Non-perfect maze generation using Kruskal algorithm

Article *in* Jurnal Natural · February 2021

DOI: 10.24815/jn.v21i1.18840

5 authors, including:

Dedi Suhaimi
Syiah Kuala University
3 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Siti Mardiati Yuni
Universitas Negeri Semarang
23 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Ikhsan Maulidi
Syiah Kuala University
9 PUBLICATIONS **19** CITATIONS
SEE PROFILE

Jurnal Natural
Vol. 21, () 2020
DOI 10.24815/jn.v20i1.xxxx
pISSN 1411-8513
eISSN 2541-4062

ORIGINAL RESEARCH

Non-perfect maze generation using Kruskal algorithm

MAHYUS IHSAN[1,2], DEDI SUHAIMI[1,2], MARWAN RAMLI[1*], SYARIFAH MEURAH YUNI[1,2], IKHSAN MAULIDI[1]

[1]Department of Mathematics, Universitas Syiah Kuala, Banda Aceh, Indonesia
[2]Multimedia Research Group, Department of Mathematics, Universitas Syiah Kuala, Banda Aceh, Indonesia

Abstract. A non-perfect maze is a maze that contains loop or cycle and has no isolated cell. A non-perfect maze is an alternative to obtain a maze that cannot be satisfied by a perfect maze. This paper discusses non-perfect maze generation with two kind of biases, namely, horizontal and vertical wall bias and cycle bias. In this research, a maze was modeled as a graph in order to generate a non-perfect maze using Kruskal algorithm modifications. The modified Kruskal algorithm used Fisher Yates algorithm to obtain a random edge sequence and disjoint set data structure to reduce processing time of the algorithm. The modifications mentioned above were adding edges randomly while taking account of the edge's orientation, and adding additional edges after a spanning tree is formed. The algorithm designed in this research constructs a $m \times n$ non-perfect maze with complexity of $O(E\ log\ V)$ where V and E denote vertex and edge set of a $m \times n$ grid graph, respectively. Several biased non-perfect mazes were shown in this research by varying their dimension, wall bias and cycle bias.

Keywords: cycle bias, grid graph, Kruskal algorithm modifications, non-perfect maze, wall bias

INTRODUCTION

Maze is a puzzle consisting of several passages structured in an intricate manner. Solving a maze means the player is required to find a route that connects two points inside the maze [1]. Using technology like a computer, a maze can also become an arena of a game [2,3]. In addition to that, a maze is also used in several fields especially as cognitive exercise and education. For example, a maze was used to observe animal's behavior [4,5] or as cognitive exercises for children [6,7].

Generally, a maze is a rectangular-shaped object resembling a grid with entry and exit point located in opposite side of the maze [8]. However, a maze can also be circularly-shaped [8,9]. It was also noted that Pullen classified a maze into seven categories, one of which is routing [10]. The routing category classifies a maze based on its passage pattern and structure, one of which is a perfect maze. A perfect maze is a maze which does not contain any loop or cycle, has no isolated point or cell and there is exactly one path connecting any two points [1]. On the other hand, a non-perfect maze is a maze that does not contain isolated point and has two points connected by more than one path [8]. In other words, a non-perfect maze is a maze which contains a cycle [11]. Additionally, it was also noted in [10] that a maze is also classified by its texture. For example, a maze containing relatively more horizontal paths than vertical paths is said to have horizontally-bias texture.

*Corresponding Author: marwan.math@unsyiah.ac.id

Received: November 2020| Revised: January 2021 | Accepted: January 2021

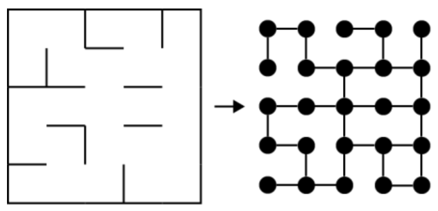


Figure 1. A maze represented in a graph

A maze can be viewed as a graph as seen in Figure 1. A grid-shaped maze is composed of collection of cells in which a cell denotes a vertex and having no wall between two cells denotes an edge between the associated vertices [1]. Furthermore, a maze can be seen as a subgraph of a grid graph. A perfect maze can then be viewed as a spanning tree of a grid graph. A perfect maze, like a spanning tree, does not contain any isolated cell and has each of its cell connected by exactly one path. This is consistent with the nature of spanning tree, which is connected and has exactly one path between any two vertices [1]. A non-perfect maze is a perfect maze with at least one wall removed. Removing a wall is equivalent to adding an edge to the graph. In the context of graphs, adding an edge to a spanning tree will create a cycle, which is identical to a non-perfect maze. Because of this structural similarity, graph-based algorithm can be used for maze generation problem.

