Journal of Marine Science and Technology, Vol. 27, No. 1, pp. 35-45 (2019)
DOI: 10.6119/JMST.201902_27(1).0005

RECTANGLE-BASED APPROACHES FOR REPRESENTING FLOOD DATA IN SPATIAL INDICES*

Ya-Hui Chang, Wen-Han Lee, and Tai-Feng Ke

Key words: flood data, spatial index, R-tree.

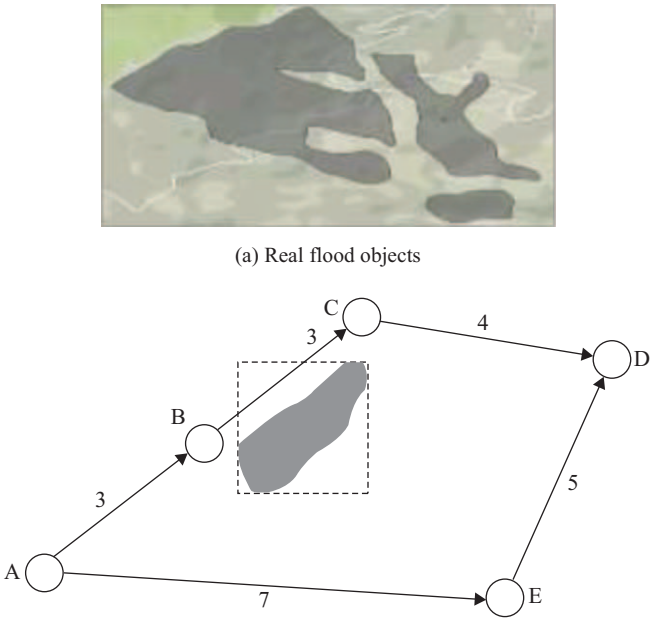
ABSTRACT

Appropriate representation of flood data in indices is crucial for efficient future querying. A common approach is to use minimum bounding rectangles to build R-tree indices rather than storing real complex objects; however, the results may be imprecise. Herein, to improve the performance of this approach, we discuss methods for producing corresponding rectangles. The first method involves generating a set of small fixed-sized squares, and the second one attempts to initially obtain a large rectangle. The third approach recursively partitions the space into four quadrants until the given size constraint is satisfied. Experimental results based on the real flood data of two representative cities demonstrate that the quadruple partitioning method can best approximate the original area using fewer cells and is thus the recommended approach.

I. INTRODUCTION

Natural disasters caused by heavy rain have become increasingly severe as a result of global warming. In certain regions bounded by the ocean or rivers, the situation is even worse because of the complex interaction of rain, rivers, waves, and tides. Researchers have investigated methods for creating flood-forecasting systems or quickly identifying flood zones using near-real time satellite data and other useful information. A more detailed discussion of relevant research is provided in Section 6.

After identifying locations that will be or are already flooded, the next critical step is to properly store spatial information in indices for efficient future querying. This is not an easy task because flood objects have irregular shapes that are dependent on the geographical characteristics. Although some types of database software have extended their type systems for users to represent complex geometrical objects using point sequences and some parameters, a large flood object may consist of up to hundreds of points or more. For example, three real flood objects that occurred in Keelung city are displayed in Fig. 1(a). The leftmost one actually consists of 655 points. Such large amounts of point data require a lot of storage space and also make the future spatial operation (e.g., computing intersected areas) difficult to process.



(a) Real flood objects

(b) A minimum bounding rectangle

Fig. 1. Examples of real flood objects and minimum bounding rectangles.

Therefore, researchers have proposed the approximation of complex spatial objects using minimum bounding rectangles (MBRs) because of their simplicity. Many spatial indices, such as the R-tree index designed by Guttman (1984) and the R*-tree index proposed by Beckmann et al. (1990) have utilized the MBRs of spatial objects as the keys of index nodes to enable quick performance of various spatial operations. However, MBRs may cause inaccurate answers in some applications. An example is

Paper submitted 10/13/18; revised 11/28/18; accepted 01/17/19. Author for correspondence: Ya-Hui Chang (e-mail: yahui@ntou.edu.tw).
[1] Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan, R.O.C.
** This work was partially supported by the Ministry of Science and Technology under Contract No. MOST 107-2221-E-019-043-.*

provided using the road network displayed in Fig. 1(b), in which the gray object represents a flood object and its MBR is depicted by dashed lines. In the event that a user wishes to depart from node A and arrive at node D, the system erroneously judges that the road is flooded and impassable because the dashed rectangle intersects the road (B, C). If the path-planning system computes the shortest path based on this information, it will identify the path (A, E, D), which is longer than the *real* shortest unflooded path (A, B, C, D).

Some researchers, such as Freeman and Shapira (1975) and Brinkhoff et al. (1994), have discussed methods for reducing the number of misjudgments by using different forms of approximation, but identifying optimal methods for flood objects is difficult because of their irregular shapes. Moreover, more accurate approximations tend to involve complex shapes, which are difficult to use directly in building spatial indices for pruning impossible matches. Therefore, in this paper, we advocate the use of rectangles to retain the applicability of the most commonly used R-tree index. However, rather than directly using the simple MBR, we attempted to produce a set of smaller cells to more accurately approximate the original flooded areas and reduce the occurrence of misjudgment.

In this paper, we propose three methods for producing flooded cells. The baseline approach is to partition the MBR of each flood object into a set of squares with a given side length. This method is easily implemented but increases the difficulty of determining the proper side length. A small side length can reduce the degree of misjudgment, as depicted in Fig. 1(b), but may produce many cells; this could result in the creation of a big R-tree and slow subsequent spatial join operations. Therefore, we designed the Minimum Intersection Rectangle (MIR) approach, which attempts to first obtain a "big" flooded cell before running the baseline approach on the remaining space; the aim of this method is to reduce the total number of flooded cells being produced. The final approach is the most flexible one in terms of the size of generated flooded cells. It first partitions the MBR into four quadrants. A component quadrant is returned as a flooded cell by itself if most of its space is flooded; otherwise, it is partitioned recursively. The primary difficulty lies in how to appropriately process the point sequence representing a flood object to ensure that each sub-sequence within a quadrant continues to form a reasonable shape after partitioning. We have derived the properties of a valid sub-sequence and used them as the basis of the partitioning algorithm. The contributions of this paper are summarized as follows:

