Abstract
Large-scale spatio-temporal data is crucial for testing the spatio-temporal pattern mining algorithms. In this paper we present a data set generator that can generate moving spatio-temporal instances, with evolving regions. The characteristics of data generated can be configured with parameters given to program and those parameters allow users to test the performance of spatio-temporal mining algorithms in extreme conditions.

Keywords
Spatiotemporal — Data Mining — Pattern Mining

Introduction
Evolving region moving object random dataset generator creates a dataset of event instances which consist of number of polygons in different time instances. Dataset generator is mainly designed for the purposes of spatiotemporal co-occurrence pattern generation. Therefore the initial step of the program is to generate patterns and after that, generating instances for those patterns. The pattern generation and instance generation steps and heuristics are based on the work of Huang et al. [1]. In the following parts of this document, the detailed explanation about event, pattern and instance generation are provided.

1. Generation of Dataset
As mentioned earlier, this dataset generator is designed for spatiotemporal co-occurrence pattern mining purposes. Initial step of dataset generation is to create patterns. Spatial relationships are generated using these initially created patterns. Lastly, instances are created within this spatial relationship rectangles. This process can be summed up as follows:

- Initializing a spatial framework
- Generation of events
- Generation of patterns
- Generation of spatial neighborhood relationships (clamps)
- Generation of instances

1.1 Spatial framework and event generation
Spatial framework is defined as a square in this program. The upper-left corner of this square is located at point \((0, 0)\) and edges of this framework is of size \(D\). Therefore, the framework can be defined with following four points: \((0,0), (0,D), (D,D), (D,0)\). All the instances and spatial neighborhood relationships will be created inside this framework.

After creating the spatial framework, program creates events. Events are defined with following features:

- Name of the event
- Size of the event (area of starting polygon of event’s instances)
- Area change of the event
- Duration of the event (life time of event instances)
- Speed of the event (how quick is the movement of event’s instances)

Number of events are denoted with \(E\) and program gets an average number of events as parameter and uses it as the mean of Poisson distribution. The event names are formatted as \(e_i\), where \(0 < i < (E + 1)\) and \(i\) is an integer. Size, duration and speed of an event are chosen from uniform distribution and the values for them are integers that are varying from 1 to 5. For size, 1 is very small and 5 is very large; for duration,
1 is very short and 5 is very long and for speed 1 is very slow and 5 is very fast. Area change variable is also taken from a uniform random distribution but it varies from $-2$ to $2$. For area change, negative values represent the diminution and positive values represent expansion in the area. If area change of an event is 0, then the instances of this particular event is not changing area. The values of these will further be explained in instance generation section.

1.2 Generation of patterns

Pattern generation can be examined in two parts. First part is the generation of core patterns and the second part is the generation of maximal co-occurrence patterns.

Core patterns are the fundamental patterns for a generated dataset, where we expect to see more of them when pattern mining algorithms are applied to dataset. The number of core patterns to be generated are given as a parameter to program. A core pattern can be at least of size 2 and at most of size $E - 2$. A pattern can be defined as a set of events. Events in each pattern are selected from complete event set generated earlier. The events in each pattern is taken randomly (from a uniform distribution) without considering the features (size, speed, duration or area change) of events. Therefore, a core pattern is a set of randomly selected events where the size of the set is defined by a parameter. Given a parameter (maximum size of core pattern - $S_{core}$) program creates one core pattern for each size, $k$, where $k$ is between 2 and $S_{core}$ and $k$ is an integer. Note that the number of core patterns generated is $S_{core} - 1$.

The second phase of pattern generation is carried out using core patterns, and in this phase, program creates maximal co-occurrence patterns. The number of maximal co-occurrence patterns, $m$, are obtained using a Poisson distribution (with a given parameter $N_{maxima}$ as the mean of this Poisson distribution). Then, the program creates $m$ maximal co-occurrence patterns from $S_{core} - 1$ core patterns. A size $- (k + 1)$ maximal co-occurrence pattern is generated from size $- k$ core pattern. The events in size $- k$ core pattern is copied to maximal co-occurrence pattern’s event set and one more event is chosen randomly from a uniform distribution. Program also guarantees, not generating the same pattern, however, obviously some patterns can be subset or superset of another pattern.