Maze generation is a problem that has been researched since several decades ago. For example, Walter Pullen used a computer to generate the largest maze in 1987. His maze has size of 23 x 11 feet and requires 688 sheets of paper to print [12]. Several methods have been developed to generate a maze, even using a computer. These methods can accommodate several characteristics of maze [11]. These characteristics include the geometrical shape, perfectness and bias. A perfect maze can be generated using several algorithms, such as DFS, Growing Tree, Recursive Backtracking, Prim and Kruskal.

As mentioned above, a non-perfect maze can be obtained by deleting a wall of a perfect maze. As a result, any two cells in a non-perfect maze can be connected by one or more paths. Another consequence is that it is possible to find 4 adjacent cells which create a cycle in perfect maze. If we applied a non-perfect maze as a game arena, then it is possible to place an object (e.g., bomb) inside the 4-cells without blocking the characters’ path. In short, a non-perfect can be an alternative if a perfect maze cannot accommodate certain rules or gameplays in a maze game.

This paper aims to introduce an algorithm to generate a biased non-perfect maze by modifying Kruskal algorithm. There are two kind of bias accommodated in this paper, namely, horizontal/vertical wall bias and cycle bias. A maze with high ratio of horizontal bias will have relatively more horizontal paths compared to vertical ones, vice versa. In addition to that, the use of cycle bias will regulate the amount of cycle contained in the maze. Afterwards, the algorithm complexity is analyzed. This paper is structured into several sections starting from introduction followed by perfect and non-perfect maze related works. The research methodology is in the next section following with result and discussion. Finally, conclusion is included in the last section.

Related works

Several studies on maze generation have been done before. Dubey and Sarita [13] used Depth First Search (DFS) algorithm to generate a perfect maze. Their research used stack data structure to remove walls in a grid to create a perfect maze. Singh et al. [14] did a research to optimize a perfect maze generation using stack and disjoint set data structure that is able to generate a perfect maze in shorter amount of time than the general disjoint set method.

Another method to generate a perfect maze is Wilson algorithm [11]. The research compared several maze solution-finding methods such as genetic algorithm, DFS and BFS. The results of this research showed that genetic algorithm worked better on a maze with high level of openness. Hendrawan [15] used Growing Tree algorithm to develop an Android game based on a maze. Perfect maze generation can also be done using recursive methods such as recursive backtracking [16] and recursive division [17]. Perhaps the closest maze generation method to this study is Prim and Kruskal algorithm. Shah et al. [18] used DFS, Prim and Kruskal algorithm, which describe three conceptually different approaches for generating a maze.

Based on some of the researches mentioned above, it can be seen that maze generation, especially perfect maze, is a problem that has been widely studied and has led to several methods using graph-based algorithm. However, most of these studies did not consider bias and create a random perfect or non-perfect maze. We view that having a controlled bias in a maze can be beneficial and practical in some use cases (e.g., in a game, mazes with more open areas are used in low levels while those with fewer open areas are used in high levels). Using the same general idea and openness (or cycle) bias defined in [11], a perfect maze generation algorithm (i.e., Kruskal algorithm) can be extended to a non-perfect maze generation algorithm. The extension allows the algorithm to create a non-perfect maze with certain level of openness. Moreover, the algorithm can also create a perfect maze by setting openness level to its lowest value. We also add horizontal/vertical bias to allow more controls of the mazes. In short, this research tries to modify Kruskal algorithm to produce a non-perfect maze that satisfy horizontal/vertical wall bias and cycle bias in order to adjust the characteristics of the maze.

METHODOLOGY

The structural similarity between a perfect maze and a spanning tree allows graph-based algorithm to be implemented as a perfect maze generation method. One of the algorithms used to generate minimum spanning tree is Kruskal algorithm. The algorithm works on a weighted graph, which is a graph whose edges have weight. Kruskal algorithm works iteratively by picking an edge from the edge set and adding it to the set of selected edges to create a minimum spanning tree. The order of edge selection refers to the order of edge based on the weights, nondecreasingly. Cormen et al. [19] implemented Kruskal algorithm with disjoint set data structure, resulting overall complexity of $O(E\ logE)$. The algorithm uses MAKE-SET, FIND-SET and UNION function. MAKE-SET is a function to create distinct disjoint set for each vertex. FIND-SET is a function to find a set which contains a vertex, while UNION is used to join two disjoint sets containing two vertices [19]. The procedure of Kruskal algorithm is described in Algorithm 1.

