Pyspatiotemporalgeom: A Python Library for Spatiotemporal Types and Operations

Mark McKenney, Niharika Nyalakonda, Jarrod McEvers, Mitchell Shipton
Southern Illinois University Edwardsville
Department of Computer Science
Edwardsville, IL 62026
{marmcke,nnyalak,jmcever,mshipto}@siue.edu

ABSTRACT
The Pyspatiotemporalgeom library is a pure-python library implementing spatial data types, spatiotemporal data types for moving regions, and operations to create and analyze those types. The library is available on the Python Package Index (PyPI) and has been downloaded over 18,000 times since its release. In this paper, we demonstrate mechanisms to create random spatial data and perform operations over them. We then show how to create moving regions from existing data, and demonstrate aggregate operations over moving regions.

CCS Concepts
• Information systems → Geographic information systems; • Theory of computation → Data structures and algorithms for data management;

Keywords
Pyspatiotemporalgeom library, moving regions, component moving regions, aggregate operations.

1. INTRODUCTION
Spatiotemporal data refers to data with both a spatial and temporal component. One conception of spatiotemporal data, with regards to spatial database systems, is the idea of moving objects [4, 17]. A moving object is a spatial object that changes in position and/or extent over time. For example, a car may be represented as a moving point as it travels through a transportation network. A river that changes course over a geologic time scale may be represented as a moving line. A hurricane that changes extent and position as it develops can be represented as a moving region.

In this paper, we are concerned with moving regions. Moving regions are difficult to deal with in terms of implementation; this is evident in the few systems that support moving regions ([1, 2] is a good example of a system that does support moving object types). There are multiple reasons for the difficulty in implementing moving regions. First, implementing temporally static regions and their associated operations is no simple task. Second, data collection sources in the real world tend to collect data for moving regions as a series of temporally static snapshots that develop over time; thus, to create moving region data from real world sensors, we require algorithms to convert a series of snapshots to a single, temporally continuous, moving region. Third, once we have moving region data, we need algorithms to perform operations to analyze that data. The algorithms underlying moving region operations are difficult to implement, and the algebra over moving regions is not yet complete; particularly, aggregate operations over collections of moving regions are not yet fully explored [11, 7, 12, 6]. The Pyspatiotemporalgeom library, as described in this paper, provides mechanisms to deal with all of these difficulties.

The Pyspatiotemporalgeom (PSTGeo) library is a pure python library that implements data types and operations for regions and moving regions. The PSTGeo library has been downloaded over 18,000 times and is available in a packaged format as well as a git repository. The PSTGeo library is designed as a reference implementation for regions, for algorithms that define region operations, for moving regions as defined by the CMR model of moving regions [14, 13], for algorithms that define operations over moving regions, and for algorithms that define aggregate operations over moving regions. Because the PSTGeo library is designed as a reference implementation, it focuses on functionality and clarity over performance; this has lead to several fundamental design decisions regarding the library. First, the PSTGeo library is a pure python library that is available on the Python Package Index [9, 10]. Because it is a pure python library, it is portable, easy to install, and the source code is easily accessible for reading. Second, source code documentation is embedded with the source code and rendered in web page format with links to the actual source code [8]. Therefore, source code and documentation are tightly coupled and evolve together. Third, because the PSTGeo library is a reference implementation, it provides an ideal source upon which to test new operations, implement additional libraries, and it provides a useful tool for teaching spatial and spatiotemporal algorithms in courses. One goal of the library is to provide a source for fundamental algorithmic primitives that can be used in spatially oriented classes.

In this paper, we provide a brief demonstration of the capabilities of the PSTGeo library. We describe some of the capabilities of the PSTGeo library beginning with the
implementation of regions, moving into algorithms to construct moving regions from temporally static region snapshots, and finally discussing implementations of moving regions and spatiotemporal aggregate operations.

2. STRUCTURE OF THE PSTGeo LIBRARY

The PSTGeo library is available in the Python Package Index (PyPI) and its source is available through a git repository [10]. Using the pip tool, the library may be installed by simply executing the command in Figure 1.

The PSTGeo library is organized into three modules. The first is the Utilities Module, containing files that implement helper operations for the higher level objects. The utility files are named according to the category of data they operate on. For example, segLibrary.py includes operations over line segments, including line segment intersection algorithms, collinearity tests, and orientation tests. hsegLibrary.py contains operations over halfsegments, including operations to create halfsegment representations of regions, operations to identify cyclic structures in regions, and operations to label halfsegments to indicate the topology of the region they define. The convexHull.py file contains convex hull algorithms. The triangle.py library contains operations over triangles in 3D space.