(1) We investigate the problem of representing flood data in indices. We advocate the idea of generating smaller rectangles by partitioning the MBRs of flood objects to reduce the occurrence of misjudgment while maintaining the easy construction of spatial indices.
(2) We designed three partitioning algorithms. Algorithm FS (FixedSized) produces a set of squares with a specified side length. Algorithm MIR creates a large rectangle and a set of smaller rectangles. Algorithm Quad (Quadruple) recursively partitions the space into four quadrants if necessary and may generate flooded cells with different sizes. To address the problem of appropriately processing the point sequence representing a flood object to successfully perform the recursive procedure, we investigated the characteristics of reasonable shapes possessed by flood objects and designed the algorithm accordingly.
(3) We have implemented the proposed three approaches and conducted extensive empirical studies using the real flood data of two representative cities. The primary metrics used to compare the flooded cells returned with each method are the area covered by these cells and the number of produced cells. The experimental results revealed that the Quad approach was optimal for approximating the real flooded area with fewer cells.

The remainder of this paper is organized as follows. In Section 2, we formally define the problem to solve and discuss Algorithm FS. In Section 3 and Section 4, we present Algorithm MIR and Algorithm Quad, respectively. Finally, experimental results are provided in Section 5, related works are discussed in Section 6, and conclusions are provided in Section 7.

II. PROBLEM DESCRIPTION AND THE FIXED-SIZE APPROACH

We first formally define the terms used in this paper and the problem to be solved. We also describe the fixed-size approach in this section. Although the data being processed are geographic, we assume the Cartesian coordinate system for simplicity, where the x axis represents the longitude values and the y axis represents the latitude values.

Definition 1 A point p refers to a specific geographic location and is identified by a pair of longitude and latitude values, which will be referred as x and y values throughout the remainder of this manuscript.

Definition 2 A *flood object* is denoted by the variable f and is represented by a sequence of points $(p_1, \ldots, p_n)$, which forms a simply connected domain[1] and the area inside of which is flooded.

Definition 3 A rectangle r is identified by its lower left point $(\min_x, \min_y)$ and upper right point $(\max_x, \max_y)$ and denoted by the quadruple $[\min_x, \min_y, \max_x, \max_y]$. The sides of the rectangle are parallel to the x and y axes, respectively.

[1] A domain D is considered to be simply connected if every closed curve in D can be continuously shrunk to any point in D without leaving D (Kreyszig (2011)). Note that most flood objects satisfy this constraint because of geographical characteristics, as indicated in Fig. 1(a). If a flood object is not simply connected, we will decompose it into several objects, where each one satisfies such constraints in the preprocess stage.

```
Algorithm FS
Input: f //a flood object, e //the fixed side length of a cell
Output: OutputList
1: initialize OutputList and TempList as null;
2: set r as MBR(f) with the quadruple [min_x, min_y, max_x, max_y];
3: if r.size < e^2 then
4:     return OutputList.add(r).
5: end if
6: lng ← m2Lng(e); lat ← m2Lat(e);
7: set p as the point located at (min_x, min_y);
8: while (p.x + lng < max_x) and (p.y + lat < max_y) do
9:     add the cell located at p into TempList;
10:    advance p properly;
11: end while
12: partition the rightmost and uppermost remaining area;
13: for each square c in TempList do
14:    if intersects(f, c) then
15:        OutputList.add(c);
16:    end if
17: end for
18: return OutputList.
```

Fig. 2. Algorithm FS.

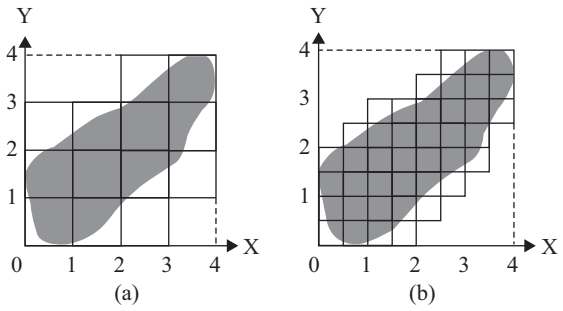


Fig. 3. Examples of fixed-size flooded cells.

Definition 4 A *flooded cell*, denoted by c, is a rectangle that has been constructed based on a flood object and all or part of the area inside the cell c is flooded.

In this paper, we discuss the problem of producing flooded cells for constructing rectangle-based spatial indices. As discussed in Section 1, the simplest form is the MBR of a flood object, but it might cover too much space and yield imprecise results. Therefore, our first approach, Algorithm FS, was to divide the MBR into many smaller fixed-sized cells. As indicated in Fig. 2, we first identified the MBR of the given flood object f and then directly returned the MBR as the only flooded cell if it was already relatively small (L1-5). Otherwise, we partitioned the MBR into a set of squares with the side length e. Because the parameter e was provided in meters, it was required first to be converted to the equivalent longitude and latitude values (L6), and the partitioning process began from the lower left point of the MBR to the upper right point (L7-12). After obtaining the set of squares, the function *intersection* in L14 ensured that only those squares overlapping the given flood object f were returned.

An example is provided in Fig. 3(a). The shaded object represents a flood object. Its MBR is depicted by dashed lines, and the flooded cells have solid edges. The set of the 13 smaller flooded cells clearly offers a better approximation of the flood object than the MBR in terms of their respective unused space. This measurement was formally defined by the *coverage ratio* (*cr*), which compares the total sizes of constructed flooded cells c_i with that of the original flood object f as follows:

$$cr = \frac{\sum c_i.size}{f.size} \tag{1}$$

Based on our method of producing flooded cells, the *cr* value is always larger than one. Moreover, if the *cr* value is smaller and near to one, the total sizes of flooded cells will be similar to that of the original flood object and the inaccurate situation depicted in Fig. 1(b) is most likely to be avoided. Although smaller flooded cells yield favorable *cr* values, an extremely small e value is impractical because more cells will be produced. Compare the flooded cells displayed in Figs. 3(a) and (b) respectively, in which those in (a) have the side length "1" and those in (b) have the side length "0.5". We can see that their *cr* values are 13:(41/4) and their counts are 13:41. To determine the side length solely based on the *cr* value, we should set it as "0.5"; however, generating many flooded cells poses two disadvantages. First, the intersection operation between two spatial objects, as specified in L14 of Fig. 2, is relatively costly compared with other operations. Second, more cells would expand the R-tree index, which would negatively affect future querying efficiency. Because an effective method for determining an appropriate e value that achieves a balance between avoiding misjudgment and demonstrating adequate querying efficiency was not obvious, we designed other partitioning methods and compared their performance, as detailed in the following sections.

