Hybrid Algorithm to Find Minimum Expected Escape Time From a Maze

Sheinna Yendri, Rully Soelaiman, *Member, IAENG*, and Yudhi Purwananto

***Abstract*—Expected value is one of the important factors used in scenario analysis and decision makings. The most commonly known method in finding expected value of a probability problem is using deterministic approach to formulate and solve a system of linear equations that represent the given problem. This method, nonetheless, requires unrealistic conditions for it to be executable which are knowing all the scenarios that might happen and the probability for each scenario to happen. In real life situations, most problems do not have a definite scenarios provided with each of their probabilities. This is a major loophole in deterministic approach for finding an expected value of certain problems. One specific probability problem that has this issue is to find the minimum expected time needed to escape from a two-dimensional maze without any information given for the escape path and the movement probabilities. In this paper, we propose a novel solution for the aforementioned problem using hybrid algorithm which is a combination of deterministic and heuristic approach. In our hybrid algorithm, the heuristic method is for optimizing the assembly of possible path scenarios, while the deterministic method is for counting and finding the minimum expected time for each path scenario. Based on the case study testing result, the solution using this hybrid algorithm requires an average time of only 2.4 seconds which is seven times faster than the required time limit, and an average memory of 5.31MB which is only using 0.3% resources from the required memory limit.**

***Index Terms*—expected value, hybrid algorithm, linear algebra, and path scenario.**

I. Introduction

EXPECTED value has a significant role in statistics because it describes where the probability is centered [1]. It indicates the anticipated value of an investment in the future which can help to decide whether an action is beneficial or detrimental [2]. In order to find an expected value, all possible scenarios and each of their probability must be provided [3]. This is the only way for an expected value to be calculated using a deterministic approach, such as linear algebra. However, most of real life problems do not have exact definition of all possible scenarios, let alone each probability. One particular problem that faces this issue is finding the minimum expected time needed to escape a two-dimensional maze without knowing any escape path and the movement probabilities. In this problem, a blind agent tries to escape a two-dimensional maze with movements limited to four directions, namely North, East, South, and West. Making a move to an adjacent cell takes exactly one unit of time. If the agent attempts to move out of the grid or attempts to move into an obstacle, he will stay in his current location but it will still cost him one unit of time. There is exactly one exit in the maze, located in one of the free cells. Since the agent is blind, to avoid confusion while making a move, his starting movement probabilities is fixed to four unknown real numbers for each of the direction (North, East, South, and West). Regardless of his position in the maze, he will always attempt to move North with probability P_N, East with probability P_E, South with probability P_S, and West with probability P_W. In other words, the sum of all four probabilities must equal to one. The challenge is to help the agent escape the maze as soon as possible with the following information provided; the agent starting position, the exit cell location, and the maze configuration itself. This paper will elaborate on how we approach this challenge to minimize the escape time by finding the optimal values of P_N, P_E, P_S, and P_W [4].

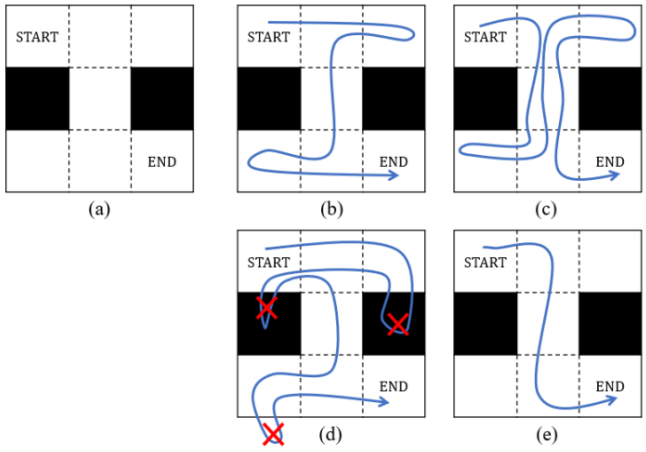

Fig. 1. Various path possibilities of a blind agent trying to escape from a maze. (a) Example of a 3×3 two-dimensional maze. (b) One possible escape path with inefficiency. (c) One possible escape path with more inefficiency. (d) One possible escape path where the agent stumbled to obstacles. (e) One possible escape path with the most efficiency.

One conventional way to approach this problem is to simply iterate all possible scenarios by trying all possible movement probabilities one at a time. From there, for each scenario, the expected escape time can easily be obtained using deterministic method. However, this naïve approach is actually impossible to be done, knowing that the blind maze-escapee's movement probabilities are real numbers which have infinite possible values. Even if we limit the precision to nine decimal places, the time complexity will be $\mathcal{O}(10^{27})$ and takes 10^{18} seconds which is equivalent to 31.7 billion years, and thus not an ideal solution for real life

Manuscript received July 14, 2022; revised January 12, 2023. This work was supported in part by Sepuluh Nopember Institute of Technology, Surabaya, Indonesia.