Important note: The user of the program must be sensible for the number of maximal pattern generation in the pattern generation process. The calculation of combination operation for big numbers is quite inefficient and superfluous. The maximum number of maximal co-occurrence patterns to be generated depends on number of events and parameter $m$ (number of maximal patterns to be generated). For instance, if number of events, $E$, is 10, the event list of pattern $P$ is $\{e_1, e_2, e_3, e_4, e_5\}$ and we want to generate six $size - 6$ maximal co-occurrence patterns. It is possible to generate five maximal co-occurrence patterns but there is no way the sixth pattern will be generated.

1.3 Generation of spatial neighborhood relationships and instances

After patterns are generated, next is to generate instances that will populate the patterns. Number of instances to be generated for a pattern is determined using a parameter to program. This parameter is used as the mean of a Poisson random distribution. The value $S_{instance}$ denotes the number of instances to be generated for each pattern.

Instance generation starts with creating spatial neighborhood relationships. A clamp is a spatial neighborhood relationship that is defined over a rectangle inside the spatial framework. A clamp contains the instances of events in a pattern. Number of instances in a clamp is determined with a parameter given to program. This parameter is used as the mean of a Poisson random distribution and the number of instances of each event in a pattern is called clamp size, and denoted by $M_{clamp}$. A clamp rectangle is a random rectangle inside the spatial framework and the width/height ratio of this rectangle is given as a parameter (denoted by $C_{X/Y-ratio}$) and the ratio between clamp rectangle area and spatial framework area is also a parameter given to the program (denoted by $C_{area-ratio}$). The location (x and y coordinates of upper-left corner) of that rectangle is randomly chosen from a uniform distribution and it is ensured by the program that this rectangle is always inside the spatial framework rectangle.

1.3.1 Pattern instance generation

The instance objects in this program has following attributes:

- Instance id
- Related event object
- Start and end time of instance
- Speed of instance
- List of polygon objects

The class constructor of ‘Instance’ objects gets event type, starting time, duration, starting area, speed, number of points of polygons, ratio of area change between starting and ending area and the clamp rectangle containing the polygon geometries of instance created.

The speed, duration, size and area change of an instance determined before the instantiation. As mentioned in event generation section, events have features (namely duration, size, speed, area change). Area change is the ratio between ending polygon’s area and starting polygon’s area. Using event features and base speed, base area and base duration parameters, mentioned attributes of instances are determined. Base speed, base area and base duration parameters are provided by the user of the program. Base speed attribute is denoted by $Speed_{base}$ and it represents the number of pixels that a polygon moves in unit time interval. Base duration attribute is denoted by $Duration_{base}$. Base area attribute is $Area_{base}$ and it represents the number of pixels that a polygon covers. These values are subject to change according to the needs of user and prone to affect the prevalence of patterns strongly. It is highly recommended that those values should be changed according to the requirements of users and users
must be sensible enough for capabilities and limitations of instance generation process. For instance, suppose we have a clamp rectangle whose area is 20,000 pixels. Base area obviously cannot be more than 20,000. Another example can be width requirements: let clamp rectangle’s width be 1000. (DurationxSpeed) will give us replacement and it must not be more than 1000.