The High Level Modules contain the region module, the interval region module, and the component moving region. The region module contains high level operations for regions, such as region intersection, difference, and union, as well as operations to construct regions from a collection of line segments and to create regions randomly. The interval region module provides mechanisms to create interval regions. An interval regions describes the motion of a region across a fixed time interval [5, 14]. The component moving region module contains data types and operations for interval regions under the CMR model.

Finally, the Example Modules provide working scripts that use the library in various ways in order to provide examples of proper use. The examples are documented, and can be called from external scripts.

3. SNIPPETS FOR THE CLASSROOM

Because the PSTGeo library is a reference implementation, many of the basic geometric utility functions that are often inlined in more complex functions are clearly defined and documented. This feature allows the PSTGeo library to be a useful teaching tool for practical geospatial programming courses. For example, the left-hand-turn-test (LHTT) is frequently used in geospatial computations. The LHTT takes three points in euclidean space and returns a value of true if when one walks from the first point to the second then turns to walk towards the third, one makes a left hand turn. This function is useful in determining line segment intersection and orientation. Figure 2 depicts the location and signature of the LHTT implementation within the PSTGeo package.

4. REGIONS

The main goal of the PSTGeo library is to support moving regions, so the region functionality is present to support the more complex types. A region in the PSTGeo library is defined as a list of halfsegments. Halfsegments are assumed to be labeled such that each segment indicates if the interior of the region lies above or below the halfsegment. This representation allows users to interact with regions rather easily.

The region module includes functionality to create a random region (i.e., a region induced in the plane by a set of randomly generated line segments), as well as to create a region from a list of existing line segments. Once regions are created, a user can compute the intersection, difference, or union of regions. The giveUniqueLabelToEachCycle function will assign each cycle in a region to a unique identifier so that a user can examine the structure of a region. Finally, mechanisms to create maps, as defined by spatial partitions [3], are provided.

For example, Figure 3 demonstrates the creation of random regions and the region intersection operation. The ability to create random regions of varying complexity makes the PSTGeo library ideal for generating data sets for research and educational purposes.

5. FROM REGIONS TO MOVING REGIONS

One concern in dealing with moving regions is that ob-

Figure 1: Installing Pyspatiotemporalgeom via the pip tool. Administrator privileges may be necessary.

```python
pip install pyspatiotemporalgeom
```

Figure 2: A code snippet demonstrating the location and signature of the isLeftTurn function that implements the left-hand-turn-test.

```python
import pyspatiotemporalgeom.region as region
r1 = region.getRandomRegion( 200 )
r2 = region.getRandomRegion( 200 )
print region.intersection( r1, r2 )
```

Figure 3: A code snippet demonstrating the creation of two regions using random region generation. The parameter indicates the number of randomly generated line segments that will be used to attempt to construct a region; higher numbers lead to more complex regions. The final line shows how one computes the intersection of two regions.

```
print region.intersection( r1, r2 )
```
import pyspatiotemporalgeom.REGION as INTERVALREGION
import pyspatiotemporalgeom.REGION as REGION
r1 = region.getRandomRegion( 50 )
r2 = region.getRandomRegion( 50 )
print INTERVALREGION.interpolateRegions( r1, r2, 10, 20 )

Figure 4: A code snippet showing the interpolateRegions function, which creates an interval region between a source region $R1$ at time 10 and destination region $R2$ at time 20.

Storing moving region data is difficult. The PSTGeo library provides an implementation of the algorithms defined in [16, 15] to automatically generate one possible path of motion between a source and destination region. The motion is represented as an interval region, which describes the motion of a region as it travels from a source configuration to a destination configuration. An interval region is represented simply as a collection of triangles in 3D space such that the first two dimensions are the spatial dimensions and the third is the time dimension. A triangle must have one edge form a boundary in the source or destination region, and the other edges then extend across the time interval (Figure 7 depicts an interval region). To create an interval region, a user simply uses the interpolateRegions function; the function takes two regions, and two times. The first region is considered the source configuration as it exists at the earlier time, and the second region is the destination configuration as it exists at the later time. Figure 4 demonstrates the usage of the interpolateRegions function.

