Online Evasive algorithm for Sierpinski Knoop curve

Ashay Wakode[1], Arpita Sinha[2†]

***Abstract*—The paper deals with the robotic exploration problem using Sierpinski Knoop curve. An online algorithm is presented which gives an alternative path for exploration of a region using Sierpinski Knoop curve in presence of single obstacle. The non-uniform coverage and multiple obstacle problems are also dealt in brief.**

***Index Terms*—Coverage Path Planning, Sierpinski Knoop curve, Robotic Exploration, Online Evasive Algorithm.**

I. Introduction

IN 1878, George Cantor demonstrated that an interval [0,1] can be mapped bijectively onto $[0,1]^2$. A year later E. Netto proved that such mappings are discontinuous. Then the next question asked was whether there exists a curve that passes every point of an area. This was answered by G.Peano who discovered one such curve, from which such curves are called space-filling or Peano curves. Further in 1891 D. Hilbert discovered the Hilbert space-filling curve. There were more space-filling curves discovered later on by E. Moore (1900), H.Lebesgue (1904), W. Sierpinski (1912) and G. Polya (1913) [1].

In 1912 W. Sierpinski discovered the Sierpinski curve which can be constructed by repeatedly dividing the isosceles right-angled triangle into congruent sub-triangles. The interior points of the array of the sub-triangles are then connected to form a Sierpinski curve as shown in Figure 1. Sierpinski curve is originally drawn for squares. The Sierpinski curves for isosceles right-triangles are named as Sierpinski-Knoop curves, in honor of Konrad Knoop who introduced the mathematical description of the curve. From here on in the paper, we will call Sierpinski-Knoop curve as Sierpinski curve for the sake of simplicity.

Since the inception space-filling curves were studied with great enthusiasm by mathematicians all over the world due to their interesting applications in Computer Science, Robotics, Manufacturing processes [2] and Optimization [3]. Particularly in Robotics, the space-filling curves are used in exploratory problems, in which a search agent travels through a geographical region in search of interesting objects like metal, debris, a valuable resource, etc. The area which the robot is traversing may not always have simple terrain, there may be obstacles in the path, but the robot should search through each point in the demarcated area. This paper is an effort in the same direction in which we address the evasive algorithm for a search agent traversing a flat isosceles triangular area with a single obstacle placed at one of the way-point on the curve. Similar efforts have been done for Hilbert Space-filling curves in [4] and [5]. The order of a Sierpinski curve is a natural number which determines the extent to which the triangle is divided and further sub-divided. Simple division of the right angled isosceles triangle by a median will give us first order Sierpinski curve, Further dividing the triangles will give us order 2 curve and so on as shown in Figure 1. We assume a search agent to be a point object which has a certain range of detection as shown in Figure 2.

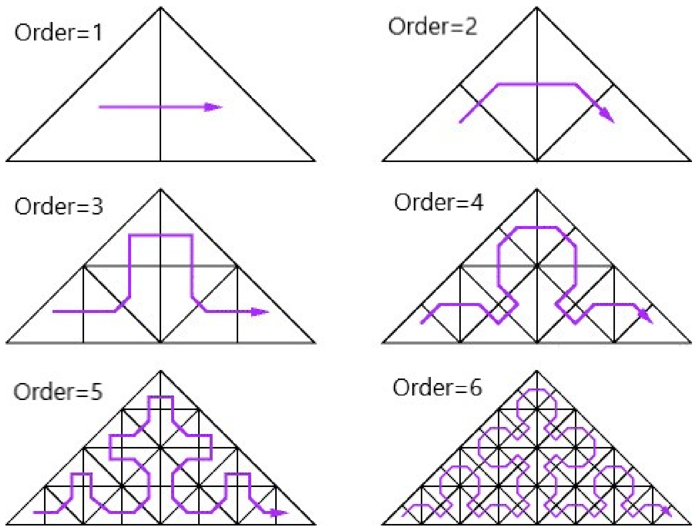


Fig. 1: Sierpinski curve for right-angled isosceles triangle

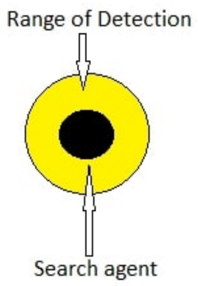


Fig. 2: Search agent with the range of detection encircling it

Rest of the paper is divided into 7 sections. Sections II presents observations made regarding the geometry of the Sierpinski curve and some properties necessary for the algorithm introduced in section III. Section III introduces the algorithm for online evasion of a single obstacle on a triangle. Section