III. MINIMUM INTERSECTION RECTANGLES

By examining the flood object in Fig. 3(b), we can discern that its center part is partitioned into many cells, which seems unnecessary. In this section, we propose another partitioning method that involves creating a "big" flooded cell around the center part first with the aim of reducing the total number of flooded cells. Rather than identifying an optimal method that requires complicated computation, we designed a heuristic approach based on the *intersection points* of the shape. Specifically, given a flood object and its MBR, we first identified the center of the MBR o and then assigned the coordinate axes x and y accordingly. For easy reference, we sometimes refer to a specific axis by the directions *east* (E), *north* (N), *west* (W), and *south* (S), as indicated in Fig. 4(a). The intersection points and the particular specified flooded cells are then defined as follows:

Definition 5 An *intersection point*, θ, is a point where the flood object intersects the x or y axis. We use the superscript d, such as θ^d, to represent the direction of the intersection point. The intersection point may have already existed in the point sequence representing the flood object or be computed through

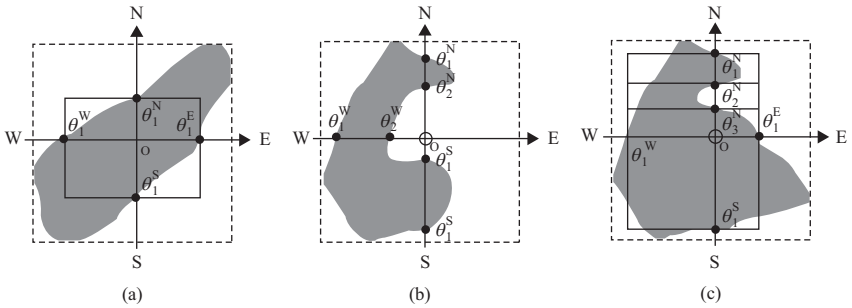


Fig. 4. Examples of intersection points between a flood object and axes.

```
Algorithm MIR
Input: f //a flood object, e //the minimum side length of a cell
Output: OutputList
 1: initialize OutputList and TempList as Null;
 2: set r as MBR(f) with the quadruple [min_x, min_y, max_x, max_y];
 3: if(r.size < e^2) then
 4:     return OutputList.add(r).
 5: end if
 6: if not within(r.center, f) then
 7:     return Outputlist.add(FS(f)).
 8: end if
 9: for each point s[i] representing f do
10:   if InAxis(s[i]) then
11:       d ← GetDirection(s[i], r.center);
12:       UpdateAxisPoint(s[i], d);
13:   else if Cross(s[i], s[i+1]) then
14:       θ ← GetInterPoint(s[i], s[i+1], r.center);
15:       d ← GetDirection(θ, r.center);
16:       UpdateAxisPoint(θ, d);
17:   end if
18: end for
19: MIR ← SetMIR(); OutputList.add(MIR);
20: TempList ← FS(r-MIR, e);
21: for each cell c in TempList do
22:   if intersects(f, c) then
23:       OutputList.add(c);
24:   end if
25: end for
26: return OutputList.
```

Fig. 5. Algorithm MIR.

interpolation.

Definition 6 Consider a flood object *f*, the MBR center of which *o* is located inside *f*. For the four directions *east*, *north*, *west*, and *south*, the intersection points nearest to *o* were obtained. The rectangle with the four specified intersection points on the sides was termed the MIR of *f*.

First note that we require the center of the MBR to be located inside the flood object. Otherwise, as depicted in Fig. 4(b), the center part of the flood object will not cover much of the flooded space to meet the motivation, so there is no need to use this method. Also note that there is a *nearest* requirement for determining among multiple intersection points in the same direction. As shown in Fig. 4(c), there are three intersection points in the north. We adopt the conservative approach and choose the one nearest to the center, i.e., θ_3^N, so that a higher percentage of the space covered by an MIR is flooded.

The corresponding MIR Algorithm is displayed in Fig. 5. As discussed, if the center of the MBR of the object was located outside of the shape, using this partitioning method served no advantage and we directly invoked Algorithm FS to partition this flood object (L6-8). Otherwise, we examined the *s*[*i*] in the point sequence representing the flood object (L9-18). First, we determined whether the point was located directly on the *x* or *y* axis. If so, we identified the direction of the axis and recorded the intersection point nearest to the center in that axis observed thus far (L10-12). Otherwise, if two adjacent points crossed a certain axis, we computed the intersection point through interpolation and similarly updated and recorded the nearest one (L13-16). After concluding an examination of the entire point sequence, we constructed the MIR based on the four nearest intersection points and returned it as a flooded cell (L19). Finally, we partitioned the remaining space between the MBR and MIR using Algorithm FS, and only cells that intersected the original flood object were returned (L20-25).

IV. QUADRUPLE PARTITIONING

The MIR method may be suitable for those flood objects that are "fat" in the center. However, for certain thin-shaped objects that are commonly seen in the flooded regions along river banks, the MIR may be too small or too big to be useful. Therefore, we propose a more flexible method in which the sizes of constructed flooded cells could differ. The idea was to divide the MBR of a flood object into four quadrants. If a high percentage of the space covered by a quadrant was flooded, we directly returned the quadrant as a flooded cell. Otherwise, we performed the partitioning procedure recursively on the quadrant until it satisfied the threshold constraint. Fig. 6(a) presents an example, in which the four quadrants are denoted as Q_1, Q_2, Q_3, and Q_4. Supposing that the threshold constraint was that more than half of the quadrant must be flooded, quadrants Q_1 and Q_3 clearly meet the requirement and can be flooded cells by themselves. In contrast, Q_2 and Q_4 must be further partitioned. We can see that the generated flooded cells have varying sizes.