Sheinna Yendri is a graduate student at Sepuluh Nopember Institute of Technology, Department of Informatics, Surabaya, Indonesia (e-mail: 6025212005@mhs.its.ac.id)

Rully Soelaiman is an associate professor at Sepuluh Nopember Institute of Technology, Department of Informatics, Surabaya, Indonesia (e-mail: rully130270@gmail.com)

Yudhi Purwananto is an associate professor at Sepuluh Nopember Institute of Technology, Department of Informatics, Surabaya, Indonesia (e-mail: purwananto@gmail.com)

applications. Figure 1 shows the various possibilities of a blind agent trying to escape from a 3×3 two-dimensional maze (Figure 1(a)). In Figure 1(d), it is shown that the path taken by the agent might stumble to obstacles or walls before finally reaching the destination cell. Figure 1(b)(c)(e) shows other path possibilities that might be either an efficient or a inefficient one.

By taking previous researches with similar topics of process optimization problem solving as reference [5] [6] [7] [8], it can be deduced that merging heuristic approach with deterministic approach makes this maze-escaping problem possible to solve. Therefore, in this paper, we proposed a novel solution that only needs an average time of 2.39 seconds to count the minimum expected escape time from a maze sized 8×8, which is approximately four hundred million times faster compared to the conventional way and satisfies the problem time limitation of 16 seconds. This result can be achieved by using deterministic and heuristic hybrid approach to optimize the escape path finding and also count the expected escape time.

The rest of the paper is organized as follows: Section II presents our novel hybrid method: the combination of deterministic and heuristic methods, which is proposed to find the minimum expected escape time. Section III presents the experimental results and analysis. Finally, the conclusion is stated in Section IV.

II. Methodology

The hybrid algorithm that we propose as a solution to find the minimum expected escape time from a two-dimensional maze is a combination of deterministic and heuristic methods. Similar to how it is used in naïve approach, the deterministic method is for calculating the expected escape time by solving a system of linear equations. On the other hand, the heuristic method holds the key to overcome the issue of finding the most optimal escape path probability that was not achievable in naïve approach. In general, the hybrid algorithm is divided into three main steps:

1) Determine movement probability values randomly (or intuitively) as the starting point that will be used in the deterministic part.
2) Calculate the expected escape time by solving the linear equation system formed by the predefined movement probabilities.
3) Improve the expected escape time until the minimum value is achieved by updating the movement probabilities according to the objective function and recalculate the expected escape time.

The three main steps above are described in more details through the flowchart shown in Figure 2.

Taking into account that hybrid algorithm is constructed by two major parts: deterministic and heuristic, the following discussion will also consisted of two major sections to elaborate each method, Section II-A and Section II-B, and closed with Section II-C which will show how these two parts are combined and used alternately to find the minimum expected escape time.

A. Deterministic part

As mentioned in the Section II, the deterministic part is for computing the expected escape time. To compute the

