
SPATIAL DATA GENERATORS

Tin Vu

Computer Science and Engineering Department
University of California, Riverside
tin.vu@email.ucr.edu

Sara Migliorini

Computer Science Department
University of Verona, Italy
sara.migliorini@univr.it

Ahmed Eldawy

Computer Science and Engineering Department
University of California, Riverside
eldawy@ucr.edu

Alberto Belussi

Computer Science Department
University of Verona, Italy
alberto.belussi@univr.it

ABSTRACT

This gem describes a standard method for generating synthetic spatial data that can be used in benchmarking and scalability tests. The goal is to improve the reproducibility and increase the trust in experiments on synthetic data by using standard widely acceptable dataset distributions. In addition, this article describes how to assign a unique identifier to each synthetic dataset that can be shared in papers for reproducibility of results. Finally, this gem provides a supplementary material that gives a reference implementation for all the provided distributions.

Keywords benchmark, synthetic data, generator

1 Introduction

In many published papers, researchers often need to test their implementations of new index structures or query execution methods on large scale spatial data. While some real datasets exist, the research community also needs to try datasets with specific characteristics to highlight how the proposed research behaves under certain circumstances. Synthetic data generation gives researchers full control over the data characteristics such as data skewness, complexity of geometries, or amount of overlap between datasets.

This article proposes a practical tool for generating synthetic spatial datasets with various skewed distributions. These generators have been successfully used in existing research to evaluate index construction, query processing, spatial partitioning, and cost model verification. While the generators are already used in many papers, there is a fact that the researchers rarely describe the details of generating these datasets for two reasons. First, it is not usually a research contribution and the authors do not want to draw attention to it. Second, it takes a precious space of the paper that authors usually prefer to utilize for other parts.

This gem takes the burden of describing, in detail, how to generate synthetic data of six common distributions. As the gems are designed to be flexible, it fits very well the generation of synthetic data where other researchers can add more datasets in the future.

Figure 1 gives an overview of the main parts of the proposed generator. First, the dataset descriptor is a vector that contains information about the dataset to be generated. It acts a unique identifier for the synthetic dataset and it consists of three parts, (1) the distribution ID $i \in [1, 6]$ for six implemented distributions, (2) the model parameters depending on the chosen distribution, and (3) a transformation matrix used later by the transformer. The first two components of the dataset descriptor, i.e., distribution ID and model parameters, are passed to the generator which generates the desired dataset. After that, the transformer applies an affine transformation on the generated data according to the third component of the dataset descriptor.

Since the dataset descriptor fully identifies the generated dataset, researchers who use these generators can simply cite this gem and list the descriptors of the datasets they used. Other researchers can then regenerate the same datasets

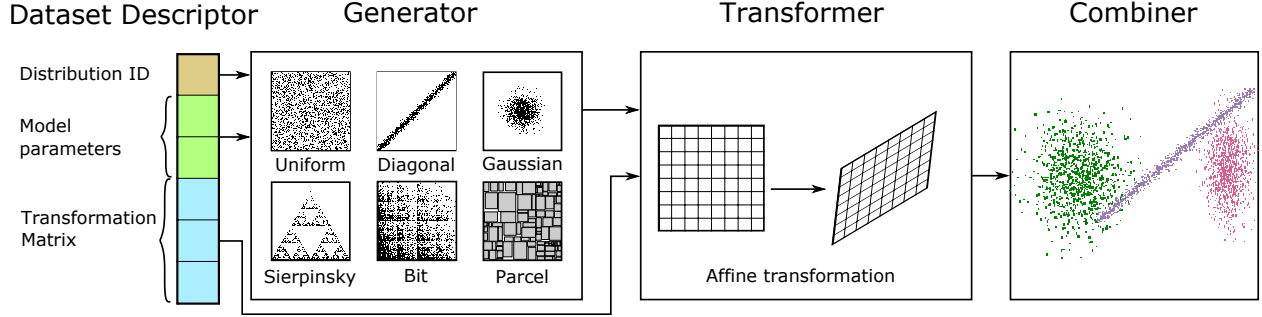


Figure 1: Overview of the spatial generator

following the same procedure described in this gem. For guidance, it provides a reference implementation [1] to all the generators as a supplementary material, but researchers can develop other generators that follow the same guidelines, e.g., a Spark-based generator for big spatial data. For example, Table 1 at the end of this gem defines the six datasets used in this article.