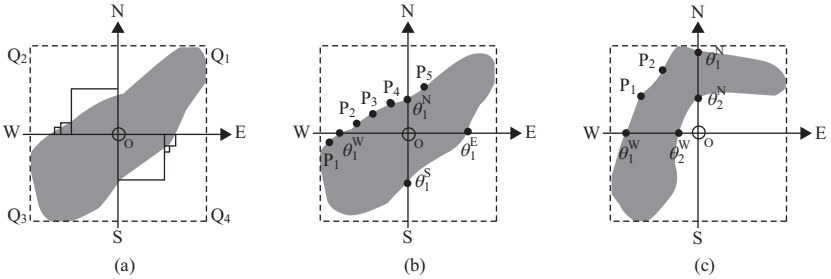


Fig. 6. Examples of the quadruple partitioning and formulating proper point sequences for sub-flood objects.

Although the procedure for partitioning seems similar to that for constructing a Quadtree in Finkel and Bentley (1974), processing the flood data represented as point sequences is nontrivial. Using the flood object depicted in Fig. 6(b) as an example and supposing that part of the point sequence representing the original flood object is (p_1, p_2, p_3, p_4, p_5), we must further partition Q_2 because the flooded area within the Q_2 quadrant does not exceed the percentage threshold. However, we cannot directly use the point sequence (p_2, p_3, p_4) to represent the sub-flood object in this quadrant because the resultant polygon significantly differs from the shaded area. A more appropriate method is to form the point sequence (θ_1^W, p_2, p_3, p_4, θ_1^N, o) for the sub-flood object, where points θ_1^W and θ_1^N are the intersection points on the borders of Q_2, as indicated in Definition 6, and o is the center of the MBR.

Further note that we cannot directly insert the center of the MBR into the sub-point sequence in all cases. Consider Fig. 6(c). The center o is not located inside the original flood object; therefore, it is not on the boundary of any sub-object and should not appear in any sub-point sequence. As an example, the appropriate point sequence for the sub-flood object in Q_2 should be (θ_1^W, p_1, p_2, θ_1^N, θ_2^N, θ_2^W), without the center o. As demonstrated by the two aforementioned representative examples, we derive the following property to enable the sub-point sequence to more accurately approximate the real flooded area based on where the center of the MBR is located:

Property 1 For a flood object f, if the center of the MBR o is within the shape, the sub-point sequence for each quadrant Q should consist of the intersection points on Q*'s* borders and o. Otherwise, the sub-point sequence should only consist of the intersection points on the borders.

We also hope for the sub-point sequence of each quadrant to continue to form a simply connected domain as the original flood object does. This condition facilitates future spatial calculation, such as computing the size of the sub-flood object. Therefore, we directly ceased the recursive partitioning procedure when we discovered that not just any sub-object would satisfy this property. The main challenge lies in how to perform the detection task easily, and we propose to use *the number of intersection points* due to its direct relationship with the shapes of sub-objects. In the following, we first provide some definitions for assisting the detection process:

Definition 7 For a flood object f, the term IP(f, d) represents the number of f*'s* intersection points located on the axis in the d direction. Moreover, the term MIP(f) is the maximum value obtained from the four directions, and the term SIP(f) is their sum. These two terms are formally defined as follows:

$$MIP(f) = \max_{d \in \{E,N,W,S\}} IP(f,d) \tag{2}$$

$$SIP(f) = \sum_{d \in \{E,N,W,S\}} IP(f,d) \tag{3}$$

We now exhaustively present the possible shapes of sub-flood objects based on the values of MIP(f) and SIP(f). Note that the shape of the original flood object f forms a simply connected domain; therefore, we do not need to consider the shape with holes. More intersection points correspond to more complex shapes, and thus the following discussion starts from the smallest possible number of MIP(f):

Case I: MIP(f) = 1

In this case, the point sequence of a reasonable flood object f passes through the axis in each direction exactly once (i.e., the flood object covers all quadrants and also contains the center o). For the flood object f presented in Fig. 6(a), IP(f, E) = IP(f, N) = IP(f, W) = IP(f, S) = 1. Therefore, MIP(f) = 1 and SIP(f) = 4. We can see that the sub-flood objects for all quadrants form simply connected domains.

Case II: MIP(f) = 2

In this case, the point sequence of f either leaves and later re-enters a certain quadrant to create two intersection points on the border or does not appear in a certain quadrant and has no corresponding intersection points. That is, all IP(f, d)'s are even and either 0 or 2. Three sub-cases based on the value of SIP(f) are as follows:

1) SIP(f) = 4

For the flood object depicted in Fig. 6(c), IP(f, N) = IP(f, W) = 2 and IP(f, E) = IP(f, S) = 0. In this case, the shape of the ob-

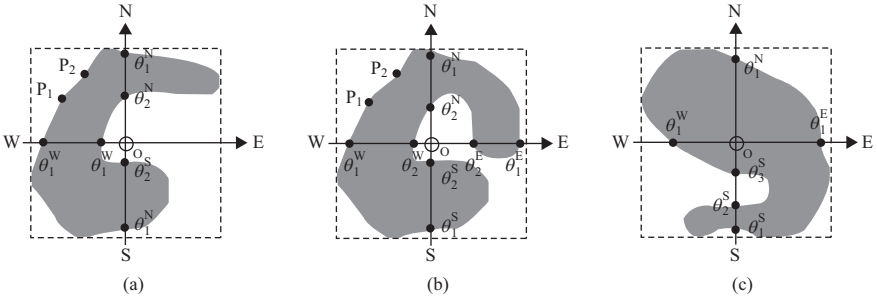


Fig. 7. Examples of sub-flood objects which form simply connected domains or not.

ject f does not contain the center o and does not cover quadrant Q_4. However, the shapes of the sub-objects located within quadrants Q_1-Q_3 still form simply connected domains.

2) SIP(f) = 6

In Fig. 7(a), IP(f, N) = IP(f, W) = IP(f, S) = 2 and IP(f, E) = 0. We can see that the original object also does not contain the center o. However, it covers all quadrants and every sub-flood object still forms simply connected domains.

3) SIP(f) = 8

In this case, IP(f, d) = 2 for all directions d. The only possible means is the point sequence separately entering a particular quadrant from two directions, as indicated by the two gray areas in the quadrant Q_4 of Fig. 7(b). The shape of the sub-object located within Q_4 will clearly not form a simply connected domain.

Case III: MIP(f) ≥ 3

Consider Fig. 7(c). IP(f, S) = 3 and IP(f, E) = IP(f, N) = IP(f, W) = 1. We can see that the point sequence of the original flood object leaves and re-enters Q_3 more than once. This cuts its shape off by the y axis, and the shape of the sub-object in Q_3 does not form a simply connected domain. We can easily see that a similar situation would occur with more intersection points.

