algorithms

Article

An Efficient GNSS Coordinate Recognition Algorithm for Epidemic Management

Jong-Shin Chen [1,*], Chun-Ming Kuo [1] and Ruo-Wei Hung [2,*]

[1] Department of Information and Communication Engineering, Chaoyang University of Technology, Taichung 413310, Taiwan
[2] Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taichung 413310, Taiwan
* Correspondence: jschen26@cyut.edu.tw (J.-S.C.); rwhung@cyut.edu.tw (R.-W.H.)

Abstract: Many highly contagious infectious diseases, such as COVID-19, monkeypox, chickenpox, influenza, etc., have seriously affected or currently are seriously affecting human health, economic activities, education, sports, and leisure. Many people will be infected or quarantined when an epidemic spreads in specific areas. These people whose activities must be restricted due to the epidemic are represented by targets in the article. Managing targets by using targeted areas is an effective option for slowing the spread. The Centers for Disease Control (CDC) usually determine management strategies by tracking targets in specific areas. A global navigation satellite system (GNSS) that can provide autonomous geospatial positioning of targets by using tiny electronic receivers can assist in recognition. The recognition of targets within a targeted area is a point-in-polygon (PtInPy) problem in computational geometry. Most previous methods used the method of identifying one target at a time, which made them unable to effectively deal with many targets. An earlier method was able to simultaneously recognize several targets but had the problem of the repeated recognition of the same targets. Therefore, we propose a GNSS coordinate recognition algorithm. This algorithm can efficiently recognize a large number of targets within a targeted area, which can provide assistance in epidemic management.

Keywords: GNSS; recognition; point-in-polygon; computational geometry; infectious disease; isolate

check for updates

Citation: Chen, J.-S.; Kuo, C.-M.; Hung, R.-W. An Efficient GNSS Coordinate Recognition Algorithm for Epidemic Management. *Algorithms* **2023**, *16*, 132. https://doi.org/10.3390/a16030132

Academic Editors: Dmytro Chumachenko and Sergiy Yakovlev

Received: 15 December 2022
Revised: 18 February 2023
Accepted: 20 February 2023
Published: 1 March 2023

1. Introduction

Infectious diseases are diseases caused by microorganisms [1]. Many infectious diseases are highly contagious, such as COVID-19, monkeypox, chickenpox, influenza, and so on, and these have seriously affected or are currently seriously affecting human health, economic activities, education, sports, and leisure. Restriction, tracing, and isolation of people's movements when an epidemic spreads are effective ways to slow its spread [2,3]. A GNSS allows a tiny electronic receiver to determine its position, including its longitude, latitude, and altitude (elevation), with an accuracy of a few centimeters to a few meters by using time signals transmitted by a satellite radio along the line of sight [4,5]. Since GNSSs can provide the precise location information of targets, they have been widely used in various remote sensing applications [6–12], such as in epidemic monitoring and geographic information systems. In this study, we design a method of using longitude and latitude coordinates to calculate if a large number of targets are in an area while considering a large number of targets and a wide range of the target area. Under the demand of epidemic prevention, we want to know in real-time whether the targets are in a specific range or stay within the official designation. Therefore, we only need to define this range, and our method can obtain the answer.

In geography, an area can be represented by a polygon. A target is a geographic point, which is a point with latitude and longitude coordinates. Determining if a geographic point is within an area is a PtInPy problem. Much of the literature on such issues is focused on special conditions relating to points and polygons [13–18]. In simple terms, every method contains a PtInPy function and uses a geographical point and a polygon as its input. This PtInPy function can confirm if the point is an inner point of this polygon or an outer point of this polygon. When the number of targets is huge, it is inefficient to confirm them one by one by using a PtInPy function. However, there may be many targets to be tracked in many areas, such as in epidemic management.

Take the coronavirus disease 2019 as an example. It is a contagious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The first known case was in Wuhan, China, in December 2019. After that, the disease spread worldwide, leading to the COVID-19 pandemic. According to the World Health Organization's weekly report, as of 22 May 2022, over 522 million confirmed cases and over 6 million deaths had been reported globally [19]. This example generated many targets that were distributed over broad and numerous areas. The CDC often determines its management strategy by tracking targets within such areas. To find the targets in a specific area, they can be confirmed one by one by using a PtInPy function. However, this way could be more efficient because the number of targets is large, and the targets within an area represent only a tiny fraction of the total targets. The number of times that a PtInPy function is used reflects the performance of a method in finding targets within a given area.

In [20,21], methods for reducing the use of PtInPy functions were proposed. In [20], the authors covered an area with a rectangle, and a PtInPy function was used only for the targets within this rectangle. However, because the range of the rectangle could be much more extensive than the range of the target area, the use of the PtInPy function was still very high. In [21], the authors considered that the number of targets was huge. Therefore, they also included a way to reduce the number of points that were confirmed by the PtInPy function. However, this method has two main areas for improvement. First, the range of a small area is significant, but there needs to be a plan in the technique. Second, the small areas planned by this method overlapped. These shortcomings caused a bottleneck in the performance of this method. The topic of PtInPy has been discussed in many fields in the past, especially in the GIS field. Some methods can obtain good results with area preprocessing if these target areas are fixed. However, targets with areas are a lot in epidemic tracking and may continuously change. Therefore, when considering many targets and regions and evolving rapidly, this research needs to include those skills.

In light of the above discussion, we propose an efficient GNSS coordinate recognition algorithm for epidemic management. This research is applicable to the positioning of two-dimensional coordinates, not only GNSS coordinates. However, for epidemic management, the coordinates of GNSS latitude and longitude need to be known in order to map this to the location in the real world. The main novelty of this algorithm is the dynamic blocking and real-time setting of range. This algorithm can be applied in order to recognize targets within a targeted area, even if the targeted area's range is very large and there are numerous targets. Furthermore, for most targets, it is not necessary to use a PtInPy function to confirm whether these targets are inside or outside of a targeted area in order to save computation time. In addition, this method also considers the capacities of database access and computation to achieve efficient recognition.

The planning of a simulation requires areas with an extensive range as the targeted areas and many geographical points with latitude and longitude coordinates as the targets. For our samples, we used the check-in locations on a well-known social networking site as targets—corresponding to about one million geographical points—and three administrative cities as the targeted areas. These check-in places were locations where people had socialized in the past. These conditions made our simulations more realistic. In addition, these check-in places had their names, textual descriptions, and check-in counts, i.e., a quantitative measure of their popularity, as well as their latitude and longitude values, on the website. Many valuable applications have also been based on these check-in places [22–26].

The rest of this paper is organized as follows. In Section 2, we introduce the related work. Section 3 introduces the system architecture and describes the PtInPy problem. Section 4 describes the recognition algorithm where we use a city as the range to show the steps of the algorithm. Section 5 provides the simulated environment and performs a demonstration. This demonstration offers targets within three selected fields of hundreds to thousands of square kilometers calculated from more than one million targets. In Section 6, we offer the simulation results. Finally, in Section 7, we give a conclusion.

2. Related Work

Many infectious diseases are highly contagious and seriously affect human health, economic activities, education, sports, and leisure [1]. Restricting the movement of people (e.g., with a quarantine) is an effective infection control measure when an epidemic spreads [2,3]. GNSSs, such as the US Global Positioning System (GPS) [27] and the Russian Global Navigation Satellite System (GLONASS) [28], can provide autonomous geospatial positioning. They are widely used in various applications [6–12], such as prevention and control. In [6], some methods were proposed for the regional and global surveillance of infectious diseases; for example, they used GPS data to track the detailed movement patterns of individuals. The positions obtained from GPS can also be entered directly into a geographic information system (GIS). A GIS can locate highly endemic areas and high-risk groups and can help make resource allocation decisions [7]. In [9], the aim was to emphasize the use of GNSS satellites in environmental monitoring. In [10], smartphones were used to track COVID-19 patients and diagnosed people. In [11], mapping inundation dynamics and the extent of flooding were found to be essential in various applications, such as the prediction of the spread of infectious diseases, and a GNSS can help with this application. Finally, in [12], an Indian COVID-19 tracker was analyzed.

The confirmation of whether a target is inside or outside of a targeted area can be mapped as a PtInPy problem. Many works in the literature discussed related issues in the past [13–18]. In [13], a detailed discussion of the PtInPy problem for arbitrary polygons was presented. No single algorithm is the best in all categories, so this study compared the capabilities of the better algorithms. The variables examined, including different types of polygons, the amount of memory used, and the preprocessing costs, were discussed in [14]. In [15], the authors outlined a fast and efficient method for determining if a coordinate point lies within a closed region or polygon that is defined by any number of coordinate points. In [16], an efficient polygon clipping algorithm, as opposed to a rectangularly clipped window, was proposed. It used a parametric representation of polygon edges. By using the concept of point clipping, it found the required intersection points of the edges of a polygon with clip window boundaries. In [17], an extension of a winding number and PtInPy algorithm was presented. In [18], a provably correct implementation of a PtInPy method that was based on the computation of the winding number was presented.