The final component, combiner, can be used to create compound datasets by simply merging two or more simple datasets. In this case, the descriptor of the compound dataset is simply the concatenation of all the descriptors of the simple datasets.

As shown in Figure 1, this gem uses six different distributions for generating the simple datasets, which are all described in the next section.

2 Data Generators

This section describes the six synthetic generators which are used in this gem, namely, uniform, diagonal, Gaussian, Sierpinski, bit, and parcel. Some of these generators are inspired by a benchmark developed by Beckmann and Seeger [2]. This article provides more details about these generators and defines some additional generators. Each generator G_* takes a list of common parameters $[cp_1, cp_2, \dots]$ and a list of distribution-specific parameters $[sp_1, sp_2, \dots]$. For the family of generators that are considered in this gem, there are two common parameters, the DATASET CARDINALITY (*card*) specifying the total number of geometries and the NUMBER OF DIMENSIONS (d). The generator here considers the generation of two-dimensional geometries; the extension to multi-dimensional datasets is straightforward.

For all these generators, the REFERENCE SPACE that contains the generated data is $[0, 1]^d$, where d is the number of dimensions. Additionally, these generators assume the existence of a random number generator $\text{RND}()$ which generates random numbers in the range $[0, 1)$. This generator can be used to generate random numbers for three popular distributions, Bernoulli, Uniform, and Normal, as follows ¹.

$$\text{Bernoulli}(p) = \begin{cases} 1 & ; \text{RND}() < p \\ 0 & ; \text{otherwise} \end{cases} \quad (1)$$

$$U(a, b) = (b - a)\text{RND}() + a \quad (2)$$

$$N(\mu, \sigma) = \mu + \sigma \sqrt{-2 \ln \text{RND}_1()} \cdot \sin(2\pi \text{RND}_2()) \quad (\text{Box and Muller [3]}) \quad (3)$$

Algorithm 1 shows the general schemata of the first five generators, namely, uniform, diagonal, Gaussian, Sierpinski, and bit distribution. These five generators can generate both points and rectangles. Points are generated using the GenPNT_* methods, described shortly, while rectangles are generated by using these points as centers. Lines 1,2 initialize the result set \mathcal{G} to the empty set and the random number generator. If desired, the seed of the random number generator can be fixed to generate exactly the same random dataset. The loop in lines 3-11 generates one record at a time. Line 4 calls a generic GENPNT_* function that is different for each generator. This function returns a random point (x, y) according to the desired distribution. Then, Line 5 tests if the point is inside the reference space $[0, 1]^d$ as some generators can generate points outside that space, e.g., Gaussian. If the point is inside the reference space, the