From the aforementioned exhaustive cases, we can derive the following property for the shapes of sub-flood objects:

Theorem 1 All four sub-objects of a flood object f continue to form simply connected domains if one of the following two conditions are true: (1) the center of the MBR of f is located within f and MIP(f) = 1; (2) the center of the MBR of f is located outside of f, MIP(f) = 2, and two or three directions $d \in$ {E, N, W, S} establish IP(f, d) as equal to 2.

Proof

The first condition was directly obtained from Case I. The second condition was formed by combining the first and second sub-cases of Case II. □

Algorithm Quad was designed using the two aforementioned properties and is specified in Fig. 8. It is invoked in a recursive manner and the stopping condition is provided in L3. Basically, we directly returned the MBR as a flooded cell if its size was relatively small based on the threshold t_s or highly similar to that of the flood object according to the threshold t_r. If partitioning was required, we first set the value of the variable *InFlag* to signal whether the MBR's center is located inside the flood object (L6- 9), which is a crucial property, as stated in Observations 1-2.

```
Algorithm Quad
Input: f //a flood object, t_s //the size threshold, t_r //the ratio threshold
Output: OutputList
1: initialize OutputList as Null;
2: set r as MBR(f);
3: if (r.size < t_s) or (f.size/r.size > t_r) then
4:     return OutputList.add(r).
5: end if
6: InFlag ← FALSE;
7: if within(r.center, f) then
8:     InFlag ← TRUE;
9: end if
10: for each point s[i] representing f do
11:  d_1 ← GetDirection(s[i], r.center);
12:  if d_1 ∈ {E, N, W, S} then
13:      d_2 ← d_1;
14:      d_1 ← GetDirection(s[i−1], r.center);
15:      θ ← s[i];
16:  else if Cross(s[i], s[i+1]) then
17:      θ ← GetInterPoint(s[i], s[i+1], r.center);
18:      d_2 ← GetDirection(θ, r.center);
19:  else
20:      Seq[d_1].add(s[i]); continue;
21:  end if
22:  UpdateAxisCount(θ, d_2);
23:  if not SatisfyProperty(InFlag) then
24:      return OutputList.add(r).
25:  end if
26:  Seq[d_1].add(s[i]);
27:  if Cross(s[i], s[i+1]) then
28:      Seq[d_1].add(θ);
29:  end if
30:  if InFlag then
31:      Seq[d_1].add(r.center);
32:  end if
33:  d_3 ← GetDirection(s[i+1], r.center);
34:  Seq[d_3].add(θ);
35: end for
36: for each quadrant d do
37:  OutputList.add(Quad(Seq[d], t_s, t_r));
38: end for
39: return OutputList.
```

Fig. 8. Algorithm Quad.

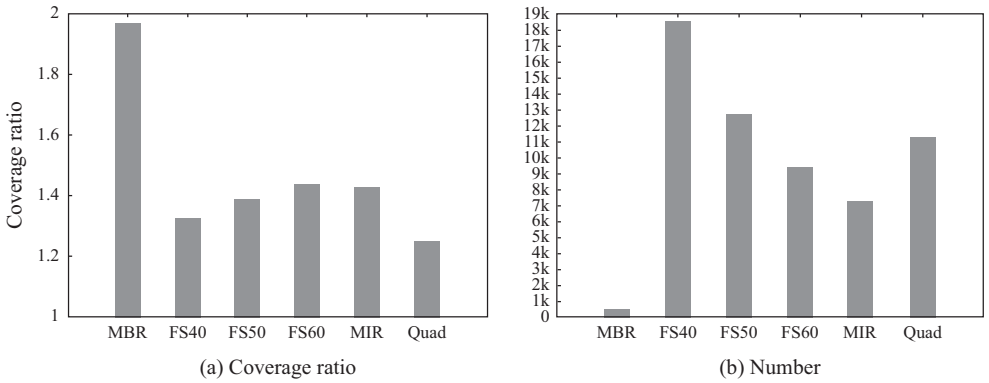


Fig. 9. Flooded cells based on the Keelung data set.

We then examined each point representing the flood object in sequence (L10-35). In most cases, we simply determined which quadrant within which this point was located (L11) and appended it to the corresponding sub-point sequence (L20). By contrast, when an intersection point was detected (L12-15) or obtained through interpolation (L16-18), we were required to increase the count for that particular axis (L22) and determine whether the property of the simply connected domain was still true based on Observation 2 (L23). If we expected a violation to occur, we directly returned the MBR as the flooded cell (L24). Otherwise, we were required to properly form the sub-point sequences for the two quadrants adjacent to the intersection point as stated in Observation 1. Specifically, the intersection point should always be inserted into the sub-point sequence for the previous quadrant (L26-29), but the center of the MBR was required only if it was located within the object (L30-32). Moreover, the intersection point was also required to be inserted into the sub-point sequence for the next quadrant (L33-34). After examining all points representing the flood object, this algorithm was invoked recursively to process each quadrant with its corresponding sub-point sequence.

V. EXPERIMENTAL RESULTS

We designed several experiments to evaluate and compare the performance of the three proposed partitioning methods, which will be referenced and denoted as *FS*, *MIR*, and *Quad*. For the purpose of comparison, we also implemented the *MBR* approach, which directly uses the MBR of each flood object as the flooded cell without any partitioning.

The programs of the proposed methods are implemented in Visual C++, and the experiments are performed on a personal computer with Intel i7-2600 3.4GHz CPU and 16GB memory. The default value of the parameter e, i.e., the side length of the flooded cell required by the FS method, is set as 50 m by observing the smallest possible real flood objects, unless explicitly specified. Besides, the parameter t_r required by the *Quad* method is set as 0.95 since we wish the value as close to one as possible. The flood data were obtained from the government website[2] and the empirical studies were conducted on two representative cities, Taipei (411 flood objects) and Keelung (436 flood objects).

1. Flooded Cells Produced by the Partitioning Algorithms

We first compared the set of flooded cells produced by each partitioning algorithm and applied two metrics, which are the coverage ratio based on Eq. (1) and the numbers of produced cells. Note that both values are the smaller the better. To examine how the parameter e affects the output of the FS approach, we use three values 40 m, 50 m, and 60 m, and their results are denoted as FS40, FS50, and FS60, respectively.