Simply, a PtInPy function can also be used to represent a function whose inputs are a geographical point and a polygon. Applying a PtInPy function will assert that the geographical point is an inner point or an outer point of the polygon. In addition, O_PtInPy (Only PtInPy) can also be used; it is a type of PtInPy method in which each target has to be confirmed by applying a PtInPy function to obtain a result. The methods used in [13–18] were of the O_PtInPy type, and they could not handle many targets. In [20,21], methods that did not require every target to be confirmed by applying a PtInPy function were proposed. In [20], the authors planned a large rectangle that could cover the entire target area, and the targets outside the rectangle did not need to be confirmed by a PtInPy function. In short, L_Rect (large rectangle) was used to represent this method.

However, the range of the rectangle may be much larger than the range of the specific area, and the number of targets confirmed by the PtInPy function will still be very high. When the number of targets is very high, it is unreasonable to load the data of all of the targets into the program. In addition, the computer's computational capabilities also need to be considered. A more feasible method is to store the data of the targets in a database and then load amounts of data that can be processed into the program one by one according to the capacity of the computer. In [21], the authors considered that the number of targets was very high, so the targeted data were stored in a database system. Then, several small areas were planned, and the ranges of these small areas could cover the entire targeted area. Since each small area had the shape of a circle, the method was represented as c_. Then, the targets within a small area were retrieved from the database. In addition, this method also included a mechanism that allowed some targets within a small area not to be confirmed by the PtInPy function. However, this method had a major disadvantage because the small areas overlapped. This led to a bottleneck in its performance.

GIS is a broad subject, combining geography and cartography, and has been widely used in map production, information management, and registration management of geographic-related data in various businesses. The key to any GIS is a database containing representations of geographic phenomena, modeling their geometry (location and shape) and their properties or attributes. A GIS database can be stored in many forms, such as collecting individual data files or a single spatially enabled relational database because it has pre-collected and managed these data and cooperated with the development of many tools. When new information from the real world enters the system, the system can quickly reflect the answer. Its characteristic is that the pre-processed data are stable, and the value of the data is inquiring about changing rapidly. However, in our research, the time–space relationship between targets and areas is easy to change. Therefore, it is hard to use prior modeling to achieve time efficiency.

The planning of a simulation requires an extensive range of areas to be used as targeted areas and many geographical points to be used as targets. Well-known check-in places posted on a social network service (SNS) were used as targets, and three administrative cities in Taiwan were used as the targeted areas. According to this study's methods, administrative regions could also be used as check-in places. These check-in places were locations where people had socialized in the past. The positioning and classification of the administrative areas for many check-in places can also be extended to many valuable applications [22–26], such as finding delicious food or locations related to religious beliefs. In [22], what drives users to check in on Facebook was discussed. For example, belief in Wang-Ye is one of the most popular folk religions among the Chinese people. In [23], check-in places were used as an example to find locations related to this belief. In [26], a method for collecting big data from Facebook check-in places and finding places related to Taiwanese beef noodles was proposed.

3. System Architecture and Problem Description

In this section, we introduce the system architecture and describe the PtInPy problem. Figure 1 presents the system architecture. We assume that each target has a mobile device with a GNSS receiver. The mobile device can receive GNSS signals, convert them into coordinate data, and send the data to database-based storage according to the transmission mechanism of a wired or wireless network. Therefore, the targets' real-time latitude and longitude coordinates are stored in a database. By using Structured Query Language (SQL), the latitude and longitude coordinates in the database and the queried data can be sent back to a computer. The data related to a specific geographical area are also stored in the computer. A geographical area is represented as a polygon. The computer performs the operation of coordinate recognition with the input of a geographical area P and can obtain the targets in the area P. Under the demand of epidemic prevention, we want to know in real-time whether the targets are in a specific range or stay within the official designation. Therefore, we only need to define this range as a polygon. Then, using this polygon as input, our method can obtain the answer.

A target g is a geographic point with latitude $g_{.x}$ and longitude $g_{.y}$ coordinates. A targeted area is a polygon. A polygon is composed of edges with geographical points. These edges enclose a measurable interior. Accordingly, a polygon P with n points is defined as $P = \{p_0, p_1, \ldots, p_{n-1}, p_n\}$, where $p_0 = p_n$, and the straight line segment from point p_i to p_{i+1} is the edge for $i = 0, 1, \ldots, n-1$.

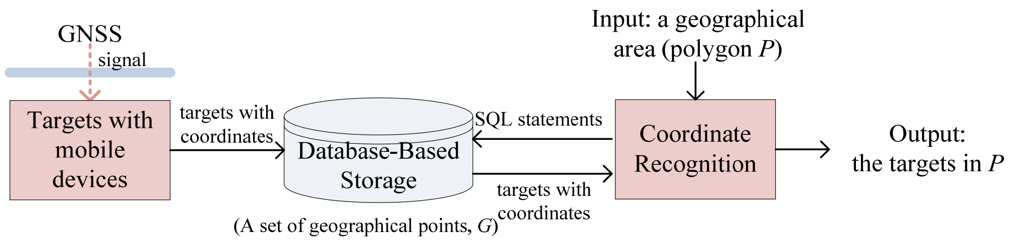


Figure 1. System architecture. Each target has a mobile device with a GNSS receiver. The mobile device can receive GNSS signals, convert them into coordinate data, and send the data to database-based storage. A computer performs the operation of coordinate recognition with the input of a geographical area P and can obtain the targets in the area P.

A ray from a point is a straight line that starts from that point and goes on in any fixed direction. In this article, $ray(g)$ is a horizontal extension segment starting from point $g = (g_{.x}, g_{.y})$ and going to the point $(0, g_{.y})$. If the number of intersections between the edges of polygon P and $ray(g)$ is odd, this indicates that point g is an inner point of polygon P. Otherwise, if the number is even, this indicates that point g is an outer point of polygon P. For example, let P and P' be two different polygons, with the space of P being much larger than the space of P'. There are three statuses of spatial overlap between polygons P and P'. The first status is that in which polygon P' is an inner polygon of polygon P. This means that the entire space of P' is part of P. This second status is that in which polygon P' is a polygon intersecting with polygon P. This means that a certain part of the space of P' is part of the space of P. The third status is that in which polygon P' is an outer polygon of polygon P. This means that no part of the space of P' is part of the space of P. If P' is a polygon intersecting with P, P' has at least one edge that intersects with the edges of P. If polygon P' is an inner polygon of P, it must satisfy the following two conditions.

- For any point p of P', p is an inner point of P.
- For each edge e of P', edge e and all edges of P have no intersections.

Figure 2 provides an example of a polygon with point and space statuses. This polygon P is composed of six points ($p_0, p_1, \ldots$, and p_6), where $p_0 = p_6$, and the straight line segment from point p_i to p_{i+1} is the edge for $i = 0, 1, \ldots, 5$. There are three geographic points (g_0, g_1, and g_2) with their corresponding rays ($ray(g_0)$, $ray(g_1)$, and $ray(g_2)$). The number of intersections between $ray(g_0)$ and the edges of P is 1 (odd number), meaning that g_0 is an inner point of P. The numbers of intersections of $ray(g_1)$ and the edges of P and those of $ray(g_1)$ and the edges of P, respectively, are two and zero; both numbers are even. This means that point g_1 and point g_2 are both outer points of P. There are three different polygons (S_0, S_1, and S_2). The entire space of S_0 is a subspace of P, so S_0 is an inner polygon of P. A certain part of the space (not all of the space) of $S1$ is part of the space of P, so S_1 is a polygon intersecting with P. No space of S_2 is part of the space of P, so S_2 is an outer polygon of P. In order to determine a polygon's architecture with point and space statuses, the calculation of the intersections of segments is necessary. It is necessary to calculate whether two segments have an intersection or not. In [29], a computer-friendly method was proposed. Suppose that there are two lines. The first line crosses over both point ga and point gb. The other line crosses over both point gc and point gd. First, as shown in (1), α can be evaluated.

$$\alpha = (g_{a.x} - g_{b.x}) \times (g_{c.y} - g_{d.y}) - (g_{a.y} - g_{b.y}) \times (g_{c.x} - g_{d.x}). \tag{1}$$

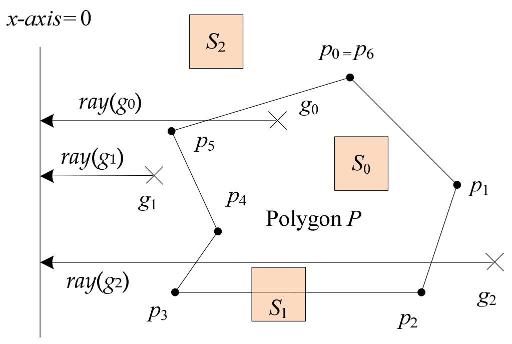


Figure 2. Example of a polygon with point and space statuses.