¹More efficient implementations are usually available in standard packages

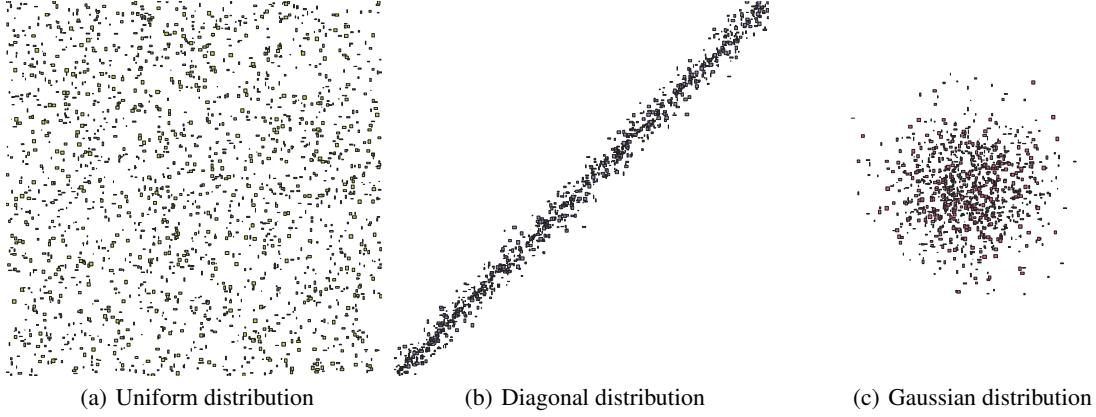


Figure 2: Examples of the first three distributions

algorithm continues by generating random width and height for the rectangle by using the parameters sp_1 and sp_2 as the maximum allowed width and height. Finally, a rectangle is generated with (x, y) as the center and (w, h) as its dimensions, i.e., the corner point is at $(x - w/2, y - h/2)$. The notation $Box(x, y, w, h)$ is used to indicate a box with its lower corner point at (x, y) and has dimensions of w and h .

The following parts are the descriptions of the point generators ($GENPNT_*$) for the first five distributions and then the parcel distribution which generates rectangles directly without generating points first.

Algorithm 1: $G_*(\cdot)$: basic algorithm for the first five generators.

Input: $card, d = 2, sp_1, \dots, sp_n$

Result: Set of geometries: $\mathcal{G} = \{geom\}$

```

1  $\mathcal{G} \leftarrow \emptyset; i \leftarrow 0;$ 
2 Initialize the random number generator;
3 while  $i < card$  do
4    $(x, y) \leftarrow GENPNT_*(i, sp_3, \dots, sp_n);$ 
5   if  $(x, y) \in [0, 1]^d$  then
6      $w = U(0, sp_1);$ 
7      $h = U(0, sp_2);$ 
8      $\mathcal{G} = \mathcal{G} \cup \{Box(x - w/2, y - h/2, w, h)\};$ 
9      $i \leftarrow i + 1;$ 
10  end
11 end
12 return  $\mathcal{G}$ 

```

2.1 Uniform

In the uniform distribution, points are generated randomly inside the reference space $[0, 1]^d$ as shown in Figure 2(a). This distribution models non-skewed data such as data in suburban areas. No additional specific parameters are needed for this generator. The point (x, y) is generated using the following equations.

$$GenPNT_{uni}() = (x = U(0, 1), y = U(0, 1)) \quad (4)$$

2.2 Diagonal

The diagonal point generator generates points that are concentrated around the diagonal line $x = y$ as illustrated in Figure 2(b). This distribution models real data concentrated around a line such as a river bank or a highway. The affine transformation, described later, can be used to arbitrarily rotate this line. This generator takes two additional parameters $perc$ and buf , where $perc \in [0, 1]$ is the percentage (ratio) of the points that are exactly on the line, and $buf \in [0, 1]$ is

the size of the buffer around the line where additional points are scattered. The additional points are scattered according to a normal distribution. Algorithm 2 illustrates the generation of a point using the diagonal distribution. Line 1 decides with a probability $perc$ to generate a point exactly on the diagonal. Otherwise, with probability $1 - perc$, it generates a point that is shifted with a distance d from the center. The distance is generated from the normal distribution $N(0, buf/5)$ which has almost a 99% probability in generating a number in the range $[-buf, +buf]$. This distance is then divided by $\sqrt{2}$ to calculate the orthogonal offset for x and y . Finally, the point location is generated.

Algorithm 2: GENPNT_{dia} ($perc, buf$)

```

1 if Bernoulli( $perc$ ) = 1 then
2   |  $x = y = U(0, 1)$ ;
3 else
4   |  $c = U(0, 1)$ ;
5   |  $d = N(0, buf/5)$ ;
6   |  $x = c + d/\sqrt{2}$ ;
7   |  $y = c - d/\sqrt{2}$ ;
8 end
9 return ( $x, y$ );