*[1]Ashay Wakode is with the Department of Mechanical Engineering, IIT Bombay, Mumbai,400076, India. ashaywakode@iitb.ac.in

†[2]Arpita Sinha is with the Faculty of Systems and Control Engineering, Indian Institute of Technology Bombay, 400076 India. arpita.sinha@iitb.ac.in

IV deals with applications in which the range of detection of the search agent is changing. Section V briefs about the method to extend Sierpinski curve to Scalene triangle. Section VI elaborates on how we can use the presented algorithm to evade multiple obstacles in a region. Section VII deals with the implementation and simulations of the prescribed algorithm. Section VIII contains the concluding remarks.

II. Geometric Observations

For devising an algorithm we, first of all, try to find a pattern present in the curve, then try and put the pattern in the form of numbers, which can be played with to form the final algorithm.

First of all number the nodes in the order they occur on the curve.

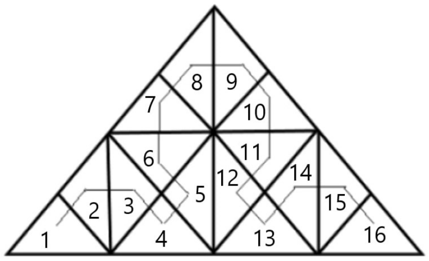


Fig. 3: Numbering of nodes for Order= 4 curve

It was observed that the Sierpinski curve of any order can be divided into 4 groups of consecutive nodes as shown in Figures 4, 5, 6 and 7 (The highlighted dots).

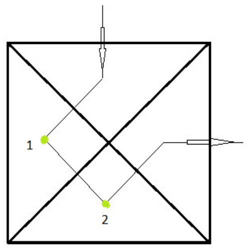


Fig. 4: Type 1 group of nodes with position of nodes in the group

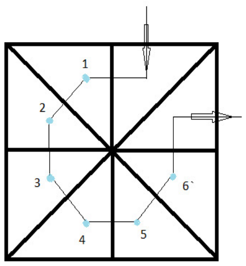


Fig. 5: Type 2 group of nodes with position of nodes in the group

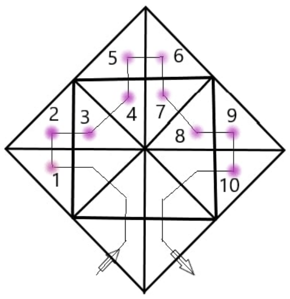


Fig. 6: Type 111 group of nodes with position of nodes in the group

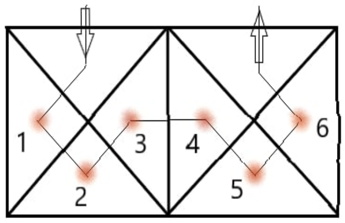


Fig. 7: Type 11 group of nodes with position of nodes in the group

Let us name them as type-1, 2, 111 and 11. The last two groups are named 11 and 111 since it can be easily observed that 11 is made by two consecutive 1s and 111 is made up of three consecutive 1s with some intervening nodes. These groups of nodes shown can be rotated by 45 or 135 degrees in the whole Sierpinski curve. It was observed that the alternate path for a node in one of the types of nodes is the same and does not vary with the position of the group in the curve. This conclusion is used in the next section of the paper.

It should be observed that if the obstacle is present at first or last three nodes then no alternate path is possible. It should be also observed that the group 11 always lies at the start or the end, So, nodes 3 and 4 of group 11 will always be among three at the start or the end, as can be seen in Figure 8. for type B. So, we need not worry about the alternate path for those.