If the value of α is 0, the two lines are parallel. Otherwise, there is an intersection between the two lines. Next, κ_1 and κ_2 can be evaluated as in (2) and (3), respectively.

$$\kappa_1 = ((g_{a}.x - g_{c}.x) \times (g_{c}.y - g_{d}.y) - (g_{a}.y - g_{c}.y) \times (g_{c}.x - g_{d}.x))/\alpha \tag{2}$$

$$\kappa_2 = ((g_{a}.x - g_{b}.x) \times (g_{a}.y - g_{c}.y) - (g_{a}.y - g_{b}.y) \times (g_{a}.x - g_{b}.x))/\alpha \tag{3}$$

If the value of κ_1 is between 0 and 1, the intersection is between g_a and g_b. If the value of κ_2 is between 0 and 1, the intersection is between g_c and g_d. In other words, if segment (g_a, g_b) and segment (g_c, g_d) intersect, the value of κ_1 is between 0 and 1, and the value of κ_2 is between 0 and 1.

Figure 3 provides examples of lines (or segments) with intersections. There are three lines (l_1, l_2, and l_3). Line l_1 passes through points (1, 3) and (4, 3). Line l_2 passes through points (1, 1) and (4, 1). Line l_3 passes through points (3, 4) and (2, 3). In lines l_1 and l_2, α is 0, which means that the two lines are parallel. In lines l_2 and l_3, α is -6, κ_1 is 0.17, and κ_2 is 1.5, which means that the intersection is between (1, 1) and (4, 1). In lines l_1 and l_3, α is -6, κ_1 is 0.5, and κ_2 is 0.5, which means that the intersection is between (1, 3) and (4, 3), and there is also an intersection between (2, 2) and (3, 4).

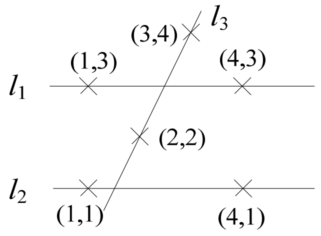


Figure 3. Example of lines with intersections.

Assume that G is the set of all geographical points. All points in G are stored in a database and can be retrieved with Structured Query Language (SQL) by using a database management system [27,28]. In this study, MySQL [30], an open-source relational data-stream management system, was applied.

4. Coordinate Recognition Algorithm

This section describes the recognition algorithm and uses a city as the range to show the steps of the algorithm. The purpose of our research is to assist the epidemic control. Therefore, we use pseudocode style to describe our method, which is to make this method easier to implement.

4.1. The Proposed Algorithm

In this section, a coordinate recognition algorithm will be introduced. We also assume that there is a large number of geographical points (targets) that are stored in table 'G' of a unique database, and there is also a unique polygon $P = \{p_0, p_1, \ldots, p_n\}$ that refers to a targeted area. This algorithm will be used to calculate the inner points of polygon P from table 'G', and several functions (SegSegInt, PtInPy, SqrInPy, RectOfPy, and RectPtnToSqr) are used. These functions are listed in Table 1.

Table 1. Functions.

Function	Description
SegSegInt	The input required by this function is two line segments, which can be used to calculate whether the two line segments intersect.
PtInPy	The input required by this function is a geographical point and a polygon. First, the function uses SegSegInt to calculate the intersections among the ray of this point and the edges of this polygon. Then, it calculates whether the geographical point is inside the polygon based on the result.
SqrInPy	The input required by this function is a square and a polygon. In this function, PtInPy is used to calculate whether the four points of the square are inside the polygon. According to this result, the spatial relationship between the square and the polygon is divided into inner, outer, and intersected.
RectOfPy	The input required by this function is a polygon, and it will identify a minimum rectangle that can cover this polygon.
RectPtnToSqr	The input required by this function is a coordinate range that can identify a rectangle, and it will divide the range of this rectangle into a square set.

The input required by this method is a polygon, a set of geographical points, and the side length of a square. Because the size of geographical points is very large, these geographical points are stored in a database and can be retrieved through the query command of the database. Our method will first define a rectangle that can cover this polygon. Then, it will partition the range of this rectangle into several squares whose side length is SL. Then, for each square, it will calculate the space–overlap status between the square and the polygon. When the range of the square is inside the polygon or is intersected with the polygon, it will retrieve the geographical points within the square range from the database. In the inner status, the retrieved points are the polygon's inner points without being confirmed through PtInPy. In the intersected status, these points must be confirmed again through PtInPy.

Algorithm 1 is the SegSegInt algorithm. It is a segment–segment intersection algorithm whose input comprises four points: g_a, g_b, g_c, and g_d. The four points compose the segment from g_a to g_b and the segment from g_c to g_d. Applying it will result in a value of 1 or 0 to represent that the two segments intersect or do not intersect.

Algorithm 1. SegSegInt(point g_a, point g_b, point g_c, point g_d)

```
1.  $\alpha := (g_a.x - g_b.x) \times (g_c.y - g_d.y) - (g_a.y - g_b.y) \times (g_c.x - g_d.x)$;
2.  **if** $\alpha = 0$ **then return** 0;
3.  **else**
4.      $\kappa_1 := ((g_a.x - g_c.x) \times (g_c.y - g_d.y) - (g_a.y - g_c.y)\times(g_c.x - g_d.x))/\alpha$;
5.      $\kappa_2 := ((g_a.x - g_b.x) \times (g_a.y - g_c.y) - (g_a.y - g_b.y)\times(g_a.x - g_b.x))/\alpha$;
6.      **if** ($\kappa_1 \geq 0$ and $\kappa_1 \leq 1$) and ($\kappa_2 \geq 0$ and $\kappa_2 \leq 1$) **then return** 1;
7.      **else return** 0;
8.      **end if**
9.  **end if**
```

Algorithm 2 is the PtInPy algorithm. It is a point-in-polygon algorithm whose inputs are a point g and a polygon $P = \{p_0, p_1, \ldots, p_n\}$. This algorithm is used to evaluate whether a point g is an inner point of polygon P or if that point g is an outer point of polygon P.

Applying it will result in a value of 1 or 0, which will indicate if g is an inner point of P or if g is an outer point of P. In this algorithm, the *count* variable is used to count the number of intersections between the rays of g and the n edges of P. Here, mod(*count*, 2), which is a modulo operation that returns the remainder after *count* is divided by 2, is used. It combines each edge (p_i, p_{i+1}) of P, where $i = 0, 1, \ldots, n-1$, and point g into the segment (g_a, g_b) and the segment (g_c, g_d). If the two segments intersect, the value of *count* is increased by 1. Finally, if the value of *count* is even (i.e., mod(*count*, 2) = 1), it returns 0. Otherwise, it returns 1.

Algorithm 2. PtInPy(point g, polygon P)

```
1.  count := 0;
2.  for i := 0 to $n - 1$ do
3.      $g_a := p_i$; $g_b := p_{i+1}$; $g_c := g$; $g_{d.x} := 0$; $g_{d.y} := g_{.y}$;
4.      if (SegSegInt ($g_a$, $g_b$, $g_c$, $g_d$) = 1) then
5.          count := count + 1;
6.      end if
7.      if (mod(count, 2) = 1) then return 1;
8.      else return 0;
9.      end if
10. end for
```

Algorithm 3 is the SqrInPy algorithm. It is a square-in-polygon algorithm whose inputs are a square $S = \{s_0, s_1, \ldots, s_4\}$ and a polygon $P = \{p_0, p_1, \ldots, p_n\}$. This algorithm is used to evaluate the space–overlap relationship between square S and polygon P. Applying it will result in a value of 0, 1, or 2, representing that S is an inner square of P, S is square intersecting with P, or S is an outer area of P. Lines 1 to 6 are used to calculate how many points of S are inner points of P, which is where PtInPy is applied, and the *count_pt* variable is used to count the number of inner points. Lines 7 to 18 are used to count the intersections between S and P, which is where SegSegInt is applied, and *count_edge* is used to count the number of intersections.

Lines 19 to 22 are used to evaluate the coverage relationships between the square S and polygon P. If the four vertices of square S are all outer points of polygon P (i.e., *count_pt* = 0) and the four edges of S and n edges of the polygon do not intersect (i.e., *count_edge* = 0), square S is an outer square of polygon P. On the other hand, if all four vertices of square S are inner points of polygon P (i.e., *count_pt* = 4) and the four edges of square S and n edges of the polygon do not intersect (i.e., *count_edge* = 0), square S is an inner area of polygon P. Otherwise, S intersects with polygon P.

Algorithm 4 is the RectOfPy algorithm. It is a rectangle-of-polygon algorithm whose input is a polygon $P = \{p_0, p_1, \ldots, p_n\}$. This algorithm is used to evaluate a minimal rectangular space that completely covers polygon P. Applying it will result in four values, i.e., x_{min}, x_{max}, y_{min}, and y_{max}. The dataset R contains these four values, and R[0], R[1], R[2], and *R[3]* represent these values. In this algorithm, these values are stored in the variables x_{min}, x_{max}, y_{min}, and y_{max}. The algorithm first assigns the coordinate values of point p_0 to these variables. Then, in lines 2 to 7, the values of $p_{j.x}$ and $p_{j.y}$ of each point p_j are compared with the values of these variables. If the value of $p_{j.x}$ is smaller (or larger) than the value of x_{min} (or x_{max}), the value of x_{min} (or x_{max}) is assigned as the value of $p_{j.x}$. Similarly, if the value of $p_{j.y}$ is smaller (or larger) than the value of y_{min} (or y_{max}), the value of y_{min} (or y_{max}) is assigned as the value of $p_{j.y}$. Finally, the values of these four variables are assigned to dataset R. The values of x_{min}, x_{max}, y_{min}, and y_{max} can define a minimal rectangle that can completely cover polygon P.

