VOL. 13, NO. 23, DECEMBER 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

A TWO-PASS SCANNING SCHEME FOR RECTANGLE SHAPE DETECTION IN PLANE AREA

Junghoon Lee and Gyung-Leen Park
Department of Computer Science and Statistics, Jeju National University, Jeju-City, Republic of Korea
E-Mail: glpark@jejunu.ac.kr

ABSTRACT

This paper challenges the problem of finding valid rectangles over the plane area containing a lot of obstacles or anchor points. A valid rectangle, formed by two anchors points in one of its two diagonal lines, does not embrace any obstacle inside it, while allowing boundary-located ones. Each anchor point is investigated from left to right to check whether it can make a valid rectangle with other remaining points one by one. To overcome the problem that all given points must be sorted in conflict directions in regard to the y-coordinate of the inspection point, our scheme suggests a two-pass scanning procedure. The first pass sorts the points on the same vertical line in descending order and checks the validity only those above the inspection point, and vice versa on the second pass. The proposed scheme reduces the constant of the dominating term, even though the overall time complexity still remains at $O(n^2)$, for n obstacles. The efficiency will be very helpful when the number of obstacles increases in counting the number of rectangles, filtering coordinates, assigning to a robot, and the like.

Keywords: two pass scanning, rectangle finding, obstacle, plane area, time complexity.

1. INTRODUCTION

There are many area search problems having different goals and restrictions. The target area can be rectangular or arbitrary shaped. Moreover, it can possibly contain some obstacles such as rocks, walls, and the like. One of the most interesting problems in this category is to find feasible rectangles in the target plane. Inspired from a coding contest problem, we find that its solution can be applied to many cases [1]. For example, given the locations of obstacles, we can find the number of rectangles containing no obstacles, locate the maximum of such rectangles, and identify the obstacle which is most critical to maximize the available square. In addition, a rectangle-by-rectangle search can be built upon the area categorization. Besides, the original problem is to find feasible rectangular spots for tent pitching, avoiding scattered wedges.

There are many researches regarding the rectangle operations on a 2-dimensional flat area. For example, [2] finds a subset of non-overlapping rectangles, each of which is attached to a boundary. Here, each element has its own weight and the selection scheme maximizes the sum of weights. It can be applied to bus escape routing in printed circuit boards, where the boundary possibly provides common electrical ground. In addition, [3] challenges the detection of rectangular objects in a still image captured from a moving vehicle. This scheme consists of corner area detection, position calibration, corner score normalization, sign hypotheses generation, and pattern training. It can be applied to recognize the directional road sign in real time for the sake of automatic driving support. Besides, rectangular area-based search can help robots to perform their missions such as map building, obstacle detection, and the like [4]. Here, a rectangle-by-rectangle area assignment supports a collision-free and independent traversal.

2. PROBLEM DESCRIPTION

Figure-1 describes the problem. There are wedges, or obstacles, over the plane area. They are labeled from *A* to *F*. Basically, two wedges form a rectangle, as long as they are not belonging to the same vertical or horizontal line. They can be considered to be two opposite corners of a rectangle. However, a rectangle is valid only when it embraces no other wedge inside it. A wedge located on the border line is permitted. In the example of the figure, a rectangle can be denoted by two points, either (left-top, right-bottom) or (left-bottom, right-top). Hence, we can find 11 valid rectangles, namely, (*A*, *B*), (*A*, *C*), (*A*, *D*), (*A*, *E*), (*B*, *E*), (*B*, *F*), (*C*, *D*), (*C*, *F*), (*E*, *D*), and (*F*, *D*). The figure shows only a subset of valid rectangles. The rectangle (*A*, *F*) is not valid as it includes *B* and *C*. (*C*, *F*) is feasible, as *E* is not inside the area but just on the border line. Two rectangles of (*B*, *E*) and (*C*, *F*) are the same in their shapes, if 4 points are located at each tip of a rectangle. However, it doesn't matter they are considered to be same or not, as the overlapped spatial search can be accomplished just with a liner overhead.