These groups may not appear complete at starting and the end of the curve, So, we play a trick of adding some nodes at start and end so that we can completely divide the Sierpinski curve with into group 1 and 2, This is can be done in 4 different ways as shown for Sierpinski curve of order 4 in Figure 8.

Completing the Sierpinski curve gives us a way to tag every node of the curve i.e we can divide the array of nodes into tag-1 or tag-2 as per the position of the node in the curve.

For example, In Figure 8., we will tag the type-A curve in following way- (Only the nodes present inside curve are tagged here. The first number in the bracket represents the node number while the second number represents the tag of the node)
(1-2),(2-2),(3-2),(4-1),(5-1),(6-2),(7-2),(8-2),(9-2),(10-2),(11-

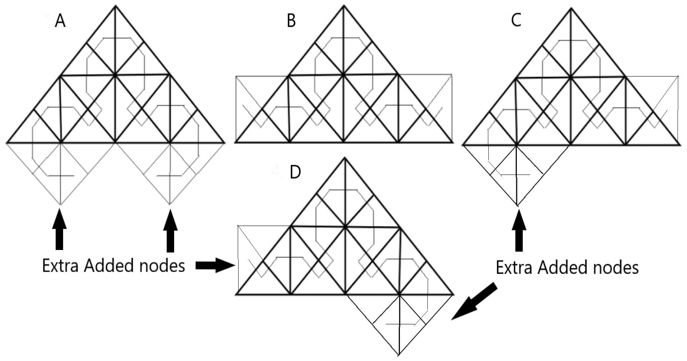


Fig. 8: Completing the Sierpinski by adding extra nodes at the start and end of the curve

2),(12-1),(13-1),(14-2),(15-2),(16-2)
Here we represent A, B, C or D type of Sierpinski curve by A(n), B(n), C(n) or D(n), where n is the order of the curve and A, B, C and D represents how the curve is completed.

So, we can represent A as a sequence of tags like
A(4)=(2,1,2,1,2)≡(21212)
We can tag B as-
(1-11),(2-11),(3-11),(4-11),(5-11),(6-2),(7-2),(8-2),(9-2),(10-2),(11-2),(12-11),(13-11),(14-11),(15-11),(16-11)
Hence, B(4)=(11,2,11)≡(11211). Similarly we can tag C and D.

We have described the curves in form of numbers. Now we observe how two lower-order Sierpinski curves can be combined to form a higher-order Sierpinski curve. We observe Figure 9. It can be seen that two B(3) Sierpinski curves can be concatenated to form A(4).

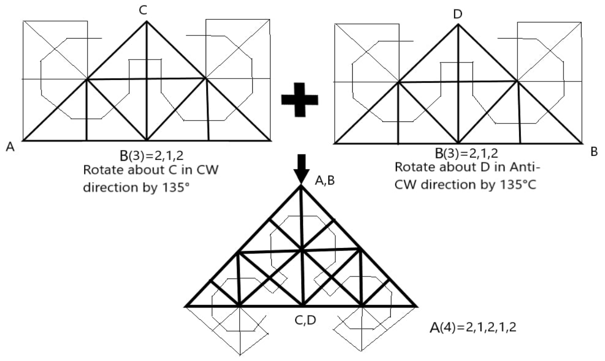

C
D
A
B(3)=2,1,2
Rotate about C in CW direction by 135°
B(3)=2,1,2
Rotate about D in Anti-CW direction by 135°C
B
A,B
C,D
A(4)=2,1,2,1,2

Fig. 9: Concatenating two B(3)s to form one A(4)

So, we can conclude that concatenating B(n) and reverse(B(n)) (reverse(B(n)) means the sequence B(n) written in reverse order) we can get A(n+1). More formal conclusion is given below,
A(n+1)=Concatenate((B(n)-last element),(If n% 2==0 then 1, else 2),reverse(B(n)-last element))
For example, in the Figure 9.
B(3)=212, therefore,
A(4)=Concatenate((212-last element),(1),reverse(212-last element))=(21212)
Now, we have established a way to create A(n+1) from B(n), but we want to create higher order B from lower order B. So, we observe how to convert A to B. It can be easily seen that if change the first and the last numbers of the A from 1→2 or 2→1 we get B. In simple words we change the first and last number of A to 1 if it is 2 or to 2 if it is 1.
So, we have established a method to describe each B type of Sierpinski curve in terms of numbers.