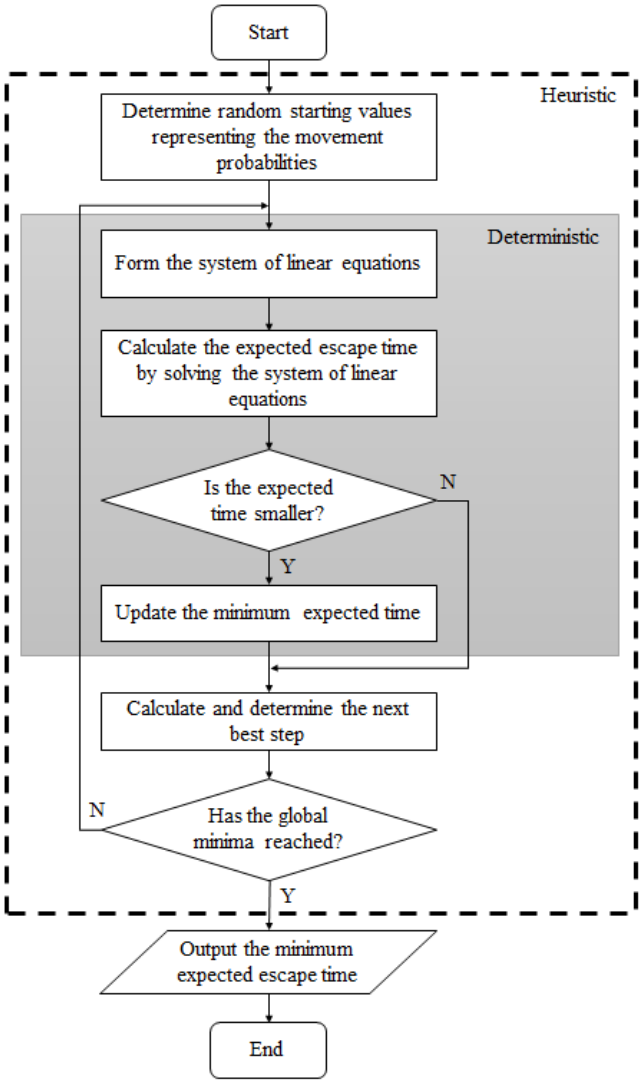

Fig. 2. Flowchart of the proposed hybrid algorithm. This algorithm will keep running until the global minima is reached, which means the minimum expected escape time is already obtained.

expected value of a certain problem, we must know the possible scenarios with its respective probabilities. Since this problem does not have any of the required information for expected value calculation, we will first assume that all possible movement probabilities made to escape the maze are already obtained. Therefore, in this section, we can eliminate the needs to find the optimal movement probability, because it will be covered in Section II-B. Starting from this point, we will assume that P_N, P_E, P_S, and P_W are valid movement probabilities, which satisfy both equations (1) and (2).

$$0 \leq P_N \leq 1, 0 \leq P_E \leq 1, 0 \leq P_S \leq 1, 0 \leq P_W \leq 1 \quad (1)$$

$$P_N + P_E + P_S + P_W = 1 \quad (2)$$

Expected value is calculated by using the probability distribution which is obtained from a multiplication of each possible outcomes by its occurrence likelihood and then summing all of those values [1]. Generally, expected value of the random variable X is denoted as $E(X)$, which is

formulated in equation (3) if X is discrete, and in equation (4) if X is continuous.

$$E(X) = \sum_{x} xf(x) \tag{3}$$

$$E(X) = \int_{-\infty}^{\infty} xf(x)\ dx \tag{4}$$

In calculating the expected value of time to escape from a two-dimensional maze, the possible outcomes depend on the movement possibilities. From one cell to another, we have multiple scenarios that might happen and each of it has different probabilities to happen. Thus, to calculate the expected escape time for this problem, we can focus on calculating the expected time based on each cell. Each cell's expected value will influence the expected values of its neighboring cell from all four directions. The reason behind this is because the expected value represents the total escape time that will be added as the agent travels through the maze. This process is illustrated in Figure 3. Each cell will have four possible next steps, namely the North, East, South, and West with each direction has its own probability to be chosen. As mentioned before, expected time is calculated by multiplying each possible scenario with its likelihood to occur and then summing all values. Hence, for this problem, the expected time for each cell (row, col) can be defined in equation (5), where row and col represent the current position of the agent in the two-dimensional maze. This equation (5) is derived from equation (3) as our problem is having a discrete random number which is the cell position, where x is represented by the expected value of adjacent cells (e.g. $E(row-1, col)$), and $f(x)$ is represented by the movement probability to reach that adjacent cell (e.g. P_N). Moreover, since the expected value is dependant on the neighboring cells, the expected value calculation is not done once only, but multiple times for each cell in each iteration.

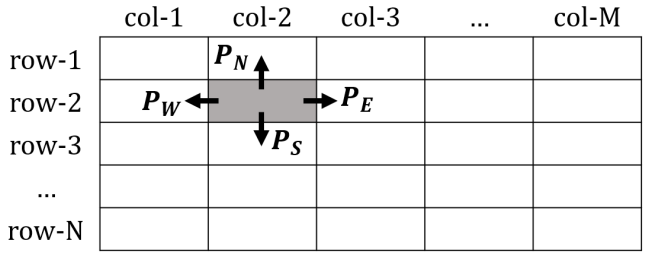

Fig. 3. Illustration of each cell's expected value findings. The grids are representing the maze cells, with each cell (row, col) indicates the agent's position and has four possible movements.

$$\begin{aligned} E(row, col) = & P_N \times E(row-1, col) + \\ & P_E \times E(row, col+1) + \\ & P_S \times E(row+1, col) + \\ & P_W \times E(row, col-1) + 1 \end{aligned} \tag{5}$$

Since we are calculating the expected escape time from the perspective of each cells, we will assign $N \times M$ as the total cells that need to be calculated. Thus, when we construct the linear equation system, we will have a total of NM equations with each equation consists of NM variables. In other words, if we represent the linear equation system using two-dimensional matrix, we will have a $NM \times NM$ sized matrix. In our problem, the maximum maze to be solved is only sized 8×8, meaning the largest matrix we can get is only sized 64×64 and both memory and time-wise is still sufficient. For further optimization, we can reduce the matrix size by forming it only from the visited cells of the maze, not using all $N \times M$ cells. Once we construct the matrix, we can transform the matrix into a reduced row echelon form and find the expected escape time by solving the linear equations of the matrix. The steps included in this process will be further discussed in section II-A1 to II-A3.