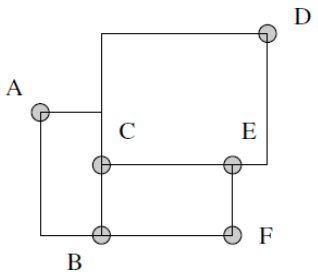

Figure-1. Rectangle finding problem.

Finding all feasible rectangle areas can be carried out by simply investigating all feasible pairs and checking whether each pair contains any other wedge. For *n* wedges, there exist ${}_nC_2$ candidates, and it needs *n*

operations to scan all other wedges. Scanning a point accompanies checking whether it is included in the candidate rectangle, comparing the x-coordinates and y-coordinates. After all, the time complexity of this scheme will be $O(n^3)$. Alternatively, we can build a $n \times n$ matrix, say E, where $E[i][j]$ stores the number of points inside the rectangle formed by (0,0) and the point consisting of ith largest x-coordinate and jth largesty-coordinate. Then, the whole matrix will be investigated to decide whether a rectangle has any other point or not. Even though it can be carried out with $O(n^2)$ complexity, it is quite complex to match the ith point in the x-axis and jth point in the y-axis. In addition, building and scanning matrix takes $O(n^2)$, respectively, making the constant of n^2will be 2.

3. ALGORITHM DESCRIPTION

3.1 Basic idea

Figure-2 illustrates our main idea, which begins a straight-forward observation that it is possible to check whether two points can meet the validity requirement just by comparing their y-coordinates, when they are sorted by the x-coordinate. Here, it takes only $O(n\ logn)$ time to sort all given points. Suppose that a point, say P, is to be the left-bottom of a rectangle and wants to sequentially check whether the respective points right to it can be its counterpart, namely, the right-top point. They can be scanned according to their x-coordinates, if a point has the same xor y-coordinate with P, it cannot be the right-top point of a valid rectangle. It's not difficult to see that a point can makea rectangle with P only if it has no point lower than itself up to its turn. In Figure 2(a), the procedure sequentially checks A, B, D, and C according to their x-coordinates. (P, A),(P, B), and (P, C) have no other point inside them. However, (P, D) includes B, which is lower to and left to D. For the case P will be the left-top point, a point can makea valid rectangle if it has no upper point left to it.

The problem arises when multiple points appear on a same vertical line, and how to handle such a case is the key to solve this problem. For those points above P, as shown in Figure 2(b), if those points sharing the same x-coordinate are sorted by the y-coordinate in descending order, we can easily determine valid rectangles in the same way as described in the previous paragraph. Here, assume that there is no point upper from A between P and A. If a point, say M, is inside the rectangle (P, A), M will replace A. (P, A) is valid first, as no point is lower than A until its turn. Next, in B's turn, points have been processed up to A and no point is lower than B, so it's counted valid. As such, (P,C) is also valid. On the contrary, for those points below P sharing the same y-coordinate, if they sorted in ascending order, we can check the validity in the order of (P, C), (P,B), and (P, A). Actually, the procedure demand that all points be ordered according to how close to the horizontal line passing P, in each of two groups, namely one above $P$and the other below P. The sorting directions are opposite, and this is why the two pass scheme is necessary.