III. Algorithm

In this paper we will be working with only one obstacle placed at a node such that a robot cannot visit or pass through the sub-triangle associated with the node. So, further we will assume that the obstacle is not present at first and last three nodes. Alternate path will take us from **(k-1)** node to **(k+1)** node, where **k** is the obstacle node number.

It is observed that the alternate path for a particular node depends only on the tag it has and it's position inside the group of the nodes. We tabulate the alternate paths for all the tags and position of node combinations possible in Table 1.

The given evasive techniques for each tag and position is exhaustive because every Sierpinski curve (with completed groups at start and end) can be drawn out of these groups of nodes. And we have given evasive technique for each of them in the table except, nodes 3 and 4 of group 11 which will always end up in the first or last three nodes of the curve.

IV. Non-uniform range applications

A search agent exploring a region on the Sierpinski curve will have a certain range for the detection of obstacles and other objects of interest. The aim is to cover the total area of the parent triangle while the search agent is traveling. It can be easily seen that the order of the Sierpinski curve followed will be directly related to the range of the search agent since the area of sub-triangles reduce with increasing order as shown in Figure 1. Also, the range is a function of power, which be optimized or probability of finding objects of interest may be high in a particular part of a region and therefore a path with higher-order Sierpinski curve may be implemented for a certain part of the region, at the same time the search agent will have to travel larger distance, so we'll always have a trade-off between the range and order of Sierpinski curve. As shown in Figure 10 the left side right triangle is traveled with a certain order and the right side right triangle is traveled with different order due to a reduction in the range of the search agent. The algorithm can be used for a single obstacle in both triangles, left and right individually.

V. Extension to Scalene triangle

The algorithm given in the above section is worked on an isosceles-right triangle, which is the most simple case we can have. While working on a real-life problem the region most of the time may not be an isosceles right triangle, or a triangle, or even a straight line curve. It will be most of the time a closed curve. In this section, we will introduce how we can apply the algorithm to a closed region with a single obstacle which is assumed to occupy a single node in the exploratory path of the search agent.
Any given closed curve is converted in a closed straight-line

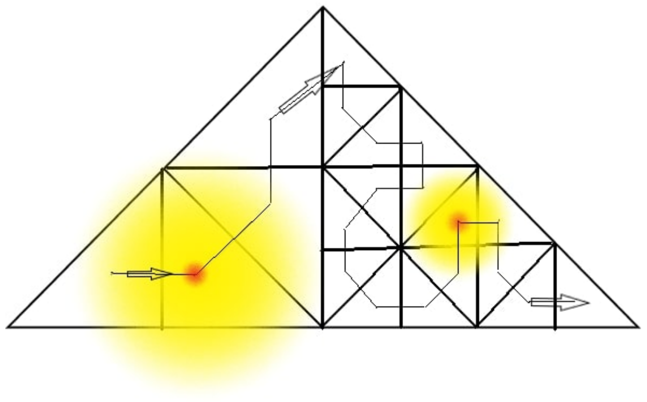


Fig. 10: Red dot represents Search agent, Yellow circle is it's range

curve as shown in Figure 11 and then it can be converted into an array of triangles as shown in the same figure (by using polygon triangulation [6]) which are essentially scalene triangles. So, it all boils down to application on a scalene triangle, Hence, once we are can apply the algorithm to a single scalene triangle then the same procedure can be applied to the whole array of triangle one after other (Or more than 1 search agents can be used) and thus the whole region can be explored. So, now given a scalene triangle we have to devise a Sierpinski curve, the procedure for the same is derived in [7]. In the method given, a median is drawn to divide the triangle into sub triangles, which are then again divided into sub triangles in the way given in Figure 12. The interior points are then connected to form the Sierpinski curve.
Now, we are able to draw a Sierpinski curve of a given order for a scalene triangle, which implies that we can apply the above-given algorithm for obstacle evasion to a scalene triangle and further to any random closed curve.