The experimental results based on the Keelung data set are presented in Fig. 9. By first examining the coverage ratio displayed in Fig. 9(a), the MBR approach can be seen to exhibit the largest coverage ratio, indicating that the flooded cells produced using this method usually cover a great deal of extra space. By contrast, the Quad approach produced the smallest coverage ratio, demonstrating that its flooded cells could best approximate the original flooded areas. Regarding other approaches, no much difference was observed between the FS approach and the MIR approach because of the shapes of the original flood objects. We compared the number of cells produced by each method, as displayed in Fig. 9(b). The MBR approach could be seen to produce the least number of cells because each flood object corresponded to exactly one flooded cell. Notably, the Quad approach produced fewer cells than the FS40 and FS50 approaches did. This was a result of its ability to produce a single cell covering a great deal of flooded space. However, because the stopping criteria was relatively strict (i.e., 95% of the cell space should be flooded), more cells were produced than the MIR and FS60 approaches.

We then examined the experimental results based on the Taipei data set depicted in Fig. 10. The performances of the proposed methods in Taipei were similar to those in Keelung. This

[2] http://dmap.ncdr.nat.gov.tw.

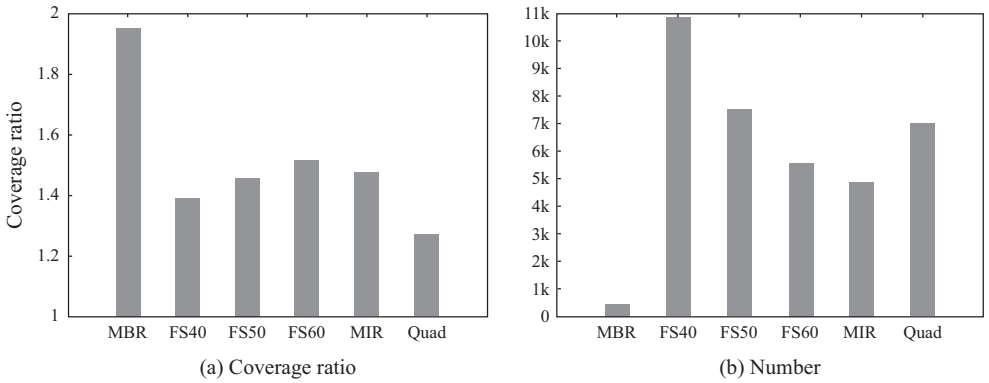


Fig. 10. Flooded cells based on the Taipei data set.

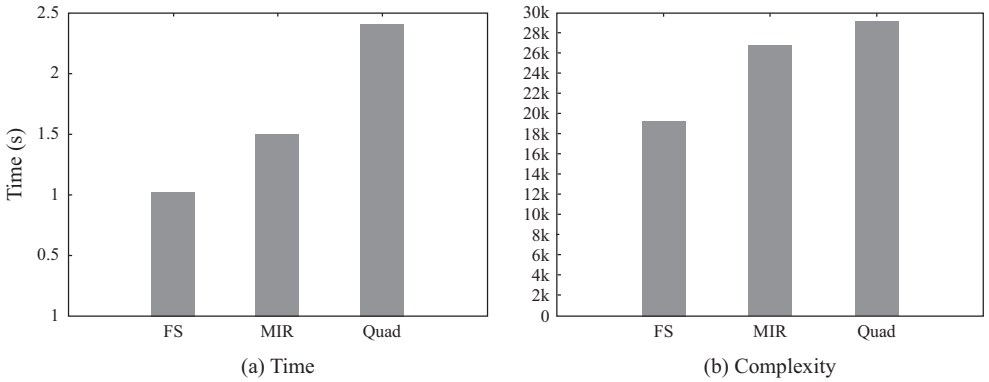


Fig. 11. Partitioning time based on the Keelung data set.

demonstrates that our partitioning algorithms were all better approximations of the original flood objects than the MBR approach, regardless of the city geographical characteristics. Finally, comparing the outcomes of FS40 and FS50, we can see that the coverage ratio of the former is indeed better than that of the later (about 0.96 times), but the number of flooded cells will increase a lot (about 1.5 times). By contrast, the coverage ratio of FS60 will be even larger than that of the MIR approach and is unacceptable. Therefore, we will use 50 m as the side length of the flooded cell for the remaining sets of experiments.

2. Execution Time of the Partitioning Algorithms

In this subsection, we evaluate the difference in execution time among partitioning methods. First, we examined the time required to produce flooded cells. Several spatial operations are involved in this task, which include the function *intersects* determining whether the two rectangles overlap (L14 of Algorithm FS), the size computation and comparison (L3 of Algorithm FS, Algorithm MIR, and Algorithm Quad), and the function *within* performing the point containment test (L6 of Algorithm MIR and L7 of Algorithm Quad). All of these spatial operations were implemented in Boost C++ libraries, and we summed up the times required to perform these spatial operations to approximate the complexity of each algorithm because they are more expensive than other standard operations.

The experimental results for the Keelung city are displayed in Fig. 11. The partitioning time (Fig. 11(a)) is obviously affected by spatial complexity (Fig. 11(b)). We can also see that the FS method is relatively straightforward and required the least amount of time. The Quad algorithm was the slowest because of its complexity. Regarding the MIR method, although the number of generated flooded cells was successfully reduced (Fig. 9(b)), it required each element in the point sequence to be examined and costly spatial operations to be performed and thus was slower than the FS method. The experimental results for Taipei city are presented in Fig. 12, and indicate that the time required for execution was less for Taipei city data set than that for the Keelung data set because of fewer input flood objects; however, the relative performances among the four methods are similar.

We then compared the time required to create the R-tree for the Keelung data set. As depicted in Fig. 13(a), the MBR method was the fastest and the FS method was the slowest because they were primarily affected by the number of produced flooded cells, as indicated in Fig. 9(b). The indexing time for the Taipei data set is displayed in Fig. 13(b). The performance was the same for the four methods, all of which could complete construction of the index within 0.06 seconds.