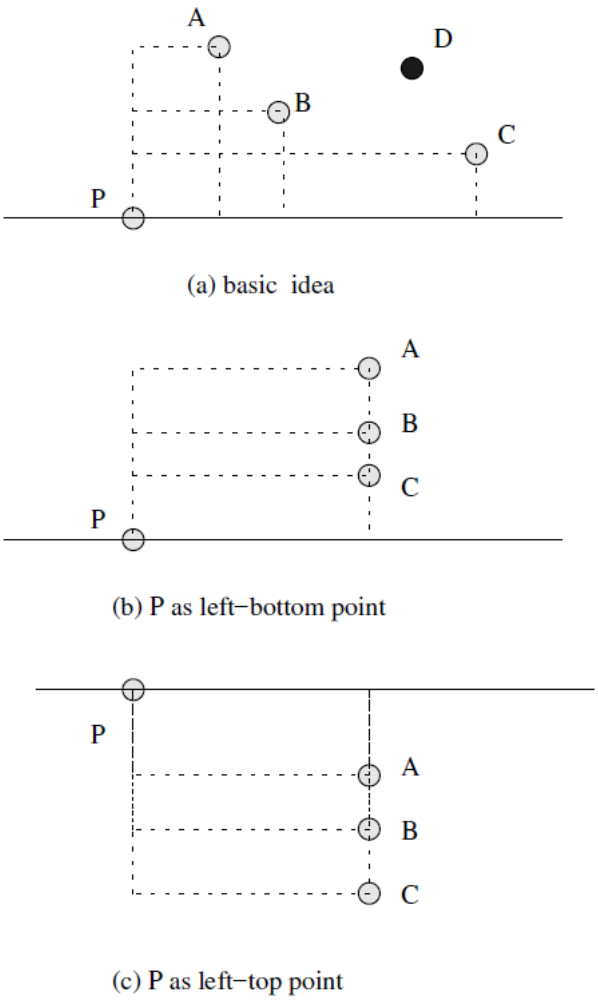

Figure-2. Main idea.

3.2 Algorithm description

The procedure begins from the leftmost point and checks the validity with all other points right to it one by one. In Figure-1, A, as an inspection point, will be checked with all others, namely, (A, C), (A, B), (A, E), (A, F), and (A, D).Then, C becomes the next inspection point and is checked with B, E, F, and D, sequentially. For each point, it is necessary to sort all points right to it in ascending order for those above and in descending order for those below. For example, in Figure 3, P_1 needs to sort (A, B, C) in descending and (E, D) in ascending order. P_2 needs to sort(A, B) in descending and (E, D, C) in ascending order. This makes it impossible to scan all points in a single pass. Hence, our scheme employs a two pass scanning procedure. At the first step, points on the same vertical line are sorted in descending order. Points are sorted to (P_1, P_2, A, B, C, D, E). Then the scan procedure considers the point only above inspection points. In the second step, those points sharing the same x-coordinate are sorted in ascending order. Namely, points are sorted to (P_2, P_1, E, D, C, B, A). Then, only the points below the inspection points are investigated.

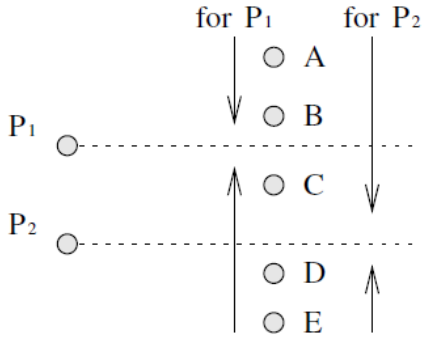

Figure-3. Different order requirement.

The proposed scheme can be described as in Figure-4. Initially, the set of $\{P_i\ (x_i, y_i) \mid 0 \le i < n\}$ is given. That is, each point P_i is specified by x_i and y_i. In addition, y_{max} and y_{min} are maximum and minimum bounds of y-coordinates, respectively. *up* traces the current lowest for those points above the inspection point. It monotonously decreases each time a valid rectangle is found. If a point has larger y-coordinate, it is not valid. *down* also traces the current highest for those points below the inspection point and increases in the same way but opposite direction. They are reinitialized for each inspection point. It must be mentioned that the processing order for inspection points on the same vertical line, namely $P_1 \rightarrow P_2$ or $P_2 \rightarrow P_1$, doesn't matter, as they are not interrelated. Each time the procedure finds a valid rectangle, it invokes a pre-defined operation, which can be just counting the number of rectangles, filtering coordinates, assigning to a robot, and the like.