```

2.3 Gaussian

In the Gaussian distribution, the points are concentrated around the center of the input space $(0.5, 0.5)$ as illustrated in Figure 2(c). This distribution can model real data concentrated around a point such as a metro area. The coordinates follow a normal distribution $x, y \sim N(0.5, 0.1)$. This ensures that almost 99% of the points fall in the box $[0, 1]^d$. If points are generated outside that space, the loop in Algorithm 1 will drop the point and generate another one. This might make the data distribution is a bit different from original Gaussian distribution. In summary, the Gaussian point generator follows the following formula.

$$\text{GENPNT}_{\text{Gaussian}} = (x = N(0.5, 0.1), y = N(0.5, 0.1)) \quad (5)$$

2.4 Sierpinski triangle

In this case, the skewed distribution is obtained by applying a rule for generating points that belong to a fractal (the Sierpinski's triangle) [4]. Figure 3(a) gives an example of this data. This pattern could be found in many real model such as cellular automata or motors. This rule is based on an iterative approach such that the generation of the next point of the set depends on the current point and a random function. The function GENPNT_{sie}(pnt, i) has two specific parameters: the previous point of the iteration pnt and the iteration variable i . It generates a two-dimensional point at each iteration as follows:

$$\text{GENPNT}_{\text{sie}}(pnt, i) = \begin{cases} (0.0, 0.0) & \text{if } i = 0 \\ (1.0, 0.0) & \text{if } i = 1 \\ (1/2, \sqrt{3}/2) & \text{if } i = 2 \\ \text{MIDDLEPOINT}(pnt, (0.0, 0.0)) & \text{if } i > 2 \wedge \text{DICE}(5) \in \{1, 2\} \\ \text{MIDDLEPOINT}(pnt, (1.0, 0.0)) & \text{if } i > 2 \wedge \text{DICE}(5) \in \{3, 4\} \\ \text{MIDDLEPOINT}(pnt, (1/2, \sqrt{3}/2)) & \text{if } i > 2 \wedge \text{DICE}(5) = 5 \end{cases}$$

where $\text{DICE}(5) = \lfloor U(0, 5) \rfloor + 1$ is a random function producing a number between 1 and 5 and $\text{MIDDLEPOINT}(pnt_1, pnt_2)$ computes the middle point between two points, pnt_1 and pnt_2 . Notice that, the first three points are the corners of the triangle; the successive points are generated starting from the current point pnt and computing the middle point between pnt and one of the vertices of the triangle chosen according to the random function $\text{DICE}(5)$. The vertices of the base are chosen with a probability of $2/5$, the other vertex with a probability of $1/5$. Figure 3(a) shows an example of the resulting set.

2.5 Bit distribution

Another approach for generating a skewed point dataset is to introduce a rule for generating the coordinates of the points by assigning higher probability to a subset of coordinates. For instance in the Bit distribution, the point coordinates are generated as a bit string of a fixed length where each bit is set with a fixed probability $p \in [0, 1]$. This generator takes