Following example can help readers better understand the calculation of instance attributes: To find the speed of an instance, using the speed parameter of event and base speed, we get an interval. Let instance \( I_j \) be an instance whose event type is \( E_k \) and let speed level of \( E_k \) instances be 4. If base speed parameter is 20, then the average speed for \( E_k \) instances is \( 4 \times \text{Base speed} \) that is 80. The speed values of \( E_k \) instances vary from 70 to 90 (randomly chosen from a uniform distribution). For each speed level instances can get 20 (Base speed) different values and the average speed value for speed level-4 is \( (4 \times 20 = 80) \). Same process is applied while finding the duration and area. General formula for an attribute value \( A \) of instance \( I_j \) is following:

\[
A_j = (\text{Base}_A \times R) + ((\text{Base}_A - 0.5) \times A_{E_k})
\]

where \( \text{Base}_A \) is base value of attribute, \( R \) is a random number in range \([0, 1)\) and \( A_{E_k} \) is the level of that property (obtained from event). Finding area change of an instance is a bit more tricky. Area change level of an event can be between \(-2\) to \(2\). \(-2\) and \(2\) represents big diminutions and expansions respectively. \(-1\) and \(1\) represents relatively smaller diminution and expansions. Figure 1 shows the distribution of area change values. When the area change level is \(-2\), area change attribute can get values between 0.25 to 0.5; if level is \(-1\), then attribute can get values ranging from 0.5 to 1. Level-0 means no change, so the instance’s area change attribute will be 1. For level-1, area change is between 1 and 2 and lastly for level-2 the area expansion is between 2 and 4.

Lastly, number of points in the polygon is also uniformly distributed between 4 to 10.

**Polygon generation:** When a polygon first instantiated, the parameters clamp rectangle, area and number of points are given. Using that information, we randomly create a starting polygon as follows:

1. A unit polygon in 2 by 2 rectangle is generated using number of points parameter
   (a) Unit rectangle is a square, whose upper-left corner is located at point \((0, 0)\) and width and height is both 2. The center of square is located at point \((1, 1)\). Given number of points, \(N_P\) program artificially divides the unit square into triangles. Each of those triangles share point \((1, 1)\) as a vertex and they have same angle at that vertex. Then, program chooses a random point inside each of those triangles (by selecting a random angle and the random vector length – see Figure 2 ) and all those points create a random polygon.

2. The area of unit polygon is calculated (let current area be \(\text{Area}_{curr}\)), and polygon is enlarged by multiplying point coordinates with \(\frac{\text{Area}_{desired}}{\text{Area}_{current}}\) ratio, where \(\text{Area}_{desired}\) is the desired area that is given as a parameter when initializing the instance object.

3. After the enlargement, we locate the polygon inside the clamp rectangle. For the simplicity of relocation, we divided the clamp rectangle to 16 grids \((4 \times 4)\). Also, the instances move from left to right. The starting location of instances are selected randomly from four left most grids in clamp rectangles.

This process is followed by creating an ending polygon. The process is as follows:

1. Firstly, the bounding box of starting polygon is obtained

   Figure 1. Distribution of area change values

   Figure 2. Unit polygon generation with four points
and it is enlarged by area change parameter. Let this rectangle be $R_{end}$.

2. We find the displacement by multiplying speed and duration of object. For simplicity, left upper corners of starting and ending polygons are taken as reference points. Again, with a random angle that is varying between $-\pi/4$ and $\pi/4$ new random displacement vector is created (note that the size of displacement vector is $(Speed \times Duration)$).

3. Then, a random polygon with $N_P$ points with desired ending area $(AreaChange \times StartingArea)$ created inside rectangle $R_{end}$. Note that polygon generation process is same with starting polygon generation. Later, we move this rectangle with randomly generated displacement vector.

Last part of the polygon generation process is to generate the geometries between starting and ending polygons. To do that, each point in starting polygon is linearly projected to points in ending polygon. Note that they have same number of points and the polygon generation algorithm ensures that none of those newly created polygons can be self-intersecting.