Algorithm 3. SqrInPy(square *S*, polygon *P*)

```
1.  $count\_pt$ := 0;
2.  for $i$ := 0 to 3 do
3.    if (PtInPy($s_i$, $P$) = 1)
4.    then $count\_pt$ := $count\_pt$ + 1;
5.    end if
6.  end for
7.  $count\_edge$ := 0;
8.  for $j$ := 0 to $n - 1$ do
9.    for $i$ := 0 to 3 do
10.     $g_a$ := $p_j$;
11.     $g_b$ := $p_{j+1}$;
12.     $g_c$ := $s_i$;
13.     $g_d$ := $s_{i+1}$;
14.     if(SegSegInt($g_a$, $g_b$, $g_c$, $g_d$) = 1)
15.     then $count\_edge$ := $count\_edge$+1;
16.     end if
17.   end for
18. end  for
19. if ($count\_pt$ = 4 and $count\_edge$ = 0) then $status$ := 0;
20. else if ($count\_pt$ = 0 and $count\_edge$ = 0) then $status$ := 2;
21. else $status$ := 1;
22. end if
23. return $status$;
```

Algorithm 4. RectOfPy(polygon *P*)

```
1. $x_{min}$ := $p_{0.x}$; $x_{max}$ := $p_{0.x}$; $y_{min}$ := $p_{0.y}$; $y_{max}$ := $p_{0.y}$;
2. for $j$ := 1 to $n$ do
3.   if $p_{j.x} < x_{min}$ then $x_{min}$ := $p_{j.x}$; end if
4.   if $p_{j.x} > x_{max}$ then $x_{max}$ := $p_{j.x}$; end if
5.   if $p_{j.y} < y_{min}$ then $y_{min}$ := $p_{j.y}$; end if
6.   if $p_{j.y} > y_{max}$ then $y_{max}$ := $p_{j.y}$; end if
7. end for
8. $R$ := {$x_{min}$, $x_{min}$, $y_{min}$, $y_{max}$};
9. return ($R$);
```

The application of Algorithm 4 results in a dataset *R*. The values of dataset *R* define the range of a polygon. The dataset *R* contains four values, and *R*[0], *R*[1], *R*[2], and *R*[3] represent these values. Algorithm 5 is used to divide this polygon into several squares of equal size, where the side length of each square is *sl*. Algorithm 5 is the RectPtnToSqr algorithm. It is a rectangle-partition-to-square algorithm, and its inputs are the five values of x_{min}, x_{max}, y_{min}, y_{max}, and *sl*. It uses the values of x_{min}, x_{max}, y_{min}, and y_{max} to define the range of a rectangle into its length and width, respectively (($x_{max} - x_{min}$) and ($y_{max} - y_{min}$)). It then partitions the range into a set *SS* of $numx \times numy$ squares, where ceiling() is applied. The ceiling() function is a least-integer function. Its input is a real number, and its output is an integer that is not smaller than the input value. This set defines each square *S* by using its upper-left vertex ($s_{0.x}$, $s_{0.y}$). Accordingly, square *S* can be defined by using s_0 and the side length s*l*.

Algorithm 5. RectPtnToSqr(x_{min}, x_{max}, y_{min}, y_{max}, sl)

```
1.  $num_x$ := ceiling $((x_{max} - x_{min})/r)$;
2.  $num_y$ := ceiling $((y_{max} - y_{min})/r)$;
3.  $SS := \emptyset$;
4.  for $i$ := 0 to $num_x - 1$ do
5.      for $j$ := 0 to $num_y - 1$ do
6.          $s_{0.x} := x_{min} + i \times sl$;
7.          $s_{0.y} := y_{min} + j \times sl$;
8.          $SS := SS \cup \{s_0\}$;
9.      end for
10. end for
11. return ($SS$)
```

Algorithm 6 is the CoordRecognit algorithm. It is a coordinate recognition algorithm, and its inputs are a polygon *P*, a point set *G*, and the side length *r* of a square. Applying it will return all inner points of polygon *P*. This algorithm first defines a rectangle *R*, which can cover polygon *P,* through RectOfPy. Next, this rectangle *R* is partitioned into *SS*, a set of squares *SS* with a side length *sl*, through RectPtnToSquare. For each square s_0 of *SS*, where s_0 is the upper-left vertex of the square, its vertices are composed of a square $S = \{s_0, s_1, s_2, s_3, s_4(=s_0)\}$. Then, the space–overlap status between square *S* and polygon *P* is evaluated through SqrInPy. If *S* is an inner square of or a square intersecting with polygon *P*, i.e., *status* = 0 **or** *status* = 1, the SQL statement is queried in order to retrieve the inner points from table 'Q', and they are stored in set G_T. If *S* is an inner square of *P*, i.e., *status* = 0, set G_T is directly stored in set G_P. If *S* is a square intersecting with *P*, i.e., *status* = 1, each point *g* of G_P must be confirmed through PtInPy. If point *g* is an inner point of *P*, incrementally, point *g* is stored in G_P. Finally, G_P is the set of all inner points of *P*.

Algorithm 6. CoordRecognit (polygon *P*, point set *G*, side length *sl*)

```
1.  $R$ := RectOfPy($P$);
2.  $x_{min}$ := $R$[0]; xmax := R[1]; ymin := R[2]; ymax := R[3];
3.  $SS$ := RectPtnToSqr ($x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $sl$);
4.  $G_P := \emptyset$; $G_T := \emptyset$;
5.  for each $s_0$ in $SS$ do
6.      $s_0 := (s_{0.x}, s_{0.y})$;
7.      $s_1 := (s_{0.x} + sl, s_{0.y})$;
8.      $s_2 := (s_{0.x} - sl, s_{0.y} - sl)$ ;
9.      $s_3 := (s_{0.x}, s_{0.y} - sl)$;
10.     $s_4 := s_0$;
11.     $S := \{s_0, s_1, s_2, s_3, s_4\}$;
12.     $status$ := SqrInPy($S$, $P$);
13.     if $status$ = 0 or $status$ = 1 then
14.         $G_T$:={SELECT * FROM 'G' WHERE 'x' $\geq s_{0.x}$ AND 'y' $\geq s_{0.y}$
                    AND 'x' > ($s_{0.x} + r$) AND 'y' > ($s_{0.y} + r$);};
15.     end if
16.     if $status$ = 0 then
17.         $G_P := G_P \cup G_T$;
18.     end if
19.     if $status$ = 1 then
20.         for each $g$ in $G_T$ do
21.                 if PtInPy($g$, $P$) = 1 then $G_P := G_P \cup \{g\}$; end if
22.         end for
23.     end if
24. end for
25. return ($G_P$);
```

4.2. Example

As an example, we took an area with a longitude ranging from 120.6 to 122.9 East and a latitude ranging from 24.8 to 25.4 North. We collected 267,858 geographic points, which were check-in places obtained from a well-known social network service platform, in this area [25]. This range also contained the capital of Taipei City. We used the city as the targeted area and the check-in places as the targets to calculate the inner points of this city. Figure 4 provides the distribution of 10,000 targets that were randomly selected from the 267,858 points. The results related to the layouts of target distributions in this study, such as those shown in Figures 4 and 5, and others, were obtained by using Ajax and the Google Maps API [31]. The city's polygon P had 2054 points that were selected along the outermost area, as shown in Figure 5a. A dataset R was obtained by using RectOfPy with the polygon P as input, where R contained an $x_{\min}$ value of about 121.457, an $x_{\max}$ value of about 121.666, a $y_{\min}$ value of about 24.960, and a $y_{\max}$ value of about 25.210. Figure 5b provides the corresponding shape, which formed four points (r_0(=($x_{\min}$, $y_{\min}$)), r_1(=($x_{\max}$, $y_{\min}$)), r_2(=($x_{\max}$, $y_{\max}$)), and r_3(=($x_{\min}$, $y_{\max}$))) that were the four vertices of a rectangle, i.e., the minimal rectangle covering polygon P. Figure 5b provides the shape of this rectangle.