Algorithm 1: Kruskal Algorithm

```
1:  $A = \emptyset$
2:  **for** each vertex $v \in G.V$
3:    MAKE-SET(v)
4:  sort the edges of G.E into nondecreasing order by weight w
5:  **for** each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6:    If FIND-SET($u$) != FIND-SET($v$)
7:      $A = A \cup \{(u, v)\}$
8:      UNION($u, v$)
9:  **return** $A$
```

The set A in Algorithm 1 is the set of edges contained in the spanning tree. At first, the set is initialized as an empty set, which means no edge has been selected yet. In line 2-3, MAKE-SET function initializes disjoint set for each vertex in $G.V$. In line 4, each edge will be sorted nondecreasingly by its weight in order to get a nondecreasing edge sequence, so that the selection process is started from the first element. In line 5-8, FIND-SET function checks whether vertex u and v, which both are incident to edge (u, v), belong to the same disjoint set. If yes, then the edge will not be added to set A since it will create a cycle. Otherwise, UNION function will join two disjoint sets which contain u and v and edge (u, v) will be added to A. Kruskal algorithm ends after a minimum spanning tree is obtained.

In this research, a non-perfect maze was formed by first creating a perfect maze using Kruskal algorithm and then followed by removing walls of the maze. This process was done by modeling a maze as a graph. If maze M has size of $m \times n$ where m is the number of rows and n is the number of columns, then its equivalent graph is a subgraph G_M of grid graph $m \times n$ where the number of vertices in G_M is mn. The number of cells in a maze corresponds to the number of vertices and the number of walls depends on the number of edges. The number of vertices of G_M is mn while the number of possible edges is $(m-1)n + (n-1)m = 2mn - m - n$. If $c(i,j)$ and $v(i,j)$ denote cell of maze M and vertex of graph G_M of row i and column j ($1 \leq i \leq m; 1 \leq j \leq n$), then $c(i,j)$ will correspond to $v(i,j)$. As a result, for any cell $c(i,j)$ and its adjacent cell $c(i',j')$, which are connected without wall, then $c(i,j)$ and $c(i',j')$ correspond to vertex $v(i,j)$ and $v(i',j')$, which are adjacent and there exists an edge incident with those two vertices. On the other hand, for any cell $c(i,j)$ and its adjacent cell $c(i',j')$, which are separated by wall, then $c(i,j)$ and $c(i',j')$ will correspond to vertex $v(i,j)$ and $v(i',j')$, which are not adjacent and there is no edge incident with those two vertices. This condition causes the number of edges of G_M to be equal to the number of nonactive walls of M, vice versa. If G_M is a spanning tree, then G_M has $mn - 1$ edges. Therefore, a perfect maze has $mn - 1$ nonactive walls and $2mn - m - n - (mn - 1) = mn - m - n + 1$ active ones.

Removing a wall is equivalent to adding an edge to spanning tree G_M. This transforms spanning tree G_M to a connected graph with cycles. If we add k edges, then G_M has $mn - 1 + k$ edges, where $mn - 1 < mn - 1 + k \leq 2mn - m - n$. These inequalities cause k to satisfy $1 \leq k \leq mn - m - n + 1$. Let us define w_a as the number of active walls, $w_a = mn - m - n - k + 1$. When $k = 1$, then $w_a = mn - m - n - 1 + 1 = mn - m - n$ and when $k = mn - m - n + 1$, then $w_a = mn - m - n - (mn - m - n + 1) + 1 = 0$. This result concludes that, in a non-perfect maze, the inequalities $0 \leq w_a \leq mn - m - n$ must hold. This result was used on cycle bias calculation discussed in the next section.

Kruskal algorithm modifications for a non-perfect maze generation was done by adding some parts and changing some process in the original algorithm. The first modification is adding the additional k edges after a spanning tree is formed. This process was done by adding yet unselected edges to set A. In this study, such edges were collected in the set $Q = G.E - A$. This creates a connected graph with cycles, which is equivalent to a non-perfect maze.

Next step is introducing bias of a maze. As mentioned earlier, the two biases refer here are horizontal/vertical bias and cycle bias. The implementation of horizontal/vertical wall bias was done by calculating the number of active walls of the maze (w_a). If p_{hw} and p_{vw} ($p_{vw} = 1 - p_{hw}$) are the ratio of horizontal walls relative to the number of walls and the ratio of vertical walls relative to the number of walls, respectively, then the number of horizontal walls and the number of vertical walls can be written as

$$t_{hw} = [p_{hw} w_a] \qquad (1)$$

$$t_{vw} = w_a - t_{hw}.$$

The round function in equation (1) was intended so that t_{hw} and t_{vw} are both integers. If t_{he} and t_{ve} denote, respectively, the number of horizontal edges and vertical ones, then:

$$t_{he} = (n-1)m \qquad (2)$$

$$t_{ve} = (m-1)n \qquad (3)$$

Based on the result obtained, the number of horizontal walls and vertical walls of G_M that satisfy wall ratio p_{hw} and p_{vw} can be calculated using equations (2) and (3).