Noise instance generation: Noise instance generation is quite similar with pattern instance generation. To determine the number of noise instances a parameter called $L_{Noise}$ is used. $L_{Noise}$ is the ratio of noise instances over total number of pattern instances. Let $|P|$ be the size of pattern (number of events in a pattern), for each clamp we generate $(M_{clamp} \times |P|)$ instances that are related with pattern. For noise instances we create $L_{Noise} \times (M_{clamp} \times |P|)$ instances. Noise instances are generated outside the related clamp rectangle. To generate a noise instance, a random rectangle (similar to clamp rectangle) that is not intersecting with clamp rectangle is created firstly. Then only one instance is created inside this rectangle. The event of the noise instance is selected randomly from a uniform distribution. The events that are to be chosen are not inside the event set of clamp’s pattern.

2. Usage of data set generator

Evolving region moving object data set generator program is programmed in Java environment. It can be downloaded from our website [1]. The compressed zip file includes an executable jar file and a properties file. The settings described in Section 1 can be found in this document.

2.1 Execution of program

Program can be executed either with double-clicking the jar file or using 'java -jar ermodg.jar' command from command line interface. For larger dataset files, program may need more memory. '-Xmx' flag is used for increasing the heap size in Java. To increase the heap size in Java, users of this program can use 'java -Xmx4G -jar ermodg.jar' command to increase the heap size to 4 GB.

2.2 Parameters of program

Parameters of this program and the explanation of these parameters can be seen in Table 1. Note that parameters $AvgNoOfEvents$, $MaxCorePatternSize$, $AverageInstanceSize$, $AvgClampSize$ are mean values of respective Poisson distributions ($\lambda_1$, $\lambda_2$, $\lambda_3$, $\lambda_4$). Their values are determined in the execution time of the program.

2.3 Output of program

Program firstly outputs a data configuration file and instance files classified for each event type. Data configuration file
Table 1. Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpatialFrameworkSize</td>
<td>Integer</td>
<td>Size of the framework (-D)</td>
</tr>
<tr>
<td>AvgNoOfEvents</td>
<td>Integer</td>
<td>Average number of events (-\lambda_1)</td>
</tr>
<tr>
<td>MaxCorePatternSize</td>
<td>Integer</td>
<td>Average number of maximal co-occurrence patterns (-\lambda_2)</td>
</tr>
<tr>
<td>AvgInstanceSize</td>
<td>Integer</td>
<td>Average number of instances to be generated for each pattern (-\lambda_3)</td>
</tr>
<tr>
<td>AvgClampSize</td>
<td>Integer</td>
<td>Average number of instances in each clamp (-\lambda_4)</td>
</tr>
<tr>
<td>NoiseRatio</td>
<td>Float</td>
<td>Ratio of noise instances to pattern instances (-L_{noise})</td>
</tr>
<tr>
<td>SamplingInterval</td>
<td>Integer</td>
<td>Time interval that instances to be output</td>
</tr>
<tr>
<td>BaseSpeed</td>
<td>Integer</td>
<td>Value of base speed (-BaseSpeed)</td>
</tr>
<tr>
<td>BaseArea</td>
<td>Integer</td>
<td>Value of base area (-BaseArea)</td>
</tr>
<tr>
<td>BaseDuration</td>
<td>Integer</td>
<td>Value of base duration (-BaseDuration)</td>
</tr>
<tr>
<td>ClampXYRatio</td>
<td>Float</td>
<td>Width/Height ratio for clamp rectangles</td>
</tr>
<tr>
<td>ClampAreaRatio</td>
<td>Float</td>
<td>Ratio between clamp rectangle area and spatial framework area</td>
</tr>
</tbody>
</table>

demonstrates the list of events, the features of events, the generated patterns, and an informative table consisting of event, instance and pattern counts. Since events and patterns are randomly generated the data configuration file is subject to change in each run of the program. Program creates a file for each event type, in this file the instances of this event type throughout all time slots are present. In each line of instance files, the polygon representing an instance at time \(t\) is shown. Hence, a line is formatted as follows:

Instance ID - Time Slot(\(t\)) - POLYGON(\(x_1, y_1, x_2, y_2, \ldots\))

All the instances, in each time interval they are present, are output using parameter SamplingInterval. For instance, if SamplingInterval is 5, then instances are output in each 5 time unit period.

References