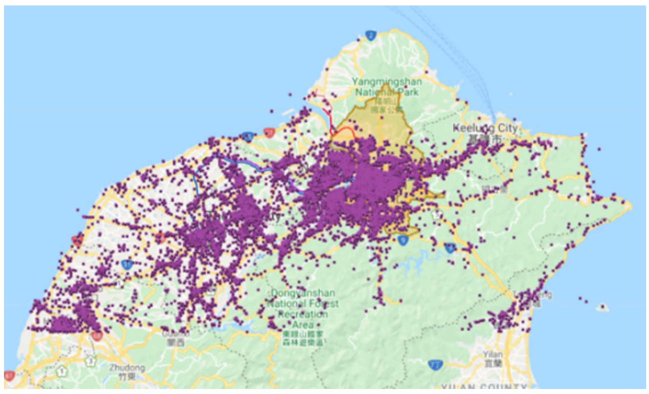

Yangmingshan National Park
Keelung City
Dongyanshan National Forest Recreation Area
Zhudong
Yilan

Figure 4. An area as an example for evaluating the targeted points in a city.

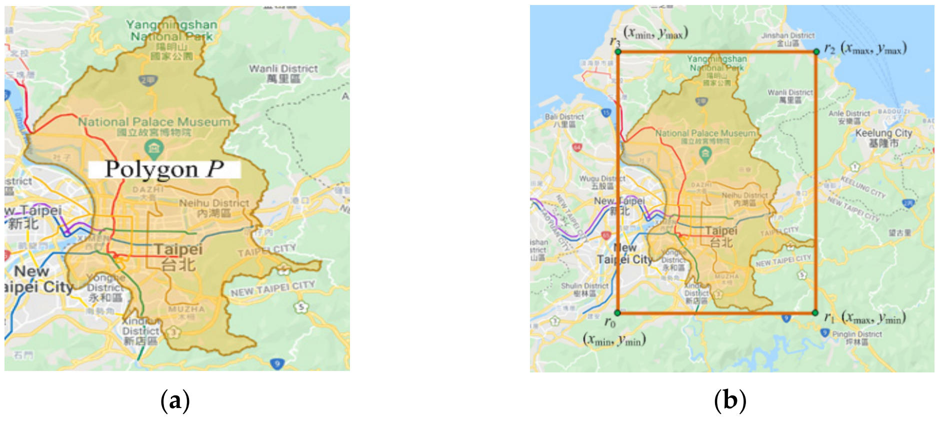
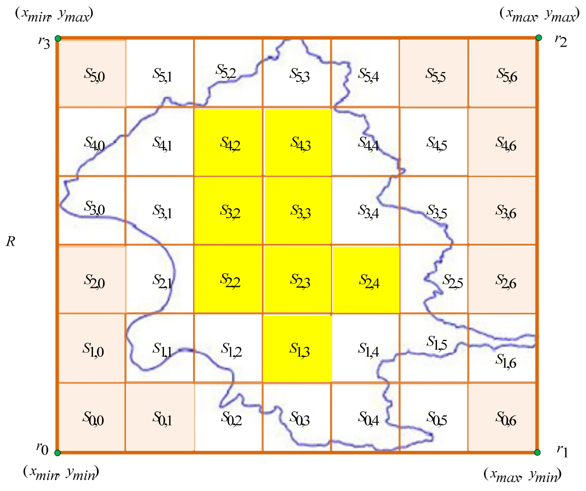

Yangmingshan National Park
National Palace Museum
Polygon P
Wanli District
Neihu District
Taipei
New Taipei City
NEW TAIPEI CITY
r_3 $(x_{\min}, y_{\max})$
r_2 $(x_{\max}, y_{\max})$
r_1 $(x_{\max}, y_{\min})$
r_0 $(x_{\min}, y_{\min})$
Keelung City
(a)
(b)

Figure 5. Polygon of the city and a minimal rectangle covering the polygon. (**a**) Polygon of the city; (**b**) Minimal rectangle covering the polygon.

Then, by using RectPtnToSquare, this rectangle could be divided into a set SS of squares. Figure 6 shows that rectangle R was divided into a square set SS with 7×6 squares. For each square S of SS, the space–overlap status of P was calculated through SqrInPy. When square S was an inner square of or a square intersecting with polygon P, the inner points of square S were calculated through the corresponding SELECT statement. If S was an inner square, the inner points of square S were also the inner points of polygon P and were directly stored in the set GP without confirmation through PtInPy. If S was a square that intersected, the points that PtInPy confirmed as inner points of P were stored in the set GP. Finally, the set GP contained all inner points of P. In this example, there were 89,209 inner points in the targeted city. Figure 7 provides the distribution of the 10,000 inner points that were randomly selected from 89,209 points.



(x_{min}, y_{max})
(x_{max}, y_{max})
r_3
r_2
R
r_0
r_1
(x_{min}, y_{min})
(x_{max}, y_{min})
$S_{5,0}$ $S_{5,1}$ $S_{5,2}$ $S_{5,3}$ $S_{5,4}$ $S_{5,5}$ $S_{5,6}$
$S_{4,0}$ $S_{4,1}$ $S_{4,2}$ $S_{4,3}$ $S_{4,4}$ $S_{4,5}$ $S_{4,6}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$ $S_{3,4}$ $S_{3,5}$ $S_{3,6}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$ $S_{2,4}$ $S_{2,5}$ $S_{2,6}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,4}$ $S_{1,5}$ $S_{1,6}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$ $S_{0,4}$ $S_{0,5}$ $S_{0,6}$

Figure 6. Partitioning a rectangle into a set of squares.

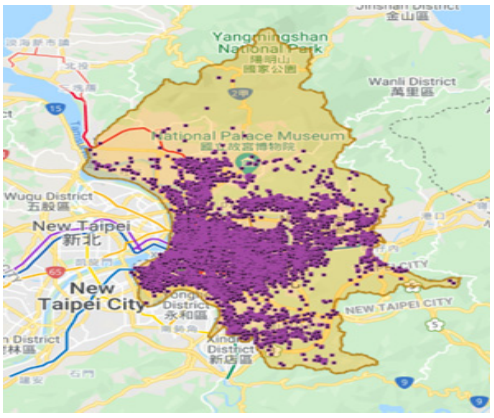


Figure 7. The distribution of the targeted points in Taipei City.

5. Simulated Environment and Demonstration

This section first introduces the simulated environment and then demonstrates the results of applying this algorithm to calculate the inner points of targeted areas. Next, we offer targets within three selected fields of hundreds to thousands of square kilometers calculated from more than one million targets.

5.1. Simulated Environment

The simulated geographic area was mainly the island of Taiwan, which is located between 120 and 122 degrees East for the longitude and 22 and 25 degrees North for the latitude. It covers an area of 35,808 km^2 and has a population of 23.7 million inhabitants.

Figure 8 presents the simulated environment. Figure 8a shows the distribution of residents per kilometer in villages in 2019 [32]. The targeted points for the simulation were based on geographical points acquired from a famous social network service platform, with a total of 1,112,188 points [24]. Figure 8b presents the distribution of the targeted points, which included 10,000 points that were randomly selected from all targeted points. Through a comparison of Figure 8a,b, The area currently contains six municipalities, of which three—Taipei City, Taichung City, and Kaohsiung City—were selected as the targeted areas. Taipei City is Taiwan's political, economic, educational, and cultural center, and it is one of the major hubs in East Asia. Taichung City, which is located in central Taiwan, plays an essential role in Taiwan's economic development and transportation systems. Kaohsiung City is a special municipality in southern Taiwan, and it is Taiwan's third most populous city and the largest city in southern Taiwan. For brevity, Taipei City, Taichung City, and Kaohsiung City are called City 1, City 2, and City 3, respectively.

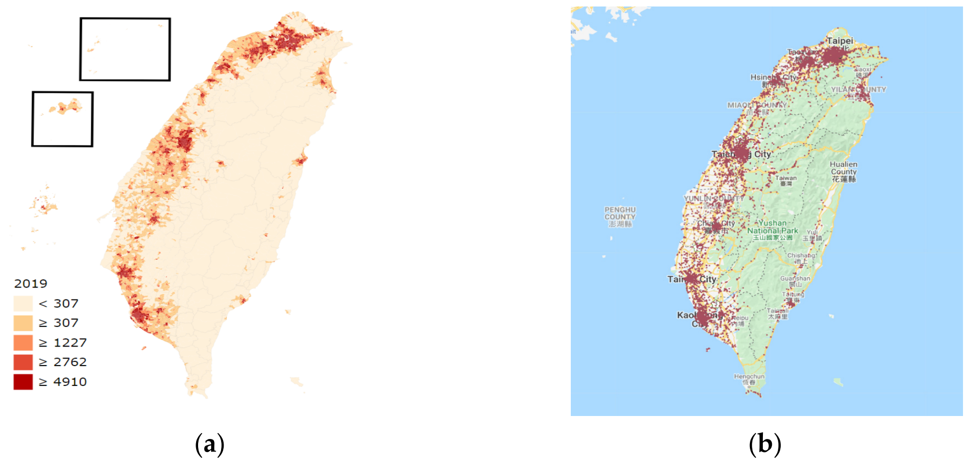


Figure 8. The simulated environment. (**a**) The distribution of residents per square kilometer by village; (**b**) the distribution of the targeted points for the simulation.