1) Traversing the Maze: Before going into further elaboration on the equation formulation part, we first need to traverse the maze to locate which cells are visited while giving identification numbers (IDs) to each cell. The point of doing this step is to count total visited cells which will be used later on when constructing the matrix. It is important to mark each cell with ID numbers as it will help in organizing the linear equation formulations.

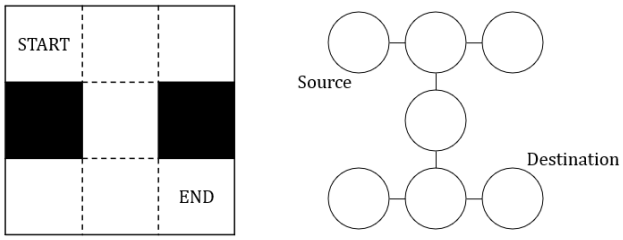

Fig. 4. Example of transforming a 3×3 maze into a graph so it is easier to be traversed. Figure on the left side is showing the example 3×3 maze, while figure on the right side is showing the graph representation of the maze.

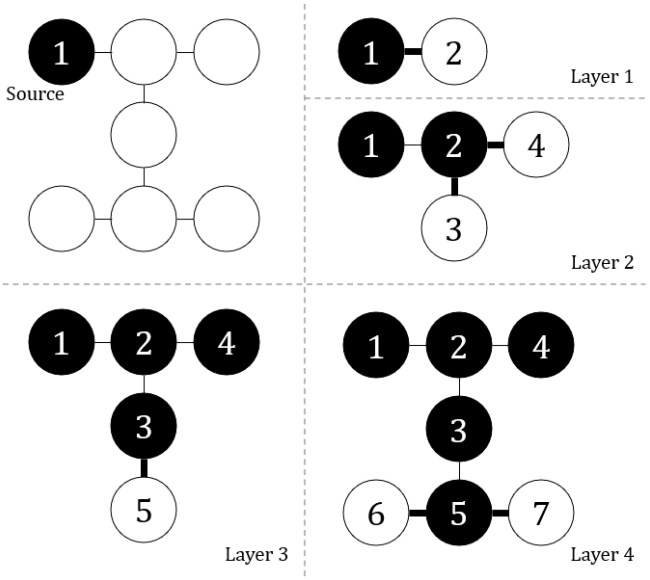

Fig. 5. Illustration of traversing a 3×3 maze using BFS. In BFS, the traversing works layer by layer, starting from the root (source) to its neighbors and so on until all nodes are visited.

A maze can be represented as a graph $G = (V, E)$ with its cells as the vertices (V) and the connection between cells as the edges (E) [9]. An example of representing a maze using a graph can be seen in Figure 4. There are various ways

to traverse a graph, for this case, we are using breadth-first search (BFS) method which is one of the simplest algorithms for searching a graph, especially in finding the shortest paths. Starting from a distinguished source vertex, BFS will traverse the graph 'breadth-first': vertices that are direct neighbors of the source vertex (first layer) will be visited first, then continue to visit the neighbors of direct neighbors (second layer), and so on, layer by layer, until all vertices are visited [10]. Illustration of this maze traversing process using BFS algorithm for an example 3×3 maze is shown in Figure 5.

2) Constructing the Matrix: After traversing the entire maze and marked the visited cells, we can start constructing the matrix. From this point, the discussion for this section will be carried on using the 3×3 example maze shown in Figure 4. From Figure 5 we know that there are 7 visited cells in the maze, numbered from one to seven. Meaning, we will need to construct a 7×7 matrix to represent a system of 7 linear equations of the expected escape time. For our example case, since we are constructing a 7×7 matrix, the basic matrix equation is shown in equation (6)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} \tag{6}$$

Now, we will modify the matrix values adjusted to the maze configuration and the movement probabilities we previously defined i.e. P_N, P_E, P_S, and P_W. We will update all a_{ij} values according to the movement probability from cell i to j, except for the destination cell itself, shown in equation (7). For example, a_{11} equals to $1 - P_N - P_S - P_W$, since a_{11} represents movement from cell 1 to cell 1. In other words, the agent stays in the same position which is caused by an attempt of moving into an obstacle or a wall. Looking at the maze configuration, this can only happen when the agent tries to move towards North, South, and West. Another example, a_{21} equals to $-P_W$, since the only way to move from cell 2 to cell 1 is by moving towards West. One unique characteristic of this equation is that a_{ii} values will always start with 1 before subtracting them with the probability movements because the matrix is initially an identity matrix.