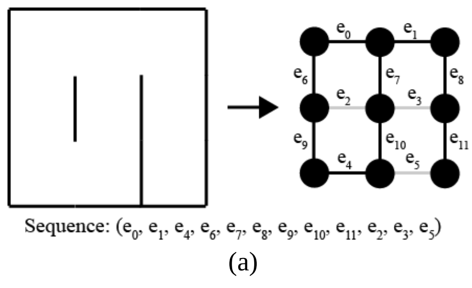
Next step is introducing the cycle bias of the maze. Cycle bias determines the number of walls removed, which is equal to the number of edges added. In this case, the value is the k we have previously defined. The addition of k edges to a spanning tree is equivalent to creating k cycle bases. A cycle basis is the cycle that cannot be formed using other cycles. In this study, the cycle bias is determined by first calculating the number of possible cycle bases in graph G_M. In a connected graph $G = (V, E)$, the number of possible cycle bases is $|E| - |V| + 1$. As a result, the maximal number of cycle bases of G_M is $2mn - m - n - (mn) + 1 = mn - m - n + 1$, which corresponds to maximal value of k. Consequently, if p_c denotes the ratio of cycle bases relative to the number of maximum cycle bases, then:

$$k = \lceil p_c(mn - m - n + 1) \rceil \qquad (4)$$

where $0 < p_c \leq 1$. The ceiling function in equation (4) was intended so that k is positive integer and $k > 0$.

In Kruskal algorithm, there exists an edge sorting process based on the edge's weight. Adding an edge in Kruskal algorithm is done based on the order of the edge obtained after the sorting. This is possible because the graph Kruskal algorithm works on is a weighted graph. However, in this study, G_M is not a weighted graph. Therefore, a method is needed to solve this problem. The technique used in this research was done by shuffling the edges of G_M using Fisher Yates algorithm. Fisher Yates algorithm shuffles a sequence with complexity of $O(n)$ where n is the number of elements needed to be shuffled. From the shuffled sequence of edges, we select $mn - 1$ edges just like in Kruskal algorithm, and select additional k edges by taking account of the bias, so that we will obtain a connected graph G_M with k cycle bases.

In this article, the addition of edges was done by referring to the order of the edge on the shuffled-sequence while taking account of the number of horizontal and vertical walls that must be present in the maze. However, not all shuffled-sequence can be used as reference of edges addition. This is because some sequences produce a disconnected graph. This case is illustrated in Figure 1.


e_0
e_1
e_6
e_7
e_8
e_2
e_3
e_9
e_{10}
e_{11}
e_4
e_5
Sequence: (e_0, e_1, e_4, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_2, e_3, e_5)
(a)

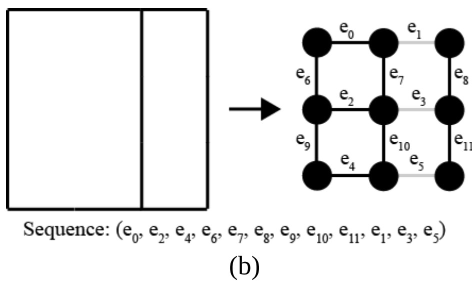

e_0
e_1
e_6
e_7
e_8
e_2
e_3
e_9
e_{10}
e_{11}
e_4
e_5
Sequence: (e_0, e_2, e_4, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_1, e_3, e_5)
(b)

Figure 2. (a) Sequence of edges producing connected graph (b) sequence of edges producing disconnected graph

Figure 2.a and 2.b both shows a graph with 3 horizontal edges and 8 vertical edges. In figure a), the edge sequence produced a connected graph while in figure b), it did not produce a connected graph. This is due to the fact that some columns of the graph have not been connected despite the graph satisfying the number of horizontal edges. The addition of e_0, e_2 and e_4 satisfy the number of horizontal edges (3 edges), but these three edges only connect first and second column of the graph. Therefore, a method is needed to solve this problem. In this study, the algorithm added edges between two adjacent columns and rows to ensure that each shuffled sequence of edges can produce a connected graph. Edges that connect adjacent rows and columns are selected to be added to the graph before adding sequence-based edges started.

In general, non-perfect maze generation algorithm produced in this study was formed by removing sorting process with shuffling edges in order to obtain a non-perfect maze. Just like Kruskal algorithm, each edge (u, v) was added if u and v belong to different disjoint sets. At this point, an additional condition was introduced to determine if an edge is valid or not to be added. The edge, whose endpoints belong to different disjoint set, would only be added if such addition does not violate the number of horizontal and vertical edges needed to satisfy the wall bias. These additions of edges eventually formed a spanning tree G_M. The next step was to add a number of edges to form cycles. At this stage, adding an edge did not take account of disjoint set partition of the endpoints, but only taking account of the number of horizontal and vertical walls. The overview of the algorithm can be found in Figure 3.