With this pseudo code, we implement the algorithm using Java, which provides a safe sorting library. In addition to the computational efficiency coming from the merge sort mechanism, this API allows us to specify multiple sort criteria. Here, the primary one is the x-coordinate in ascending order, while the secondary one is the y-coordinate in descending in the first pass and in ascending in the second pass. The complexity, $T(n)$, will be intuitively calculated as in Equation (1).

$$T(n) = 2 \times n\ logn + 2 \times \frac{n \times (n-1)}{2} \quad (1)$$

which corresponds to $O(n^2)$ solution. As described earlier, it needs two merge sort invocations, each of which takes $O(n\ logn)$. In addition, as just points from P_{i+1} to P_{n-1} are compared for point P_i, each pass needs $\frac{n \times (n-1)}{2}$ y-coordinatecomparisons. Compared with the legacy scheme which needs$2 \times n^2$ comparisons, our scheme reduces the complexity by about n^2 - 2 $n\ logn$, even though the dominating term still remains at n^2. This improvement will make the application more efficient when the number of obstacles gets higher.

```
proc CheckRect
//— first pass
  sort {Pi} (x ascending, y descending)
  for i ← 0 to n-1
    up ← Ymax
    for j ← i+1 to n-1
      if yj ≤ yi
        continue
      if yj ≤ up
        operation(Pi, Pj)
        up ← yj
      end if
    end for
  end for
//— second pass
  sort {Pi} (x ascending, y ascending)
  for i ← 0 to n-1
    down ← Ymin
    for j ← i+1 to n-1
      if yj ≥ yi
        continue
      if yj ≥ down
        operation(Pi, Pj)
        down ← yj
      end if
    end for
  end for
end proc
```

Figure-4. Algorithm description.

4. CONCLUDING REMARKS

In this paper, we have designed a two pass scanning scheme for finding valid rectangles over the plane area containing many obstacles or anchor points. The differentiated passescope with the conflict requirement on sorting directions. Each pass scans each point from left to right checking whether it can make a valid rectangle with a remaining point, once just for those above, once for those below. At the start of each pass, points belonging to a same vertical line, are sorted oppositely, to make coordinate comparison time $O(1)$.The proposed scheme reduces the constant of the dominating term, even though the overall time complexity remainsat $O(n^2)$. After all, compared with the legacy scheme which needs $2 \times n^2$ comparisons, our scheme reduces the complexity from n^2 to 2 $n\ logn$.

Now, we are planning to look for an appropriate application, such as finding essential sensor locations, combined with the monitoring capability and user demand [6]. Moreover, it can be expected that the computational efficiency can cope with the growth in the number of objects in diverse application areas [7].

ACKNOWLEDGEMENTS

This research was supported by the 2019 scientific promotion program funded by Jeju National University.

REFERENCES

[1] Kakao Code Festival. http://tech.kakao.com/2017/08/11/code-festival-round-1/

[2] Ahmadinejad A., Zarrabi-Zadeh Z. 2017. Finding Maximum Disjoint Set of Boundary Rectangles with Application to PCB Routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 36(3): 412-420.

[3] Wenzel T., Chou T., Brueggert S., Denzler J. 2017. From Corners to Rectangles - Directional Road Sign Detection using Learned Corner Representation. In: IEEE Intelligent Vehicles Symposium. 1038-1044.

[4] Savkin A., Li H. 2017. A Collision-free Area Search and 3D Map Building Algorithm for a Ground Mobile Robot in Unknown Indoor Environments. In: IEEE International Conference on Robotics and Biometics.

[5] Lee J., Park G. 2017. Analysis of Data Streams in a City-wide Electric Vehicle Charger Monitoring System. In: International Conference on Communication and Information Processing. 28-31.

[6] D. Puschmann P. Barnaghi and R. Tafazolli. 2017. Adaptive clustering for dynamic IoT data streams. IEEE Internet of Things Journal. 4(1): 64-74.

[7] J. Lee, G. Park, Y. Han, and S. Yoo. Big data analysis for an electric vehicle charging infrastructure using open data and software. ACM eEnergy. pp. 252-253.