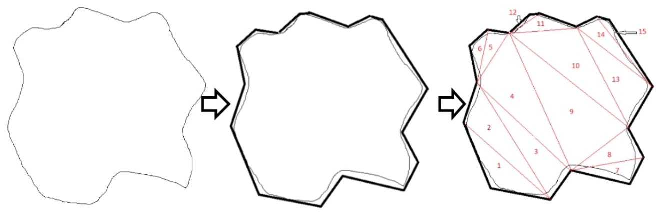


Fig. 11: Random closed curve

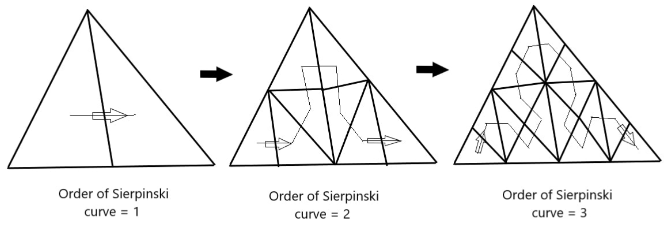

Order of Sierpinski curve = 1
Order of Sierpinski curve = 2
Order of Sierpinski curve = 3

Fig. 12: Sierpinski curve for Scalene Triangle

VI. Multiple Obstacle

The region under exploration many times may not have a single obstacle. There is a very novel way to use the given algorithm in such situations. Given any number of obstacles in a random closed region, we divide the region into an array of triangles in a such way that every obstacle lies in a different triangle, one such example is given in Figure 14. Once every obstacle occupies a single triangle, we can then apply the algorithm to each of the triangles individually.

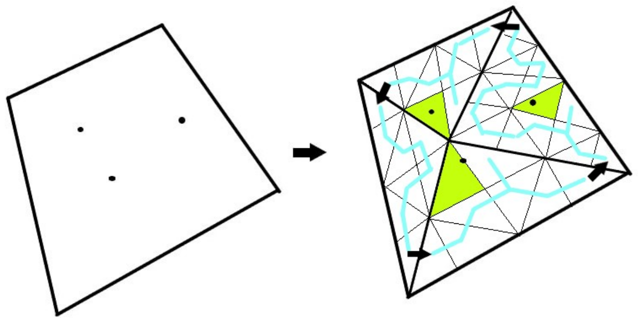


Fig. 13: Random shape with numerous obstacle (Green) divided into various triangles with each obstacle individually occupying the different triangle is transversed by a Sierpinski curve (blue)

VII. Implementation and Simulation

The above-prescribed algorithm was simulated on Python 3. A function was constructed which takes the order of the Sierpinski curve, and a system-generated random number as the number of obstacle on the curve as inputs and gives a line graph which is the alternate path to be followed by the search agent. For example in Figure 14, we observe that we have order 5 Sierpinski curve with an obstacle placed at node no.=9 (red dot), So, to come up with alternate path we observe the type of node the obstacle is present on, we see that the node type is 2 and now we observe the position of the node of the obstacle present inside the group 2 nodes, it is node number 4 in the group. Now, we will look at the table for group type 2 and node number 4 to get the alternate path that needs to be followed between node preceding the obstacle node and the node succeeding the obstacle node i.e. (k-2)→(k-3)→(k-4)→(k+3)→(k+2) where k is the node number of obstacle which in this case is 9, So, we have our alternate path shown by red arrows 7→6→5→12→11 while black arrows represent the original path 7→8→9→10→11.