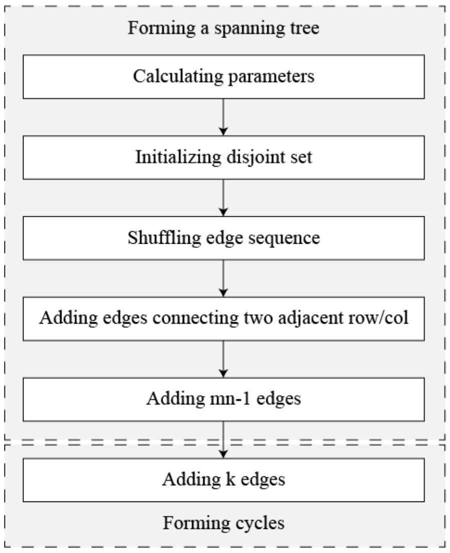


Figure 3. Overview of the algorithm

RESULTS AND DISCUSSION

The algorithm and the results are discussed in this section. This section also includes several categories of non-perfect mazes by varying the value of m, n and bias. For the first category, non-perfect mazes are generated by varying the value of m and n while keeping bias value fixed. Afterwards, the second category shows maze with fixed value of m, n, and cycle bias while varying wall bias. Finally, the third category shows mazes with fixed value of m, n, and wall bias while varying cycle bias.

Algorithm 2: Non-Perfect Maze Algorithm

```
1:  $A = \emptyset$
2:  $k = \lceil p_c(mn - m - n + 1) \rceil$
3:  $w_a = mn - m - n - k + 1$
4:  $t_{hw} = [p_{hw} w_a]$
5:  $t_{vw} = w_a - t_{hw}$
6:  $t_{he} = (n - 1)m - t_{vw}$
7:  $t_{ve} = (m - 1)n - t_{hw}$
8:  $c_h = 0$
9:  $c_h = 0$
10: for each vertex $v \in G.V$
11:   MAKE-SET(v)
12: SHUFFLE-EDGE-SEQUENCE
13: ADD-REQUIRED-EDGE
14: for each edge $e(u, v) \in G.E$, taken in random order
15:   if FIND-SET($u$) != FIND-SET ($v$)
16:     CHECK-EDGE (e($u, v$))
17: for each edge $e(u, v) \in Q = G.E - A$, taken in random order
18:   CHECK-EDGE ($e(u, v$)
19: return $A$
20: procedure CHECK-EDGE ($e(u, v)$)
21:   if $e$ is horizontal edge and $c_h < t_{he}$
22:       $A = A \cup \{e\}$
23:       UNION $(u, v)$
24:       $c_h + +$
25:   if $e$ is vertical edge and $c_v < t_{ve}$
26:       $A = A \cup \{e\}$
27:       UNION $(u, v)$
28:       $c_v + +$
29: procedure ADD-REQUIRED-EDGE(e($u, v$))
30:   for each column $i = 0$ until $n - 1$
31:     edge $e(u, v)$ is a random edge connecting column $i$ and column $i + 1$
32:     CHECK-EDGE($e(u, v$)
33:   for each row $i = 0$ until $m - 1$
34:     edge $e(u, v)$ is a random edge connecting row $i$ and row $i + 1$
35:     CHECK-EDGE($e(u, v$)
```

The algorithm produced based on the methodology mentioned earlier can be seen in Algorithm 2. In the algorithm, three helper functions were used, namely, SHUFFLE-EDGE-SEQUENCE, ADD-REQUIRED-EDGE and CHECK-EDGE. SHUFFLE-EDGE-SEQUENCE is used to shuffle sequence of edges using Fisher Yates algorithm. The addition of edges between adjacent columns and rows is done by ADD-REQUIRED-EDGE while iterative addition of edges is done in CHECK-EDGE. Line 1-9 is initialization and calculation of several desired maze parameters. The **for** loop in line 10-11 is initialization of disjoint set for each vertex. After edge shuffling in line 12, we add edges between two adjacent columns and rows in line 13. The iteration **for** in line 14 is started to form a spanning tree. In this iteration, each edge is checked if its endpoints belong to different disjoint set using FIND-SET. If yes, the edge is checked again using CHECK-EDGE. An edge will only be added if its addition will not violate the bias. The UNION is used to join two disjoint sets, which contains the endpoints of an edge. After the spanning tree is formed, we add k more edges so that the graph will contain cycles. This addition of edges will be done by **for** loop in line 17-18 using CHECK-EDGE. After the loop is done, the set A contains all edges belonging to G_M and the algorithm is now done.