3. Comparisons of the Application Results

To further evaluate the effects of different sets of flooded cells,

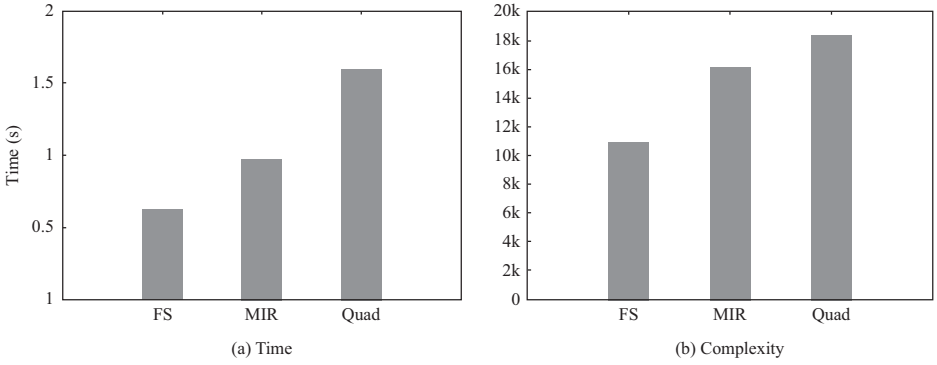


Fig. 12. Partitioning time based on the Taipei data set.

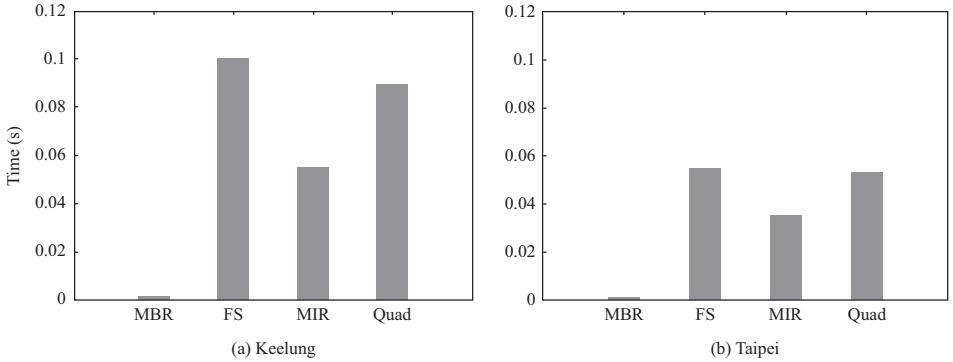


Fig. 13. Indexing time.

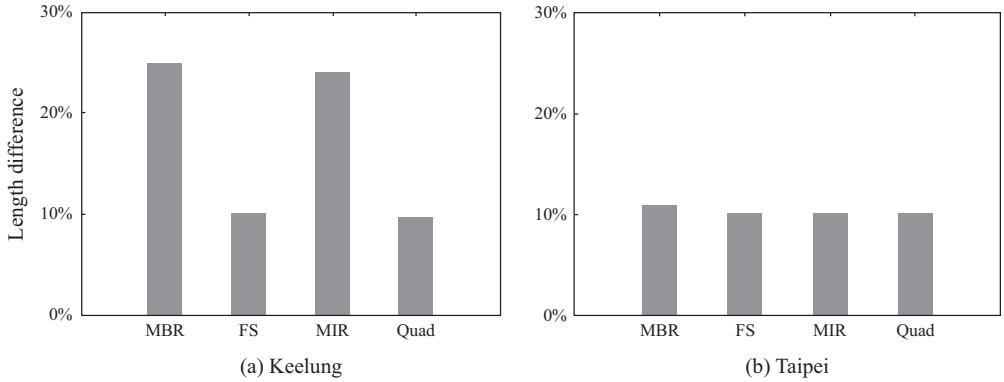


Fig. 14. Difference in length.

we created a path-planning system for flooded regions. This system first identifies impassable roads that are intersected with any flooded cell. These roads are then removed, and the remaining roads are used to run the shortest path algorithm proposed by Dijkstra (1959). Those shortest paths produced based on different sets of flooded cells are then compared. To perform the experiments, we randomly produced 150 pairs of source points and target points and randomly retained 30 data items for which at least one method produced a detour in response to flooding. For each method, we computed the normalized length difference (*nld*) between the unflooded detour and the original shortest path and obtained the average using the following equation, where a smaller *nld* value is preferred and corresponds to shorter average detours:

$$nld = \frac{\sum \frac{\text{length of the unflooded detour-length of the original path}}{\text{length of the original path}}}{\text{number of paths}} \quad (4)$$

From a comparison of the path lengths based on the Keelung data set, as depicted in Fig. 14(a), we can see that the Quad method produced the smallest value, suggesting that it could identify the shortest average detour. Specifically, the average difference in length caused by the MBR method was 2.5 times of that caused by the Quad method, demonstrating an obvious difference. The

experimental results for Taipei city are displayed in Fig. 14(b). However, because more roads are available in Taipei to plan the detour, the lengths of the shortest paths obtained by different methods were not significantly different. Therefore, for an ordinary city without many alternative roads, reducing the misjudgment by properly partitioning flood data is remarkably helpful for obtaining a shorter detour.

VI. RELATED WORK

This paper mainly discussed the proper representation of flood objects in spatial indices for efficient future reference. Because the exact algorithms for managing complex geometric objects are usually complicated as well as difficult and time-consuming to implement in practice, researchers have proposed various methods for approximating an arbitrary closed curve, and the *rectangle* is the preferred form because of its simplicity. For example, the minimum-area encasing rectangle was proposed by Freeman and Shapira (1975) and the maximum enclosing rectangle was proposed by Brinkhoff et al. (1994). In the book written by Har-Peled (2011), one chapter discusses how to approximate the minimum volume bounding box of a point set in any dimension, and exact algorithms in 2D and 3D are provided. However, we could not directly apply these types of rectangles in this paper because they are allowed to rotate to certain degrees, which does not meet our requirements as discussed in Definition 3.

Besides, efficiently indexing multidimensional spatial data has been an prominent research problem. Gaede and Günther (1998) surveyed many structures, the R-tree index being one of the most widely applied. It can be used in many spatial operations, such as the simple overlapping operation as well as more complicated queries, such as the *k*NN (nearest neighbor) queries, as discussed by Roussopoulos et al. (1995) or in geographical data, as discussed by Schubert et al. (2013). Following the initial construction algorithm of the R-tree index designed by Guttman (1984), other researchers have studied its variations to improve querying performance. For example, Beckmann et al. (1990) proposed the R*-tree to incorporate a combined optimization of areas, margins, and overlaps of each enclosing rectangle of the inner node. Ang and Tan (1997) discussed a favorable linear algorithm to split the node. More complex indices have also been proposed by extending the original structure to meet different information needs. For example, Lu and Shahabi (2017) studied geo-tagged aerial videos, the field-of-views of which are irregular quadrilateral shapes. They proposed a TetraRtree structure to represent the four corners of each quadrilateral and thereby efficiently determine whether a given object appears in a particular video. Although such a structure is not appropriate for flood data, these research results merit further study.