$$\begin{aligned} a_{11} &= 1 - P_N - P_S - P_W & a_{12} &= -P_E \\ a_{21} &= -P_W & a_{22} &= 1 - P_N \\ a_{23} &= -P_S & a_{24} &= -P_E \\ a_{32} &= -P_N & a_{33} &= 1 - P_E - P_W \\ a_{35} &= -P_S & a_{42} &= -P_W \\ a_{44} &= 1 - P_N - P_E - P_S & a_{53} &= -P_N \\ a_{55} &= 1 - P_S & a_{56} &= -P_W \\ a_{57} &= -P_E & a_{65} &= -P_E \\ a_{66} &= 1 - P_N - P_S - P_W \end{aligned} \tag{7}$$

The x_i value matrix from equation (6) is a column vector of the 7 unknowns that we are looking for. While the right-hand side matrix is another column vector that will be filled by a vector of ones, except the destination cell that will be filled by a zero. This right-hand side matrix represents the expected cost of moving between cells, which is one unit of time per movement. The reason for putting zero value in the destination cell is because we are looking for the expected escape time **to reach** the destination cell. As a result, if we assume $P_N = 0.3, P_E = 0.3, P_S = 0.2$, and $P_W = 0.2$, we will get the final matrix equation as shown in equation (8).

$$\begin{bmatrix} 0.3 & -0.3 & 0 & 0 & 0 & 0 & 0 \\ -0.2 & 0.7 & -0.2 & -0.3 & 0 & 0 & 0 \\ 0 & -0.3 & 0.3 & 0 & -0.2 & 0 & 0 \\ 0 & -0.2 & 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & -0.3 & 0 & 0.8 & -0.2 & -0.3 \\ 0 & 0 & 0 & 0 & -0.3 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{8}$$

3) Solving the System of Linear Equations: In this final section, the only remaining step is to solve the linear equation system, represented by the matrix equation constructed in Section II-A2. We define a matrix equation as stated in equation (9), where A is the coefficient matrix filled with the movement probabilities, x is the variable matrix which values represent the expected time to reach the destination cell, and z is the constant matrix filled with the amount of time needed to move from one cell to another. Time complexity to solve this matrix equation is $\mathcal{O}(n^2 \cdot m)$, where n is the number of rows of matrix A and m is the number of rows of matrix x. Thus, the maximum time complexity using this method is only $\mathcal{O}(64^2 \cdot 64)$ (equivalent to 0.1 millisecond) when the size of the maze is 8×8.

$$A \cdot x = z \tag{9}$$

Matrix equation shown in equation (8) is representing the linear equation system using the movement probabilities of $P_N = 0.3, P_E = 0.3, P_S = 0.2$, and $P_W = 0.2$. Since in that example we are starting from cell 1, and our destination is cell 7, the expected escape time will be stored in x_1, the first element from the variable matrix. To sum up, the overall algorithm of this deterministic method is summarized as follows in Algorithm 1 and 2. Algorithm 1 mainly shows how the matrix equation formulation process is done, and then calls SOLVELINEAR function which is Algorithm 2.

B. Heuristic part

As discussed earlier in Section II-A, the movement probabilities were assumed to be P_N, P_S, P_W, P_E first as these probabilities values are obtained using heuristic method which will be further explained in this section. When trying to escape from the maze, the agent has four movement directions, namely to North, East, South, and West. In each second, the agent must make a move to one of its four neighbor cells and the probabilities of the agent moving to these four directions must fulfill equation (2) from Section II-A. Hence, when looking for the optimal movement probabilities, we only need to find three probability values and obtain the remaining value by solving equation (2). In our solution, we use this heuristic approach to find P_N, P_S, P_W, and use equation (2) to solve P_E accordingly. If the precision is limited to only nine decimal places, this means we will have 10^9 probabilities for each direction. As a result, in total we

Algorithm 1 Calculate the expected escape time

Input: $vis, prob$
Output: ans

$dy[4] \leftarrow -1, 1, 0, 0$
$dx[4] \leftarrow 0, 0, -1, 1$
$INFDB \leftarrow 999999.99$
for each y, x, id in vis **do**
  $A[id][id] \leftarrow A[id][id] + 1.0$
  if $id = exitId$ **then**
    $z[id] \leftarrow 0.0$
    CONTINUE
  end if
  $z[id] \leftarrow 1.0$
  for $i \leftarrow 0$ **to** 3 **do**
    $ny \leftarrow y + dy[i]$
    $nx \leftarrow x + dx[i]$
    if !IsValidCoordinate(ny, nx) **then**
      $ny \leftarrow y$
      $nx \leftarrow x$
    end if
    $nextId \leftarrow vis[ny][nx].id$
    $A[id][nextId] \leftarrow A[id][nextId] - prob[i]$
  end for
end for
$ans \leftarrow$ SolveLinear(A, z)
if $ans = -1$ **then return** $INFDB$
else return ans
end if

have 10^{27} movement probabilities which takes approximately 31.7 billion years to be traversed. This does not satisfy the problem time limitation of 16 seconds. Thus, heuristic approach is required to optimize the movement probabilities finding.

