection matrices of the topological predicates of the particular type combination. The worst case will be the number of topological predicates present for that particular type combination.

Step 2. If we want to carry out optimization of the generation of topological requirements, the following methods can be implemented depending on the type of query which is posed:

2.1. Matrix thinning for Predicate Verification

Only a subset of 9 matrix predicates need to be verified in order to determine the validity of topological predicate between F & G. Based on this an algorithm can be implemented to thin out matrices so that only a few predicates are evaluated.

2.2. Minimum Cost Decision Tree for Predicate Determination

Method 1 - Use thinned out matrices

Method 2 - This focusses on reducing the number of tests by using a Minimum Cost Decision Tree. A full binary decision tree is constructed, whose inner nodes represent all matrix predicates, whose edges represent boolean values true or false and whose leaf nodes are the topological predicates. A DFS is used for search. Lower bound for search is O(logn) and upper bound is 8.

3. Evaluation of Dimension Refined Predicates

Dimension refined topological predicate are available only with line/line, line/region and region/region. If p(F,G) is false (from step 1) then pd(F,G) is false. If p(F,G) is true, other checks are required. Thus, first the 9IMC is done (to get p(F,G)), then the dimension which is needed can be selected, and then the dimension characterization can be done

# Requirements:

We require the types point2D, line2D, halfSegment2D and region2D object types be implemented.

We require the plane sweep algorithm should to be implemented.

Overlap numbers need to be implemented to find the relationship between region2d objects

Several events and functions in the plane sweep algorithm may need to be implemented

All spatial objects should belong to a parent Spatial Objects type.

All spatial objects should have a method boundary() that returns an appropriate representation of the object boundary

Spatial objects should have an operation String typeName() that returns the class name of the object.

halfSegment2D\* halfsegment( line2D OR region2D) returning an array of half segments  
halfSegment2D must have an attribute line2D segment component

point2D dominatingPoint(halfSegment2D) yielding the dominating point of f (in case it is not already implemented)

equality (=) and inequality (!=) of two single point2D objects

bool on(point2D, line2D) which determines whether a point is on a line segment

bool pointInRegion (...) which determines if a point lies within a region

bool segmentInRegion (...) which determines if a line segment lies within a region

bool pointIntersect (...) which determines whether two line segments meet at a point

bool segmentIntersect (...)

point2D pointIntersection (...) which returns the intersection point of two segments

pts(s) for segments (yields the infinite point set of s, further details are discussed in paper)