5.2. Demonstration

Figure 9 provides the corresponding polygons of City 1, City 2, and City 3 with 2054, 6080, and 7946 vertices. By applying the CoordRecongnit algorithm, the corresponding numbers of inner points could be calculated, which were 89,209, 156,928, and 146,626. Figures 7 and 10a,b, respectively, provide the distributions of the 10,000 random targets in City 1, City 2, and City 3. In addition, some information, such as the area and population of the cities, can be merged for various applications. These cities cover about 272, 2215, and 2595 km^2 and contain populations of about 2.65 million, 2.82 million, and 2.77 million people, respectively. In addition to the numbers of targets in specific areas, the average number per square kilometer or the average number of targets per million people in each area may be more helpful information. Table 2 provides the densities with geographical information. For the area density per square kilometer and the population density per million people, City 1 has 328 and 33,644, respectively, City 2 has 71 and 55,648, respectively, and City 3has 56 and 52,804, respectively. Regarding the area density, City 1 is the first, City 2 is the second, and City 3 is the third. In terms of population density, City 2 is the first, City 3 is the second, and City 1 is the third. If the targets are people who need to be managed during an epidemic, these data may be used to manage the epidemic in real time.

Table 2. Targets in cities with their geographical information. For a targeted city, the "target" represents the number of targets, "area" provides the area in square kilometers, "population" refers to the population in millions of people, "area density" refers to the number of targets per square kilometer, and "population density" refers to the number of targets per million people.

#	Target	Area	Population	Area Density	Population Density
City 1	89,209	272	2.65	328	33,664
City 2	156,928	2215	2.82	71	55,648
City 3	146,266	2592	2.77	56	52,804

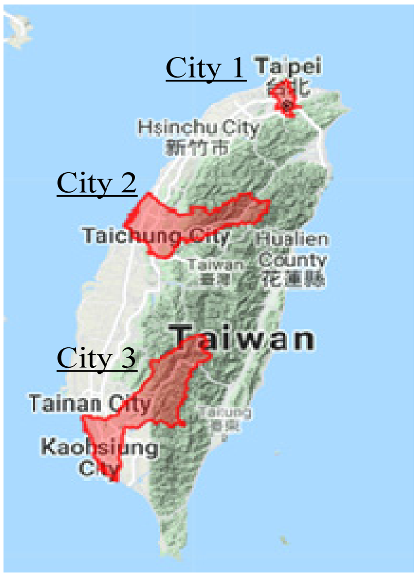


Figure 9. The polygons of targeted cities. The red areas are the corresponding polygons of City 1, City 2, and City 3.

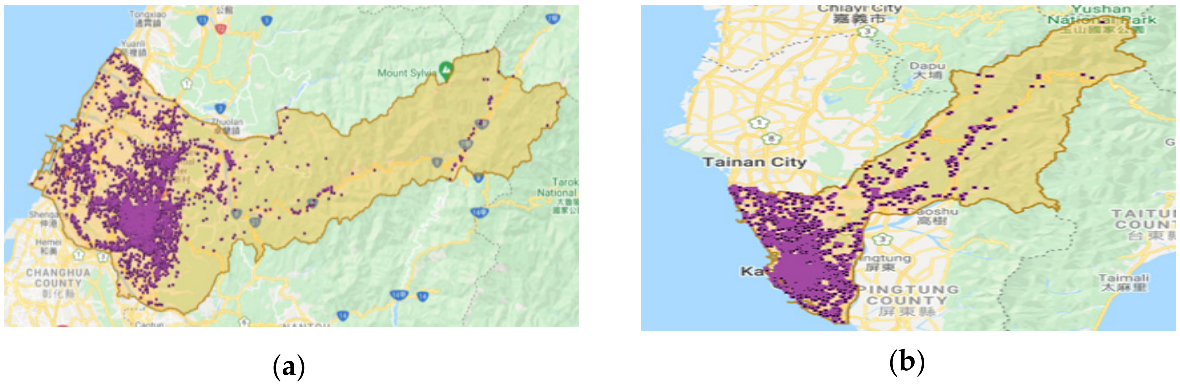


Figure 10. The distribution of targets. (**a**) The distribution of targets in City 2; (**b**) The distribution of targets in City 3.

In these three examples, the time consumed for each example was only tens of seconds. The results show that our method is capable of handling a large number of targets in a wide range.

6. Simulation Result

In this section, we compare the performance of O_PtInPy, L_Rect, c_, and our algorithm. Since a targeted area will be partitioned into a set of square areas in our algorithm, "sqr" is used to refer to our algorithm. Moreover, c_ uses a set of circles to cover a targeted area. This placement of circles starts from a seed point. The position of the seed point is selected as follows. First, a large rectangle that can cover a targeted area is defined. Then, this large rectangle is divided into 3 × 4 smaller rectangles of the same size. If the center points of the smaller rectangles are in the targeted area, these center points are used as seed points. Various seed points generate various numbers of small areas. Briefly, c_max, c_min, and c_avg are used to represent the maximum, minimum, and average numbers of small areas, respectively. The points retrieved from the database are simply called candidate points. Candidate points are divided into inner points and outer points. For a candidate point, it may or may not be necessary to execute PtInPy to confirm if this point is an inner point. The points that need to be confirmed through PtInPy are termed confirmed points.

For convenience, n_{cand}, n_{inner}, n_{outer}, n_{conf}, and n_{area} are used to refer to the numbers of candidate points, inner points, outer points, confirmed points, and subareas, respectively.

Table 3 compares n_{cand}, n_{conf}, and n_{cand}/n_{inner} for O_PtInPy and L_Rect in City 1, City 2, and City 3. For O_PtInPy, in City 1, City 2, and City 3, the values of n_{cand} were all 1,112,188. Since each candidate point needed to be confirmed by using PtInPy, the value of n_{conf} was also 1,112,188. In City 1, 89,209 target points were finally obtained, but a total (n_{cand}) of 1,112,188 candidate points were retrieved from the database. The ratio of n_{cand} to n_{inner} was about 12.47. The ratios for City 2 and City 3 were about 7.09 and 7.60. For L_Rect, in City 1, City 2, and City 3, the values of n_{cand} were, 13,869, 19,057, and 256,725, respectively. Since each candidate point also needed to be confirmed by using PtInPy, n_{conf} was equal to n_{cand}. Accordingly, the rates were about 1.55, 1.21, and 1.76, respectively.

Table 3. Comparison of candidate points, confirmation points, and inner points. The rate is n_{cand} divided by n_{inner}.

City	O_PtInPy			L_Rect		
	n_{cand}	n_{conf}	Rate	n_{cand}	n_{conf}	Rate
City 1	1,112,188	1,112,188	12.47	138,678	138,678	1.55
City 2	1,112,188	1,112,188	7.09	190,507	190,507	1.21
City 3	1,112,188	1,112,188	7.60	256,725	256,725	1.76

Next, the performance of sqr, c_max, c_min, and c_avg in City 1 was compared. Moreover, for c_, the radiuses of the small areas were 1/16, 1/32, 1/64, 1/128, and 1/256 based on one degree of latitude and longitude. We let both methods configure circles and squares with the same area. If the radius of a circle is r, the side length of the corresponding square is sl, as shown in (4), where π is the ratio of the circumference of a circle to its diameter.

$$sl := \sqrt{\pi \times r^2} \tag{4}$$

6.1. Subareas

A set of subareas in c_ and sqr was defined for a targeted area. Based on the subareas having the same area, the numbers (n_{area}) of subareas generated by c_ and sqr were compared as follows. Table 4 provides the values of n_{area}. When the radius was 1/16 to 1/256, n_{area} of c_max ranged from 10 to 707, n_{area} of c_min ranged from 7 to 694, and n_{area} of c_avg ranged from about 8 to 702. Based on the same radius, n_{area} of sqr ranged from 4 to 552, which was less than that of c_max, c_min, and c_avg. When the radius was 1/16, compared with c_max and sqr, sqr reduced the number to 6 (=10 − 4) subareas, and the reduction rate was 60% (=6/10). Figure 11 provides the subarea reduction rates for c_max and sqr, c_min and sqr, and c_avg and sqr, which are represented by c_max#sqr, c_min#sqr, and c_avg#sqr, respectively. When the radius was 1/16, the reduction rates were about 60%, 43%, and 51%. When the radius was reduced to 1/256, the reduction rates were about 22%, 20%, and 21%, respectively. Overall, the subarea reduction rate was between 60% and 20%.

Table 4. The number of subareas.

Radius	c_max	c_min	c_avg	sqr
1/16	10	7	8	4
1/32	22	19	20	13
1/64	63	57	60	41
1/128	199	192	196	146
1/256	707	694	702	552

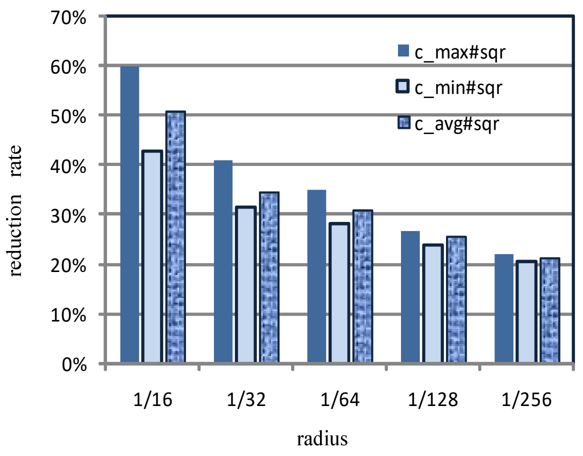


Figure 11. The subarea reduction rates.