There are plentiful of options in the field of utilising heuristic approach for optimization process, such as hill climbing, simulated annealing, genetic algorithm [11] [12], and swarm intelligence [13] [14]. For this problem, hill climbing is the best option because of its simplicity and thus does not require a lot of resources both timewise and spacewise. It is simply a loop that gradually moves in the direction of decreasing value, since we are looking for the minimum expected value. It terminates when it reaches a "peak" where no neighbor coordinates has a lower value. Hill climbing often makes a rapid progress towards the solution because it is quite easy to recover from a bad state while doing iteration hops [15]. However, hill climbing has its drawback when facing local maxima, ridges, or plateaux. In those cases, hill climbing will most probably get stuck and cannot find the global optimal solution. Fortunately, in our problem, the generated 3D-graphs between expected escape time and movement direction probabilities are all concave graphs as displayed in Figure 6, 7, and 8. Since there are 3 movement probabilities required, but only 2 movements can be represented in each graph, the relationship between expected escape time and the movement probabilities is represented in $\binom{3}{2} = 3$ graphs. Figure 6 represents the graph relationship between the movement probabilities and the expected escape time when probability to move South (P_S) is fixed, Figure 7 is when probability to move West

Algorithm 2 Solve the system of linear equations

Input: A, z
Output: EV

$n \leftarrow A.size()$
$m \leftarrow z.size()$
$rank \leftarrow 0$
$EPS \leftarrow 10^{-7}$
for $i \leftarrow 0$ **to** n **do**
  $val, zval \leftarrow 0$
  for $row \leftarrow i$ **to** n **do**
    for $col \leftarrow i$ **to** m **do**
      if $(val = |A[row][col]|) > zval$ **then**
        $zrow = row$
        $zcol = col$
        $zval = val$
      end if
    end for
  end for
  if $zval \leq EPS$ **then**
    for $j \leftarrow i$ **to** n **do**
      if $|z[j]| > EPS$ **then**
        return -1
      end if
    end for
    BREAK
  end if
  SWAP$(A[i], A[zrow])$
  SWAP$(z[i], z[zrow])$
  SWAP$(col[i], col[zrow])$
  for $j \leftarrow 0$ **to** n **do**
    SWAP$(A[j][i], A[j][zcol])$
  end for
  $zval \leftarrow \frac{1}{A[i][i]}$
  for $j \leftarrow i + 1$ **to** n **do**
    $fac \leftarrow A[j][i] * zval$
    $z[j] \leftarrow z[j] - (fac * z[i])$
    for $k \leftarrow i + 1$ **to** m **do**
      $A[j][k] \leftarrow A[j][k] - (fac * A[i][k])$
    end for
  end for
  $rank \leftarrow rank + 1$
end for
$EV \leftarrow -1$
for $i \leftarrow rank$ **downto** -1 **do**
  $z[i] \leftarrow \frac{z[i]}{A[i][i]}$
  $x[col[i]] \leftarrow z[i]$
  if $col[i] = startId$ **then**
    $EV \leftarrow x[col[i]]$
    BREAK
  end if
  for $j \leftarrow 0$ **to** i **do**
    $z[j] \leftarrow z[j] - (A[i][j] * z[i])$
  end for
end for
return EV

(P_W) is fixed and Figure 8 is when probability to move North (P_N) is fixed. These three figures are using an example of escaping from a maze sized 3×3.

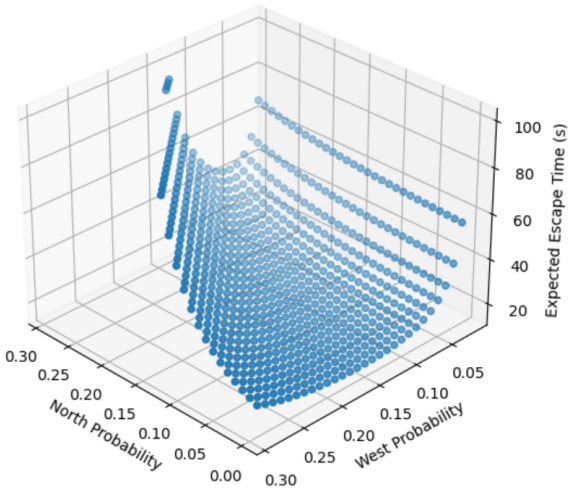

Fig. 6. Relationship between expected escape time from a 3×3 maze when P_S is fixed