VIII. Conclusion

The paper presented an algorithm that gives an alternate path to a search agent exploring a triangular region with a single obstacle and no knowledge of obstacle apriori. The following sections list down various geometric observations, which when combined gives a novel method for evasion of an obstacle. The algorithm can also be extended to non-uniform range application as pointed out. The extension to scalene triangles and any closed curve region, in general, is also discussed in

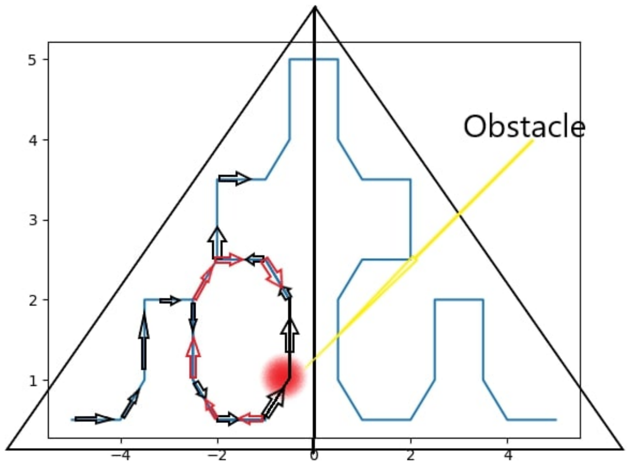


Fig. 14: Blue line is the output line graph

brief. Extension to multiple obstacles is also talked about. The future work entails working with more than one obstacle and increasing efficiency of the search.

References

[1] Sagan H. (1994) Introduction. In: Space-Filling Curves. Universitext. Springer, New York, NY

[2] Xiaoya Zhai, Falai Chen, Path Planning of a Type of Porous Structures for Additive Manufacturing , Computer-Aided Design, Volume 115,2019, Pages 218-230

[3] D. Lera and Y. D. Sergeyev, “Lipschitz and holder global optimization using space-filling curves,” Applied Numerical Mathematics, vol. 60, no. 1, pp. 115–129, 2010.

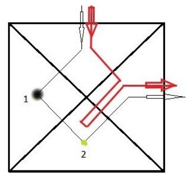
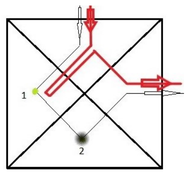
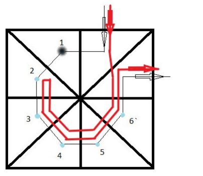
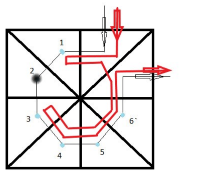
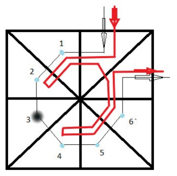
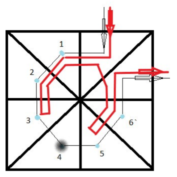
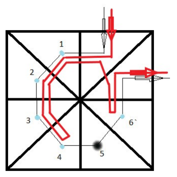
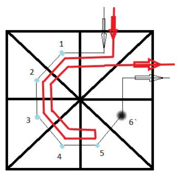
[4] S. H. Nair, A. Sinha, and L. Vachhani, “Hilbert’s space-filling curve for regions with holes,” in 2017 IEEE 56th Annual Conference on Decision and Control (CDC), Dec 2017, pp. 313–319.

[5] Joshi, Anant A., Maulik C. Bhatt, and Arpita Sinha. ”Modification of Hilbert’s Space-Filling Curve to Avoid Obstacles: A Robotic Path-Planning Strategy.” arXiv preprint arXiv:1910.03210 (2019).