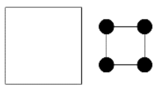
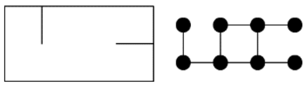
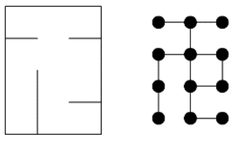
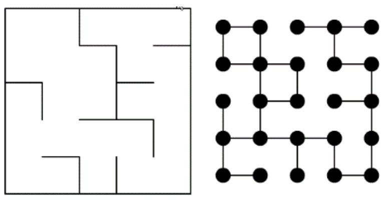
FIND-SET and UNION in disjoint set data structure with union by rank both have complexity of $O(\log n)$ where n is the number of elements in disjoint set. As with the algorithm, FIND-SET and UNION both have complexity of $O(\log V)$ where V is the number of vertices. Therefore, CHECK-EDGE has complexity of $O(\log V)$ and ADD-REQUIRED-EDGE of $O((m+n)\log V)$. The time complexity needed for parameters calculation in line 1-9 is $O(1)$. Disjoint set initialization of V times in line 10-11 takes $O(V)$. SHUFFLE-EDGE-SEQUENCE with Fisher Yates algorithm requires complexity of $O(E)$. The loop in line 14-16 is done E times so the total complexity of Find-Set is $O(\mathrm{E}\log V)$ and CHECK-EDGE is done $V-1$ times so total complexity CHECK-EDGE is $O(\mathrm{V}\log V)$. The addition of edges in loop line 17-18 takes complexity of $O(E\ \log V)$. Thus, the overall complexity of the algorithm is

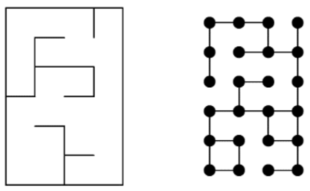
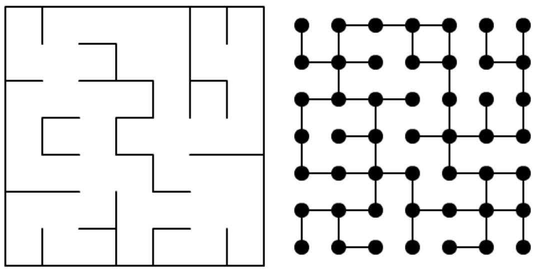
$$O(1+V+V+(m+n)\log V+\mathrm{E}\log V+\mathrm{V}\log V+E\ \log V)=O(\mathrm{E}\log V+\mathrm{V}\log V).$$

Since $|E|>|V|$, the complexity of the algorithm is $O(\mathrm{E}\log V)$.

Several mazes produced by the algorithm were shown below. Table 1 showed non-perfect mazes by varying its dimension and fixing the bias. The bias chosen here was 50% horizontal walls and 50% vertical walls with cycle bias 10%

Table 1. Mazes with m, n variations and wall and cycle bias fixed ($p_{hw}=0.5,\ p_{vw}=0.5,\ p_c=0.1$)

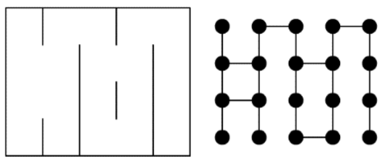
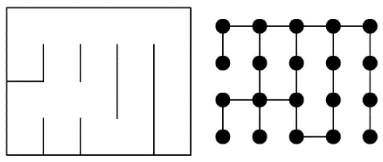
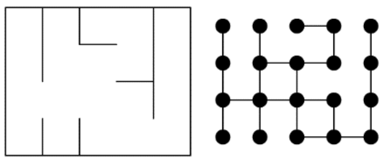
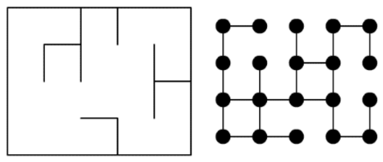
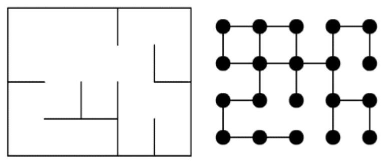
No	m	n	Non-perfect maze examples
1	2	2	
2	2	4	
3	4	3	
4	5	5	

5	6	4	
6	7	7	

Based on Table 1, it could be inferred that the non-perfect mazes generated had more passages as its size increases. This increase in size caused a maze to have more walls, so that the wall ratio was getting closer to what we desired.

Furthermore, several mazes were generated by varying wall bias. In this case, we varied wall bias 0-100% with an increment of 10%. All the mazes have fixed dimension of 4 × 5 with fixed cycle bias of 10%. The mazes generated with these variations were shown in Table 2.