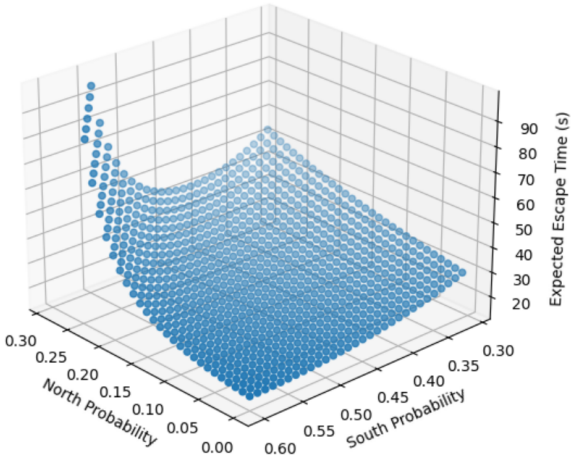

Fig. 7. Relationship between expected escape time from a 3×3 maze when P_W is fixed

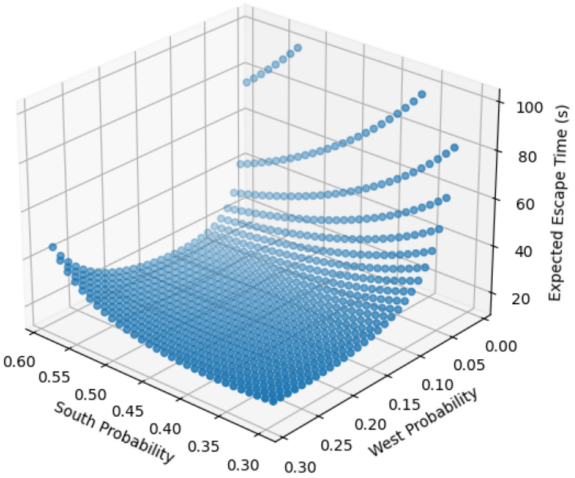

Fig. 8. Relationship between expected escape time from a 3×3 maze when P_N is fixed

Based on the concave characteristics of expected escape time against movement probabilities graphs, we can conclude that hill climbing method is applicable for optimizing this maze escape movement probabilities searching process. It is the simplest and fastest heuristic optimization method that does not need much resources to obtain the most optimal solution. The initial step of hill climbing is to set a starting point. Adopting random probabilities as starting points are the general practice for this method, but there is another better way by choosing intuitively based on the maze's source and destination cells which will be used for this paper proposed solution. Next, an objective function needs to be determined. Objective function is a function that needs to be minimized or maximized based on the problem [16]. In this problem, our objective is to find the minimum expected escape time from the maze. Hence, the objective function for this problem is defined as equation (5). By using the expected value formula as our objective function, the next iteration state value will always produce movement probabilities that has faster expected escape time. Finally, the hill climbing iteration will terminate when reaching the stopping criteria, in which there is no better next state (all neighbor cells cannot produce faster expected escape time). This condition occurs when the solution is finally convergent and there is no better solution exists.

Algorithm 3 Our proposed hybrid algorithm

Input: $maze$
Output: ans

1: $EPS \leftarrow 10^{-9}$
2: $vis \leftarrow$ TRAVERSEMAZE$(maze)$
3: $p[0], p[1], p[2], p[3] \leftarrow$ GENERATESTARTINGPOINT()
4: $ans \leftarrow$ COMPUTEEV(vis, p)
5: **for** $i \leftarrow 0.1$ **downto** EPS **step** $*0.5$ **do** ▷ hill climbing
6: **for** $j \leftarrow 0$ **to** 2 **do**
7: **for** $a \leftarrow -1$ **to** 1 **do**
8: **for** $b \leftarrow -1$ **to** 1 **do**
9: **for** $c \leftarrow -1$ **to** 1 **do**
10: $pp[0] \leftarrow p[0] + i * a$
11: $pp[1] \leftarrow p[1] + i * b$
12: $pp[2] \leftarrow p[2] + i * c$
13: $pp[3] \leftarrow 1 - pp[0] - pp[1] - pp[2]$
14: **if** ISVALIDPROBABILITY(pp) **then**
15: $res \leftarrow$ COMPUTEEV(vis, pp)
16: **if** $res < ans$ **then**
17: $res \leftarrow ans$
18: **for** $k \leftarrow 0$ **to** 3 **do**
19: $p[k] \leftarrow pp[k]$
20: **end for**
21: **end if**
22: **end if**
23: **end for**
24: **end for**
25: **end for**
26: **end for**
27: **end for**
28: **return** ans

In line 4 and 15, COMPUTEEV function is called to compute the expected escape time (the deterministic method).