6. MOVING REGIONS

Using the interpolateRegions function, one can create a moving region as a sequence of interval regions. Alternatively, the PSTGeo library offers the ability to create moving regions under the component model of moving regions (CMR). Representing an interval region as a collection of triangles in 3D space has the consequence that operations over interval regions are essentially operations over volumes in 3D space. Algorithms for such operations are difficult to implement, in practice, and are not widely available in existing systems. The CMR model defines an interval region to be a source configuration and end configuration, respectively represented as regions, a time indicating the instant the source region exists, and a time indicating the instant the destination region exists. The motion of the region across the time interval is not explicitly stored, but generated on the fly with the interpolateRegions function. Because interval regions are implemented in this way, we make the observation that the result of operations, such as the intersection of two interval regions, largely revolves around points in time when the topology of a line segment in one region changes in respect to the other region. In other words, the intersection of two interval regions may be computed by finding the points at which a line segment crosses from the exterior of the opposing region to its interior, and vice versa; we denote such times as times of interest. Once these points are known, the intersection operation can be computed using 2D operations at those time instants (see [11] for a full explanation).

To compute times of interest, we need to be able to compute the intersection of triangles in 3D space. The triangleLibrary.py provides such functionality. A triangle is defined as a 3-tuple of points, and a 3d line segment is a 2-tuple of points. Functions are included to return the intersection of a line segment and triangle, a vertical line segment and triangle, and the intersection points between two triangles.

To construct an interval region under the CMR model, a user must create a cIntervalRegion object, and then add the faces and holes of the source and destination regions. Because the faces and holes of the source and destination regions must be identified, we refer to these regions as structural regions. Once the structural regions are in place, the user can then compute operations, such as intersection, over the CMR interval regions. A moving region is then a collection of a time interval. In lines 13-18 the structural regions are added to the interval region. We assume a second interval region is constructed in the space implied in line 19. Line 20 shows how to compute the intersection of two interval regions.

Figure 5: Lines 3-18 show the construction of an interval region object under the CMR model. First, two structural regions are created, one for each end of a time interval. In lines 13-18 the structural regions are added to the interval region. We assume a second interval region is constructed in the space implied in line 19. Line 20 shows how to compute the intersection of two interval regions.

7. AGGREGATES OPERATIONS

The PSTGeo library currently offers functionality to compute temporal coverage operations over interval regions, which can then be composed to compute temporal coverage aggregates over moving regions in general.

A temporal coverage operation computes the time that a geometry is covered by an interval region. The aggregate version of the operations is expressed either as a min or a max operation. A min temporal coverage aggregate will return the spatial geometries covered for the smallest amount of time by an interval regions. The max temporal cover-
from pyspatiotemporalgeom.utilities import temporalAggregate
import pyspatiotemporalgeom.region as region
R1 = region.getRandomRegion(20)
R2 = region.getRandomRegion(20)
r=temporalAggregate.
createIntervalRegionAndComputeTemporalAggregate(R1,R2);

Figure 6: Computing a temporal coverage aggregate over two randomly generated regions.

Figure 7: An interval region the results of the max temporal coverage aggregate. The insert shows a top-down view. The unfilled dots show the end points of line segments covered by the interval region as it travels over the time interval for the maximum amount of time in the scene. The dashed polygon shows the region covered by the maximum amount of time (it is projected down to the (x,y) plane). The blue dot (in the middle) shows the point covered by the least, non-zero amount of time by the region.

age aggregate returns the spatial geometries covered for the longest amount of time by an interval region. The PSTGeo library currently computes the maximum temporal coverage of key points, line segments, and regions induced in the plane by an interval region. Functionality to is available to return the time every key point is covered, so additional aggregate operations, such as the min temporal coverage, are possible.

Figure 6 shows how to call the the temporal coverage aggregate operation in the PSTGeo library. Figure 7 depicts the result of the max temporal coverage aggregate for two randomly generated regions.

8. CONCLUSION

The Pyspatiotemporalgeom library is a versatile library for regions and moving regions. It is written to be easy to use, as illustrated by the various examples in this paper, and its source is designed to be easy to read. The library is useful in educational contexts since the various algorithms are accessible and extensible. For research, the library is freely available and open source, so it may be extended for new applications. For implementation purposes, the library provides complex algorithms that may be used to build spatiotemporal systems.

9. REFERENCES