Finally, we reviewed literature on flood-forecasting systems. For example, Thielen et al. (2009) described the European Flood Alert System for providing flood forecasting in transnational river basins. It has large-scale applications, including in grid architecture for the integration of several national hydrologic and meteorological services. The web-based flood forecasting system (WFFS) discussed in Li et al. (2006) is a small to medium-sized system, meaning that it is component-based and applies the Java technique. Chang et al. (2013) described a system for Linpien city, located in the southern part of Taiwan, that predicts the water level of each region three hours in advance according to current information regarding the weather, nearby river flow, and tide. In addition to the complete system, some researchers have focused on specific functions. For example, Granell et al. (2010) identified tasks that are frequently required by flood forecasting systems, such as formatting data, and presented them as web services. Jhong et al. (2017) and Yu and Coulthard (2015) studied how to create inundation maps, and Groeve and Riva (2009) and Liu et al. (2017) quickly identified flood zones using near-real time satellite data and other useful information. Chang et al. (2017) proposed to transform computed numerical data into landmark-based messages to provide more informative warning messages. Our research results should be suitable for incorporation into such systems to extend their functionality.

VII. CONCLUSION

Proper representation of flood objects in spatial indices is crucial for providing an efficient querying facility. For example, after a sudden heavy rain, we may wish to know whether a certain road is safe or impassable. In this paper, we advocate partitioning the minimum bounding rectangles of flood objects into smaller rectangles to reduce the possibility of misjudgment and facilitate the construction of an R-tree index. The main challenge lies in identifying a suitable method for processing the point sequence representing a flood object and determining the optimal size of the rectangle. We designed three partitioning methods based on different motivations to address this problem.

To evaluate the proposed three methods, we performed comprehensive experiments based on data of two representative cities. Unsurprisingly, the Quad approach, which generates flooded cells of different sizes to cope with the shape of the original flooded area, can usually successfully avoid misjudgment and exhibited the best performance. Moreover, the simplest FS approach also demonstrated excellent performance when the side length was set as 50 m. Notably, the value "50" was the smallest side length observed from the flood objects used for the experiments and may serve as a prime candidate when applying our system to other regions; however, the optimal value may vary slightly, and further study should be conducted on this topic.

REFERENCES

Ang, C. H. and T. C. Tan (1997). New linear node splitting algorithm for R-trees. In Proceedings of the 5th International Symposium on Advances in Spatial Databases (SSD'97), LNCS 1262.

Beckmann, N., H. Kriegel, R. Schneider and B. Seeger (1990). The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of the ACM SIGMOD conference.

Brinkhoff, T., H.-P. Kriegel, R. Schneider and B. Seeger (1994). Multi-step processing of spatial joins. In Proceedings of the ACM SIGMOD conference.

Chang, Y.-H., Y.-T. Liu and Y.-Y. Tan (2017). Landmark-based summarized

messages for flood warning. Transactions in GIS 21(5), 847-861.

Chang, Y.-H., P.-S. Wu, Y.-T. Liu and S.-P. Ma (2013). An effective flood forecasting system based on web services. In Advances in Intelligent Systems and Applications-Volume 2, pages 681-690. Springer.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik 1, 269-271.

Finkel, R. A. and J. L. Bentley (1974). Quad trees: A data structure for retrieval on composite keys. Acta Informatica 4, 1-9.

Freeman, H. and R. Shapira (1975). Determining the minimum-area encasing rectangle for an arbitrary closed curve. Communications of the ACM 18(7), 409-413.

Gaede, V. and O. Günther (1998). Multidimensional access methods. ACM Computing Survey 30(2), 170-231.

Granell, C., L. Diaz and M. Gould (2010). Service-oriented applications for environmental models: Reusable geospatial services. Environmental Modelling & Software 25(2), 182-198.

Groeve, T. D. and P. Riva (2009). Global real-time detection of major floods using passive microwave remote sensing. In Proceedings of the 33rd International Symposium on Remote Sensing of Environment Stresa.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In Proceedings of the ACM SIGMOD conference.

Har-Peled, S. (2011). Geometric Approximation Algorithms. America Mathematical Society, 1 edition.

Hart, P. E., N. J. Nilsson and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics 4(2), 100-107.

Jhong, B.-C., J.-H. Wang and G.-F. Lin (2017). An integrated two-stage support vector machine approach to forecast inundation maps during typhoons. Journal of Hydrology 547, 236-252.

Kreyszig, E. (2011). Advanced Engineering Mathematics. John Wiley & Sons, Inc., 10 edition.

Li, X.-Y., K. W. Chau, C. Cheng and Y. S. Li (2006). A web-based flood forecasting system for shuangpai region. Advances in Engineering Software 37(3), 146-158.

Liu, X., H. Sahli, Y. Meng, Q. Huang and L. Lin (2017). Flood inundation mapping from optical satellite images using spatiotemporal context learning and modest adaboost. Remote Sensing 9(6), 617.

Lu, Y. and C. Shahabi (2017). Efficient indexing and querying of geo-tagged aerial videos. In Proceedings of the ACM SIGSPATIAL conference.

Roussopoulos, N., S. Kelley and F. Vincent (1995). Nearest neighbor queries. In Proceedings of the ACM SIGMOD conference.

Schubert, E., A. Zimek and H.-P. Kriegel (2013). Geodetic distance queries on R-trees for indexing geographic data. In Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD), LNCS 8098.

Thielen, J., J. Bartholmes, M.-H. Ramos and A. de Roo (2009). The European flood alert system-part 1: Concept and development. Hydrology and Earth System Sciences, 13(2).

Yu, D. and T. J. Coulthard (2015). Evaluating the importance of catchment hydrological parameters for urban surface water flood modelling using a simple hydro-inundation model. Journal of Hydrology, 524, 385-400.