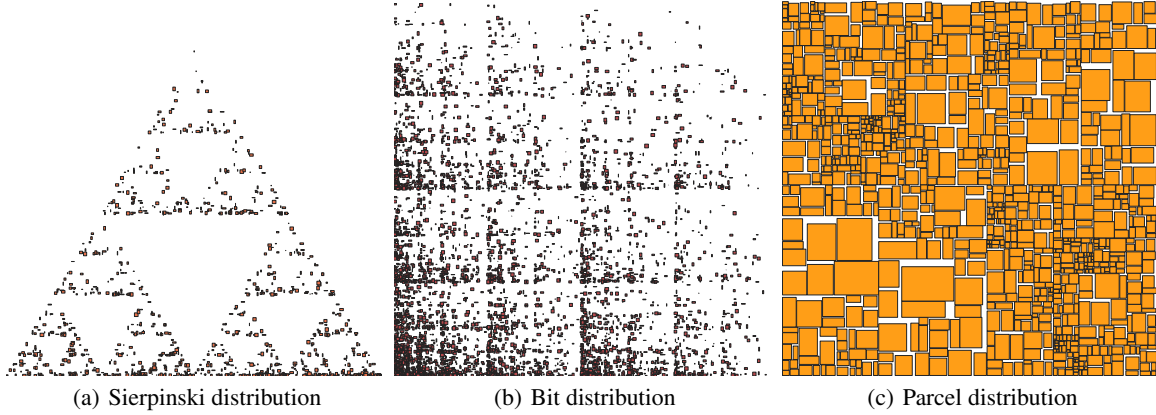


Figure 3: Examples of the last three distributions

two parameters, p and $digits$, where p represents a fixed probability of setting each bit independently to 1 and $digits$ represents the number of binary digits after the fraction point. It generates a point in a higher dimensional space by setting each dimension independently in the same method as shown below:

$$\text{GENPNT}_{bit}(p, digits) = (\text{BIT}(p, digits), \text{BIT}(p, digits))$$

where $\text{BIT}(p, digits)$ generates a real number between 0.0 and 1.0 as shown in the following Algorithm 3,

Algorithm 3: $\text{BIT}(p, digits)$

Input: $p, digits$

Result: Real number: n

```

1  $n \leftarrow 0.0; i \leftarrow 0;$ 
2 for  $i = 1$  to  $digits$  do
3    $c \leftarrow \text{Bernoulli}(p);$ 
4    $n \leftarrow n + c/2^i;$ 
5 end
6 return  $n$ 

```

2.6 Parcel distribution

This generator directly generates rectangles according to the parcel distribution. The parcel distribution generates geometries that represent boxes of different sizes as illustrated in Figure 3(c). This distribution can model land sections delineated in urban areas.

In addition to the cardinality $card$ of the generated dataset, the parcel generator takes two specific parameters r and d , where:

- $r \in [0, 0.5]$ is the minimum tiling range for splitting a box. $r = 0$ indicates that all the ranges are allowed while $r = 0.5$ indicates that a box is always split into half.
- $d \in [0, 1]$ is the dithering parameter that adds some random noise to the generated rectangles. $d = 0$ indicates no dithering and $d = 1.0$ indicates maximum dithering that can shrink rectangles down to a single point.

Algorithm 4 describes how the parcel generator works. The first loop in lines 3-15 repetitively splits the reference space $[0, 1]^d$ along one of the two axes x and y . It always splits along the longer axis of the given box. This loop runs $card - 1$ times as it generates one new box at each iteration.

The second loop in lines 16-19 adds the dithering effect by shrinking each box with a ratio $1 - U(0, d)$. This means that if $d = 0$, the ratio will always be equal to 1.0 which means no shrinking. If $d = 1.0$, the ratio can reach up-to 1.0 which shrinks the boxes all the way to a single point.

Algorithm 4: $G_{parcel}(r, d)$: Generated boxes of the parcel distribution

Input: $card, d = 2, r, d$

Result: Set of geometries: $\mathcal{G} = \{geom\}$

```

1 Initialize the random number generator;
2  $\mathcal{G} \leftarrow \{Box(0, 0, 1.0, 1.0)\}$ ;
3 while  $|\mathcal{G}| < card$  do
4    $b \leftarrow \mathcal{G}.dequeue$ ;
5   if  $b.width > b.height$  then
6      $splitSize = b.width * U(r, 1 - r)$ ;
7      $b_1 = Box(b.x, b.y, splitSize, b.height)$ ;
8      $b_2 = Box(b.x + splitSize, b.y, b.width - splitSize, b.height)$ ;
9   else
10     $splitSize = b.height * U(r, 1 - r)$ ;
11     $b_1 = Box(b.x, b.y, b.width, splitSize)$ ;
12     $b_2 = Box(b.x, b.y + splitSize, b.width, b.height - splitSize)$ ;
13  end
14   $\mathcal{G}.enqueue(b_1)$ ;  $\mathcal{G}.enqueue(b_2)$ ;
15 end
16 for  $b \in \mathcal{G}$  do
17    $b.width = b.width \cdot (1 - U(0, d))$ ;
18    $b.height = b.height \cdot (1 - U(0, d))$ ;
19 end
20 return  $\mathcal{G}$ 

```

3 Post Transformations

This section describes two methods that can give further flexibility on customizing the generated data, transformation and compounding. The transformation method applies a simple affine transformation on the generated geometries. The compounding method combines several datasets by simply unifying all their geometries.