6.2. Candidate Points and Outer Points

The number of candidate points n_{cand} and the number of outer points n_{outer} are discussed in the following. Table 5 provides the values of n_{cand} for each algorithm used in the comparison. When the radius went from 1/16 to 1/256, n_{cand} of c_max went from 179,853 to 109,639, n_{cand} of c_min went from 148,369 to 108,609, and n_{cand} of c_avg went from about 162,264 to 109,121. With the same radius, n_{cand} of sqr went from 123,374 to 89,841, which was less than the values for c_max, c_min, and c_avg. In c_max, when the radius is 1/16, n_{cand} was 179,853 with a value of n_{inner} of 89,209, and the ratio of n_{cand} to n_{inner} was about 2.02. When the radius was from 1/16 to 1/256, the ratio of c_max ranged from about 2.02 to 1.23, the ratio of c_min went from about 1.67 to 1.22, and the ratio of c_avg went from about 1.82 to 1.22. Based on the same radius, the ratio of sqr went from 1.38 to 1.01, which was less than the values for c_max, c_min, and c_avg.

Table 5. The number of candidate points.

Radius	c_max		c_min		c_avg		Sqr	
	n_{cand}	Rate	n_{cand}	Rate	n_{cand}	Rate	n_{cand}	Rate
1/16	179,853	2.02	148,369	1.66	162,264	1.82	123,374	1.38
1/32	144,919	1.62	122,179	1.37	134,579	1.51	102,322	1.15
1/64	121,457	1.36	115,708	1.30	119,440	1.34	93,721	1.05
1/128	113,037	1.27	110,719	1.24	111,776	1.25	91,204	1.02
1/256	109,639	1.23	108,609	1.22	109,121	1.22	89,841	1.01

The rate is n_{cand} divided by n_{inner}.

Table 6 provides the number of outer points n_{outer}. When the radius was 1/16, n_{outer} of c_max was 90,644, i.e., $n_{\mathrm{cand}} - n_{\mathrm{inner}}$. When the radius went from 1/16 to 1/256, n_{outer} of c_max went from 90,644 to 20,430, n_{outer} of c_min went from 59,160 to 19,400, n_{outer} of c_avg went from about 73,055 to 19,912, and n_{outer} of sqr went from 34,165 to 632. When the radius was 1/16, n_{outer} of c_max was 90,644, and n_{outer} of sqr was 34,165. The number of outer points was reduced to 56,479, and the reduction rate was about 62.3% (=56,479/90,644). Figure 12 provides the reduction rates of c_max#sqr, c_min#sqr, and c_avg#sqr. When the radius was 1/16, the reduction rates of c_max#sqr, c_min#sqr, and c_avg#sqr were about 62.3%, 25.6%, and 53.2%. When the radius was reduced to 1/256, the reduction rates of c_max#sqr, c_min#sqr, and c_avg#sqr were about 96.9%, 91.9%, and 96.8%. Accordingly, sqr resulted in a reduction rate of the outer points between 25.6% and 96.9% with respect to c_max, c_min, and c_avg.

Table 6. The numbers of outer points.

Radius	c_max	c_min	c_avg	Sqr
1/16	90,644	59,160	73,055	34,165
1/32	55,710	32,970	45,370	13,113
1/64	32,248	26,499	30,231	4512
1/128	23,828	21,510	22,567	1995
1/256	20,430	19,400	19,912	632

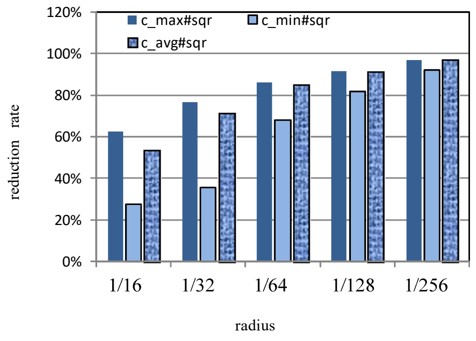


Figure 12. The reduction rate of outer points.

6.3. Confirmed Points

The number of confirmed points n_{conf} is discussed in the following. Table 7 provides the values of n_{conf}. When the radius went from 1/16 to 1/256, n_{conf} of c_max went from 142,054 to 4953, n_{conf} of c_min went from 107,065 to 2869, and n_{conf} of c_avg went from about 121,540 to 3466. With the same radius, the n_{conf} of sqr went from 56,237 to 2054, which was less than the values for c_max, c_min, and c_avg. When the radius was 1/16, n_{conf} of c_max was 142,054, and n_{conf} of sqr was 56,237. This was a reduction of 85,817 points, and the reduction rate of c_max#sqr was about 60.4% (=85,817/142,054). Figure 13 provides the reduction rates. When the radius was 1/16, the reduction rates of c_max#sqr, c_min#sqr, and c_avg#sqr were about 60.4%, 47.5%, and 53.7%. When the radius was reduced to 1/256, the reduction rates of c_max#sqr, c_min#sqr, and c_avg#sqr were about 58.5%, 28.4%, and 40.7%. Moreover, sqr resulted in a reduction rate of the confirmed points between 28.4% and 60.4%.

Table 7. The confirmed points.

Radius	c_max	c_min	c_avg	sqr
1/16	142,054	107,065	121,540	56,237
1/32	75,034	56,775	65,475	23,126
1/64	30,428	24,948	28,017	16,520
1/128	12,478	8965	11,007	5081
1/256	4953	2869	3466	2054

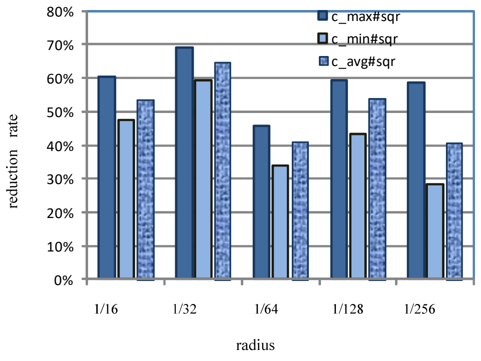


Figure 13. The reduction rates of confirmation points.

6.4. Discussion

For a given targeted area, the performance of O_PtInPy, L_Rect, c_, and sqr in identifying inner points was compared. For City 1, O_PtInPy generated 12.47 times the candidate points with respect to the inner points, and L_Rect generated 1.55 times the candidate points with respect to the inner points. The values for c_ and sqr in City 1 ranged from 2.02 to 1.22 and from 1.38 to 1.01. For c_ and sqr, different subarea plans generated different results. In addition to database access, the amount of calculation was also one of the metrics used to evaluate these methods. The number of confirmed points generated by applying a method affected the computation required. For O_PtInPy, the confirmed points contained all targets. Therefore, the amount of calculation needed for O_PtInPy was very large. For L_PtInPy, the targets within the range of the rectangle were confirmed points. Compared with O_PtInPy, the amount of calculation needed by L_PtInPy was reduced, but it was also very large. Taking City 1 as an example, O_PtInPy generated 1,112,188 confirmed points, and L_PtInPy generated 138,678 confirmed points, which was about 12.45% of the number of confirmed points for O_PtInPy.

Then, the c_ and sqr methods were individually compared. For the subareas, sqr reduced the number by 60% to 20% with respect to c_. This shows that sqr is better than c_ in the allocation of subareas. For the outer points, sqr reduced the number by 96.9% to 25.6% with respect to c_. When the area of the subareas was smaller, the number of confirmed points would be closer to the number of inner points, and the ratio of candidate points to inner points would be closer to 1. For example, when the radius was 1/256, the ratio of sqr was about 1.01, and the ratio of c_ was between 1.23 and 1.22. When the area of the subareas was smaller, the value of sqr would be closer to 1, but the value of c_ could only stay at about 1.20. The reason is that the subareas of c_ overlapped with each other. The points within the overlapped area were repeatedly retrieved from the database. The subareas of c_ were in a circular area. However, within this circular area, only a regular hexagonal area would not overlap with other subareas, and the overlap area was about 17.3%. So, even with very efficient database systems that can support the retrieval of data within smaller partitions, c_ has limitations that sqr does not.

In c_, different seed points also affected its performance, but in sqr, there was no such shortcoming. In addition, the range of the subareas was also a problem. An improper range may cause c_ to generate more candidate points than L_Rect. Accordingly, the method could determine a reasonable length (*sl*) for the side of a square. Algorithm 7 is a SqrSide algorithm, the inputs of which are a polygon *P*, a point set *P*, and a number *n*db. This algorithm first calculates a rectangle R with a range that can cover polygon *P* through RectOfPy. Next, it calculates the area of this rectangle *R*. Then, it calculates the number *ta* of inner points of rectangle *R* through an SQL statement. t_1 and t_2 are continuously calculated, where t_1 is the expected number of squares and t_2 is the expected area of a square. Finally, the algorithm calculates the length (*sl*) of a square's side by using the power function pow(), i.e., sl^2= t$_2$. Notably, in line 4, the select statement is similar to the MySQL statement for obtaining the number of records that meet the query conditions.