Table 2. Mazes with m, n, cycle bias fixed and wall bias variations $(m = 4,\ n = 5,\ p_c = 0.1)$

No	Horizontal wall bias (%)	Vertical wall bias (%)	Non-perfect maze examples
1	0	100	
2	10	90	
3	20	80	
4	30	70	
5	40	60	

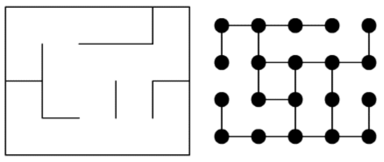
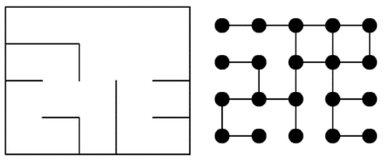
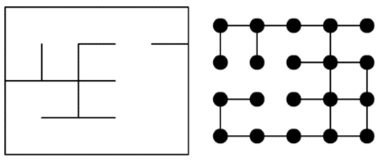
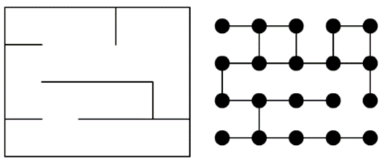
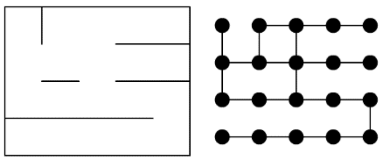
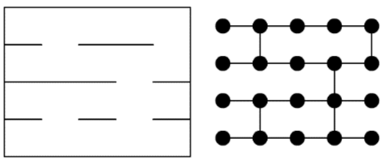
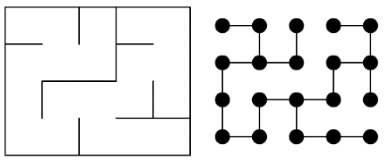
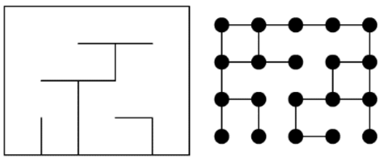
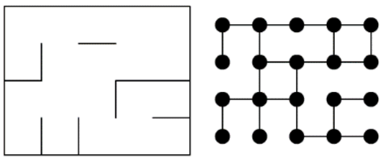
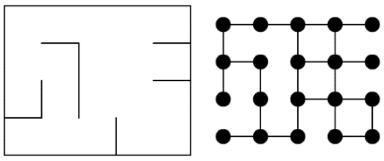
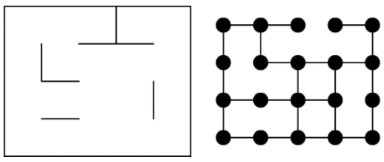
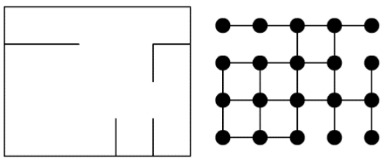
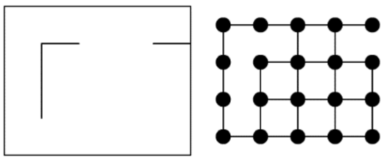
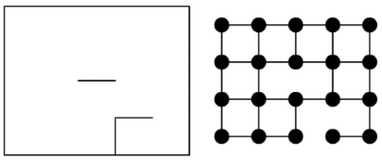
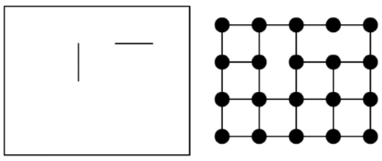
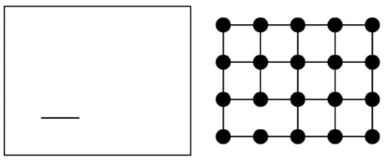
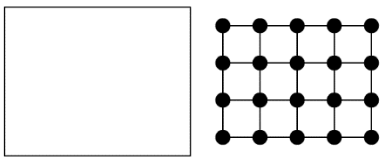
6	50	50	
7	60	40	
8	70	30	
9	80	20	
10	90	10	
11	100	0	

Table 2 showed that low horizontal bias generated a maze with low number of horizontal walls, vice versa. A 0% horizontal bias generated a maze with no horizontal walls, while a 100% generated a maze with no vertical walls.

The last parameter variation was cycle bias. The maze was generated by varying cycle bias and fixing its dimension and wall bias. The dimension we chose was 4×5 with 50% horizontal wall and 50% vertical wall. The variations of cycle were done by 10% increment. The results were shown in Table 3.

Table 3. Mazes with fixed value of m, n, and cycle bias variations ($m = 4,\ n = 5,\ p_{hw} = 0.5$, $p_{vw} = 0.5$)

No	Cycle bias (%)	Non-perfect maze examples
1	0	

2	10	
3	20	
4	30	
5	40	
6	50	
7	60	
8	70	
9	80	
10	90	
11	100	

In Table 3, it was seen that cycle bias of 0% generated a perfect maze, which is identical to a spanning tree G_M, while cycle bias of 100% generated a maze with no walls, which is identical to a grid graph G_M. We could also see that having greater cycle bias resulted in having more cycles in G_M. In general, the value of cycle bias determines the level of openness we described in the introduction section.

The three tables above only showed the result of variations of each parameter in non-perfect maze. However, these three parameters could be combined, resulting the non-perfect maze we desired.