3.1 Affine Transformation

The generators describes above are designed to be very simple on purpose. Instead of complicating each generator, most of the customization is moved to this step. This step simply applies a standard affine transformation to all the generated geometries. In the case of points, the transformation is applied to the coordinates of the point. For rectangles, the transformation is applied to the two opposite corners, i.e., the lower left and upper right corners. This ensures that the rectangle remains orthogonal even after rotation which is usually desired in data structures and algorithms that deal with bounding boxes.

An affine transformation is defined by a fixed-size matrix. For two-dimensional data, the affine transformation transforms a point (x, y) to a transformed point (x', y') according to the following equation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6)$$

where $a_1 \dots a_6$ are the parameters of the affine transformation. This formula could be modified to work with higher dimensional data.

3.2 Compound Datasets

The final stage is to combine several datasets of either the same distribution but different parameters, different distributions, or with different transformation matrices. Show some examples of how combining datasets can generate new interesting datasets.

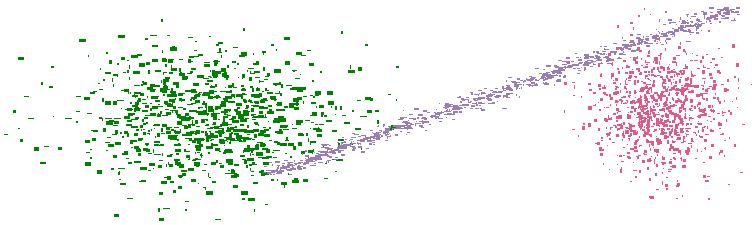


Figure 4: Example of compound dataset obtained by combining two different Gaussian distributions and one diagonal distribution.

Distribution ID	$card$	d	sp_1	sp_2	sp_3	sp_4	a_1	a_2	a_3	a_4	a_5	a_6
Uniform (Figure 2(a))	1000	2	0.02	0.02			1	0	0	0	1	0
Diagonal (Figure 2(b))	1000	2	0.01	0.01	0.2	0.1	1	0	0	0	1	0
Gaussian (Figure 2(c))	2000	2	0.1	0.1			1	0	0	0	1	0
Sierpinski (Figure 3(a))	1000	2	0.01	0.01			1	0	0	0	1	0
Bit (Figure 3(b))	5000	2	0.01	0.01	0.3	10	1	0	0	0	1	0
Parcel (Figure 3(c))	1000	2	0.2	0.2			1	0	0	0	1	0

Table 1: Identifiers for the six sample datasets shown in this paper. For simplicity, all of them use the identity transformation (affine) matrix

3.3 Identifying Datasets

Based on the proposed method, each generated simple datasets, i.e., not compound, can be identified using a fixed vector. This vector is the one illustrated in Figure 1 and it contains the generator model G_* , the common parameters $card$ and d , the specific parameters $sp_1 \cdots sp_n$, and the affine transformation matrix parameters $a_1 \cdots a_6$. Researchers who use the generators described in this paper can simply list all these parameters in a table to allow other researchers to generate datasets of the same characteristics. For example, Table 1 identifies the six sample datasets illustrated in Figure 2 and 3.

Acknowledgment

This work is supported in part by the National Science Foundation (NSF) under grants IIS-1838222 and CNS-1924694 and by the USDA National Institute of Food and Agriculture, AFRI award number A1521.

References

- [1] Spatial Data Generators. <https://github.com/tinvukhac/spatialdatagenerators>.
- [2] Norbert Beckmann and Bernhard Seeger. A Benchmark for Multidimensional Index Structures. 2019. <https://www.mathematik.uni-marburg.de/~rstar/benchmark/distributions.pdf>.
- [3] G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29(2):610–611, 06 1958.
- [4] Michael F Barnsley and Alan D Sloan. A better way to compress images. *Byte*, 13(1):215–223, 1988.