Algorithm 7. SqrSide(polygon P, point set G, number n_{db})

```
1. $R$ := RectOfPy($P$);
2. $x_{min}$ := $R$[0]; $x_{max}$ := $R$[1]; ymin := R[2]; ymax := R[3];
3. $a := (x_{max} - x_{min}) \times (y_{max} - y_{min})$;
4. $t_a$ := {SELECT count(*) FROM '$G$' WHERE 'x' $\geq x_{min}$ AND 'x' $\leq x_{max}$ AND 'y' $\geq y_{min}$ AND
   'y' $\leq y_{max}$};
5. $t_1 := t_a/n_{\text{db}}$;
6. $t_2 := a/t_1$;
7. $sl$ := pow($t_2$, 0.5);
8. return ($sl$);
```

Therefore, based on the above discussion, the performance of sqr is far superior to that of the other methods that were discussed in this work. Moreover, it is an efficient GNSS coordinate recognition algorithm.

7. Conclusions

Many infectious diseases are highly contagious and seriously affect human health, economic activities, education, sports, and leisure. As an epidemic spreads, tracking targets within specific administrative areas is an effective option for slowing the spread. In this study, we presented an efficient GNSS coordinate recognition algorithm for epidemic management. It can be used to efficiently track targets within specific administrative areas. In the future, we will consider the limitations of using the gathering of personal information and continue to discuss related issues.

Author Contributions: Conceptualization, J.-S.C. and R.-W.H.; methodology, J.-S.C., C.-M.K. and R.-W.H.; software, C.-M.K.; validation, J.-S.C. and R.-W.H.; formal analysis, J.-S.C. and R.-W.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nath, T.C.; Bhuiyan, M.J.U.; Mamun, M.A.; Datta, R.; Chowdhury, S.K.; Hossain, M.; Alam, M.S. Common infectious diseases of goats in Chittagong district of Bangladesh. *Int. J. Sci. Res. Agric. Sci.* **2014**, *1*, 43–49. [CrossRef]
2. Edmond, M. Isolation. *Infect. Control. Hosp. Epidemiol.* **1997**, *18*, 58–64. [CrossRef] [PubMed]
3. Li, T. Diagnosis and clinical management of severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) infection: An operational recommendation of Peking Union Medical College Hospital (V2. 0) working group of 2019 novel coronavirus, Peking union medical college hospital. *Emerg. Microbes Infect.* **2020**, *9*, 582–585. [PubMed]
4. Zhu, N.; Marais, J.; Bétaille, D.; Berbineau, M. GNSS position integrity in urban environments: A review of literature. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 2762–2778. [CrossRef]
5. Dow, J.M.; Neilan, R.E.; Rizos, C. The international GNSS service in a changing landscape of global navigation satellite systems. *J. Geod.* **2009**, *83*, 191–198. [CrossRef]
6. Zhbankov, G.A.; Danilkin, N.P.; Maltseva, O.A. Influence of the ionosphere on the accuracy of the satellite navigation system. *Acta Astronaut.* **2022**, *190*, 194–201. [CrossRef]
7. Gupta, R.; Jay, D.; Jain, R. Geographic information systems for the study and control of infectious diseases. In Proceedings of the Map India Conference, New Delhi, India, 28–31 January 2003.
8. Awange, J.L. *Environmental Monitoring Using Gnss: Global Navigation Satellite Systems*; Springer: Berlin/Heidelberg, Germany, 2012.
9. Awange, J. *Gnss Environmental Sensing*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10, pp. 978–983.
10. Cahyadi, M.N.; Susanto, L.O.F.; Rokhmana, C.A.; Sulistiawan, S.S.; Waloejo, C.S.; Raharjo, A.B.; Atok, M. Telemedicine technology application for COVID-19 patient tracing using smartphone gnss. *Int. J. Geoinformatics* **2022**, *18*, 103–117.
11. Chew, C.; Small, E. Estimating inundation extent using cygnss data: A conceptual modeling study. *Remote Sens. Environ.* **2020**, *246*, 111869. [CrossRef]
12. Gupta, R.; Bedi, M.; Goyal, P.; Wadhera, S.; Verma, V. Analysis of COVID-19 tracking tool in India: Case study of Aarogya Setu mobile application. *Digit. Gov. Res. Pract.* **2020**, *1*, 1–8. [CrossRef]
13. Hormann, K.; Agathos, A. The point in polygon problem for arbitrary polygons. *Comput. Geom.* **2001**, *20*, 131–144. [CrossRef]
14. Haines, E. Point in polygon strategies. *Graph. Gems* **1994**, *4*, 24–46.
15. Taylor, G. Point in polygon test. *Surv. Rev.* **1994**, *32*, 479–484. [CrossRef]
16. Dimri, S.C.; Tiwari, U.K.; Ram, M. An efficient algorithm to clip a 2D-polygon against a rectangular clip window. *Appl. Math.-A J. Chin. Univ.* **2022**, *37*, 147–158. [CrossRef]
17. Kumar, G.N.; Bangi, M. An extension to winding number and point-in-polygon algorithm. *IFAC-Pap.* **2018**, *51*, 548–553. [CrossRef]
18. Moscato, M.M.; Titolo, L.; Feliú, M.A.; Munoz, C.A. Provably correct floating-point implementation of a point-in-polygon algorithm. In *International Symposium on Formal Methods*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 21–37.
19. Weekly Epidemiological Update on COVID-19—25 May 2022. Available online: https://www.who.int/publications/m/item/weekly-epidemiological-update-on-COVID-19---25-may-2022 (accessed on 12 July 2022).
20. Chang, S.C.; Huang, H.Y.; Huang, Y.F.; Yang, C.Y.; Hsu, C.Y.; Chen, J.S. An Efficient Geographical Place Mining Strategy for Social Networking Services. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Yilan, Taiwan, 20–22 May 2019; pp. 1–2.

21. Lin, C.B.; Hung, R.W.; Hsu, C.Y.; Chen, J.S. A GNSS-based crowd-sensing strategy for specific geographical areas. *Sensors* **2020**, *20*, 4171. [CrossRef] [PubMed]
22. Kim, H.-S. What drives you to check in on Facebook? Motivations, privacy concerns, and mobile phone involvement for location-based information sharing. *Comput. Hum. Behav.* **2016**, *54*, 397–406. [CrossRef]
23. Huang, Y.-F.; Chen, J.-S.; Lin, C.-B. A specific Targeted-Place Mining Method for a Famous Social Network: Take Wang-ye Worship in Taiwan for example. In Proceedings of the 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN), Yichang, China, 16–8 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 263–266.
24. Chen, J.S.; Lin, C.B.; Yang, C.Y.; Huang, Y.F. An efficient Facebook place information extraction strategy. In *International Symposium on Mobile Internet Security*; Springer: Singapore, 2017; pp. 131–139.
25. Chen, J.S.; Huang, Y.F.; Yang, C.Y.; Hung, R.W.; Lin, C.B. An efficient hot-spot analysis method for facebook places: Take Taipei city in Taiwan for example. In Proceedings of the IEEE International Conference on Applied System Invention (ICASI), Tokyo, Japan, 13–17 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 109–112.
26. Chen, J.S.; Hsu, C.Y.; Yang, C.Y.; Wei, C.C.; Ciang, H.G. A data mining method for Facebook social network: Take "new row mian (beef noodle)" in Taiwan for example. In Proceedings of the IEEE 8th International Conference on Awareness Science and Technology (iCAST), Taichung, Taiwan, 8–10 November 2017; pp. 165–169.
27. Beaulieu, A. *Learning SQL*, 2nd ed.; Treseler, M.E., Ed.; O'Reilly: Sebastopol, CA, USA, 2009; ISBN 978-0-596-52083-0.
28. MySQL. Sql Select Syntax. Available online: https://dev.mysql.com/doc/refman/8.0/en/select.html (accessed on 12 July 2022).
29. Antonio, F. Faster line segment intersection. In *Graphics Gems III (IBM Version)*; Morgan Kaufmann: Burlington, MA, USA, 1992; pp. 199–202.
30. Dubois, Paul, MySQL, Pearson Education. 2008. Available online: https://rog.asus.com/tw/compareresult/?productline=laptops&webpathname=GV301QE-0022A5900HS&id=136348 (accessed on 12 July 2022).
31. Naz, R.; Khan, M.N.A. Rapid applications development techniques: A critical review. *Int. J. Softw. Eng. Its Appl.* **2015**, *9*, 163–176. [CrossRef]
32. Demographics of Taiwa. Available online: https://en.wikipedia.org/wiki/Demographics_of_Taiwan (accessed on 12 July 2022).