[6] de Berg M., van Kreveld M., Overmars M., Schwarzkopf O.C. (2000) Polygon Triangulation. In: Computational Geometry. Springer, Berlin, Heidelberg

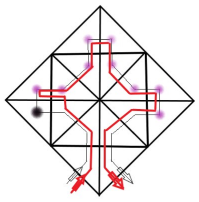
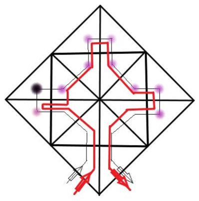
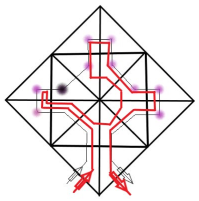
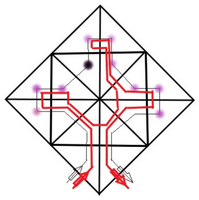
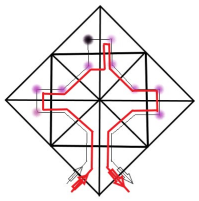
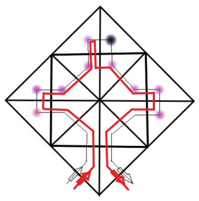
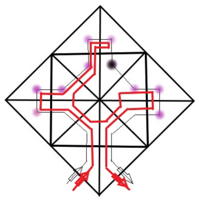
[7] Bader M. (2013) Sierpinski Curves. In: Space-Filling Curves. Texts in Computational Science and Engineering, vol 9. Springer, Berlin, Heidelberg

TABLE I: Alternate path for given group type and position of obstacle in the group

Tag-Position in group	**Alternate Path**	**Figure (Black dot is the obstacle, Red line shows altered path, thin line is the original path)**
$1-1$	(k+2)	
$1-2$	(k-2)	
$2-1$	(k+6)→(k + 5)→(k + 4)→(k + 3)→(k + 2)	
$2-2$	(k-2)→(k + 5)→(k + 4)→(k + 3)→(k + 2)	
$2-3$	(k-2)→(k-3)→(k + 4)→(k + 3)→(k + 2)	
$2-4$	(k-2)→(k-3)→(k-4)→(k + 3)→(k + 2)	
$2-5$	(k-2)→(k-3)→(k-4)→(k-5)→(k + 2)	
$2-6$	(k-2)→(k-3)→(k-4)→(k-5)→(k-6)	

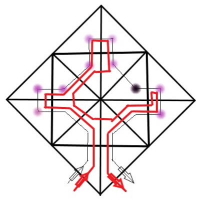
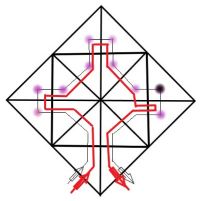
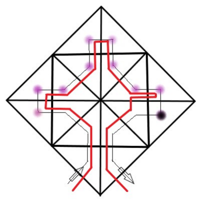
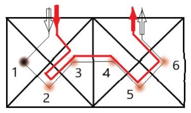
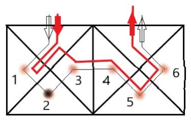
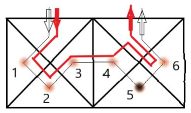
continued on the next page

TABLE I: Alternate path for given group type and position of obstacle in the group

Tag-Position in Group	**Alternate Path**	**Figure (Black dot is the obstacle, Red line shows altered path, thin line is the original path)**
111 − 1	(k+2)	
111 − 2	(k-2)	
111 − 3	(k-2)→(k-3)→(k-4)→(k + 9)→(k + 8)→(k + 5)→(k + 4)	
111 − 4	(k-4)→(k-5)→(k + 8)→(k + 7)→(k + 4)→(k + 3)→(k+2)	
111 − 5	(k+2)	
111 − 6	(K-2)	
111 − 7	(k-2)→(k-3)→(k-4)→(k-7)→(k- 8)→(k + 5)→(k + 4)	

continued on the next page

TABLE I: Alternate path for given group type and position of obstacle in the group

Tag-Position in Group	**Alternate Path**	**Figure (Black dot is the obstacle, Red line shows altered path, thin line is the original path)**
$111 - 8$	(k-4)→(k-5)→(k-8)→(k-9)→(k + 4)→(k + 3)→(k + 2)	
$111 - 9$	(k+2)	
$111 - 10$	(k-2)	
$11 - 1$		
$11 - 2$		
$11 - 3$	NA	NA
$11 - 4$	NA	NA
$11 - 5$		
$11 - 6$