CONCLUSION

Non-perfect maze generation is an interesting research when it includes the addition of several characteristics such as the size, the percentage of horizontal/vertical wall and the percentage of cycle. This study introduced an algorithm to generate an $m \times n$ non-perfect maze with the influence of horizontal/vertical wall bias and cycle bias and has overall complexity of $O(\mathrm{E} \log V)$ where V and E denotes, respectively, the set of vertices and edges of a grid graph $m \times n$. Possible further works includes reducing the time complexity, using another algorithm or introducing other biases and characteristics in a maze.

REFERENCE

[1] Kim, P.H.; Crawfis, R. 2015. The quest for the perfect perfect-maze. *2015 Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES)*. (Louisville: IEEE) 65–72.https://doi. org/10.1109/CGames.2015.7272964

[2] Haidar, A.; Fathoni, M.I.; Pu, M.; Ilham, M. 2016. Android maze game for children as an autism therapy. *Sci. J. PPI - UKM*. **3**(1), 22–25.

[3] Setiawan, W.; Hafitriani, S.; Prabawa, H.W. 2016. The scientific learning approach using multimedia-based maze game to improve learning outcomes. *AIP Conf. Proc. 050004*. https://doi.org /10.1063/1.4941162

[4] Fitzgerald, R..; Isler, R.; Rosenberg, E.; Oettinger, R.; Bättig, K. 1985. Maze patrolling by rats with and without food reward. *Anim. Learn. Behavior*. **13**(4), 451–462. https://doi.org/10.3758/BF03208022

[5] Saar, M.; Gilad, T.; Kilon-Kallner, T.; Rosenfeld, A.; Subach, A.; Scharf, I. 2017. The interplay between maze complexity, colony size, learning and memory in ants while solving a maze: A test at the colony level. *PLOS ONE*. **12**(8), 1–12. https://doi.org/10.1371 /journal.pone.0183753

[6] Jirout, J.J.; Newcombe, N.S. 2014. Mazes and Maps: Can Young Children Find Their Way? *Mind, Brain, Educ*. **8**(2), 89–96. https://doi.org/10.1111 /mbe.12048

[7] Pentland, L.M.; Anderson, V.A.; Dye, S.; Wood, S.J. 2003. The Nine Box Maze Test: A measure of spatial memory development in children. *Brain Cogn*. **52**(2), 144–154. https://doi.org/10.1016/S02782626(03)00 079-4

[8] Chou, W. 2016. Rectangular Maze Construction by Combining Algorithms and Designed Graph Patterns. *GSTF J. Comp*. **5**(1), 35–39.

[9] Kim, J. 2019. Maze Terrain Authoring System in Immersive Virtual Reality for New Visual Realism. *Sym*. **11**(4), 490.https://doi.org/10.3390/sym11040490

[10] Foltin, M. 2011. *Automated Maze Generation and Human Interaction*. (Brno: Masaryk University).

[11] Jonasson, A.; Westerlind, S. 2016. *Genetic algorithms in mazes: A comparative study of the performance for solving mazes between genetic algorithms, BFS and DFS*. (Stockholm: KTH Royal University).

[12] Pickover, C.A. 1992. *Mazes for the Mind: Computers and the Unexpected*. (New York: St. Martin’s Press).

[13] Dubey, P.; Sarita, K. 2016. Maze Generation & Solver. *Intern. J. Sci. Tech. Advanc*. **2**(4), 139–142.

[14] Singh, B.; Saxena, S.; Pandey, A.; Khulbe, P. 2014. Maze Generation Using Disjoint-Sets with Stack. *J. Basic App. Engineer. Res*. **1**(6), 19–21.

[15] Hendrawan, Y.F. 2018. A Maze Game on Android Using Growing Tree Method. *J. Phys.: Conf. Ser*. **953**(1), 012148. https://doi.org/10.1088/1742-6596/953/1/012148

[16] Pech, A.; Hingston, P.; Masek, M.; Lam, C.P. 2015. Evolving Cellular Automata for Maze Generation. *Australasian Conference on Artificial Life and Computational Intelligence*. (Newcastle: Springer International Publishing) p. 112–124. https://doi.org /10.1007/978-3-319-14803-8_9

[17] Lekova, M.; Boytchev, P. 2018. Virtual Learning Environment for Computer Graphics University Course. *12th Intern. Techn., Educat. Develop. Conf.* (Valencia: IATED) 3301–3309. https: //doi.org/10.21125/inted.2018.0633

[18] Shah, M.S.H.; Mohite, M.J.M.; Musale, A.G.; Borade, J.L. 2017. Survey Paper on Maze Generation Algorithms for Puzzle Solving Games. *Intern. J. Sci. Tech. Advanc.*. **8**(2), 1064–1067.

[19] Cormen, T.H.; E., C.; Leiserson, R.L.; Rivest; Stein, C. 2009. *Introduction to Algorithms*, 3 ed. (Cambridge: The MIT Press).