C. Hybrid algorithm

In the previous sections, we already discussed on how deterministic and heuristic methods are being used independently to solve this problem in parts. In this section, we will now elaborate on how these two different methods can be used together alternately in solving the problem as a whole. It has been explained from Section II-A that deterministic part is able to provide the expected escape time as long as the movement probabilities are known. On the other hand, Section II-B shows that heuristic part provides the optimal movement probabilities. From these two statements, it can be concluded that both methods (deterministic and heuristic) actually depend on each other. First, heuristic method is utilised to give an intuitively determined movement probabilities (P_N, P_E, P_S, P_W) as the starting point of using deterministic method for the next part of this optimized problem-solving hybrid algorithm. Once the initial movement probabilities are set, deterministic method is applied to process and calculate the expected escape time using linear algebra. Then, the expected escape time obtained from this deterministic method is processed again by heuristic method to determine the next suitable state. The suitable next state is retrieved by finding which movement probabilities produce the least expected escape time out of all the possible next states. This sequence of steps is executed back and forth into one loop of process until all possible next states result in the same expected escape time, which fulfilled the stopping criteria. This loop of process sequence born from a hybrid of deterministic and heuristic approach can be translated into an algorithm shown in Algorithm 3.

III. Experimental Results and Analysis

In the previous section, we have explained how we approach this maze problem using this novel hybrid algorithm of deterministic and heuristic methods. All proposed algorithms in this paper were implemented using C++11 programming language. Therefore, in this section, we will examine this algorithm by using various testing platforms suitable for analysing two important aspects of C++11 program which are its validity and performance. The validity of this hybrid approach was tested by submitting the source code on Sphere Online Judge which is a third-party online platform for source code checker, that uses Cube cluster with Intel Xeon E3-1220 v5 CPUs [17]. The online judge platform examined the submitted source code validity by comparing its result output to the expected answers provided by the problem originator. On the other hand, two environment testing platforms were used to evaluate the performance of the source code. The first platform on a live environment is using Sphere Online Judge's feedback to assess both memory and time consumption compared with all previous solutions. While the other platform is a local environment using a PC with Intel® Core™ i7-1165G7 4 Core Processor and 8192 MB of memory, running Windows 10 with GCC 7.50 compiler, to compare and analyze the program runtime and how efficient the heuristic part works in different maze sizes and complexities scenario.

In Section III-A, we will elaborate on the validity check of the hybrid method that is used to calculate the minimum expected escape time from a two-dimensional maze. Moreover, in Section III-B we will further go into detail about the performance examination of the proposed method, both timewise and spacewise. Lastly, in Section III-C we will analyze how efficient the heuristic method works in different maze configurations and sizes.

A. Validity Examination

Figure 9 shows how the hybrid approach used as explained in Section II are scored on Sphere Online Judge. Sphere Online Judge's testing system will give various response statuses based on its judgement to the solution submitted. **"Accepted"** status indicates that the program ran successfully and gave a correct answer, **"wrong answer"** status means the program ran successfully, but gave an incorrect answer, **"time limit exceeded"** shows that the program was compiled successfully, but it exceeded time limit, **"compilation error"** means that the program could not be compiled, and lastly **"runtime error"** status implies that the program was compiled successfully, but it exited with a runtime error or crashed [18].

The code testing submissions were executed at least 10 times to ensure not only its accuracy but also its consistency throughout this validity test. All ten submissions received **"accepted"** responses which proves that our approach to calculate minimum expected escape time using the fusion of linear algebra and hill climbing methods can provide correct answers within the time and memory limitation. For each submission, our program was challenged with multiple hidden test cases configured by the problem originator. **"Accepted"** status is given only when the code passes all test cases which proves how strongly valid our solution is. Moreover, as seen from Figure 10 only 5 out of 99 submitted solutions are accepted by the Sphere Online Judge and our proposed hybrid solution is one of them. This not only shows how complex the problem is, but also how powerful our proposed solution is.

B. Performance Examination

There are two factors that must be taken into considerations in performance examination, they are program runtime and memory usage [19]. The first factor, program runtime, will be tested in both local, using own PC, and live environment, with the help of Sphere Online Judge site. As for the second factor, memory usage, will only be evaluated in Sphere Online Judge site.

1) Runtime: In evaluating an algorithm's performance, it is essential to use worst-case scenarios to determine whether the algorithm is still acceptable and running well even in unfavorable situations. Therefore, in order to evaluate the program runtime, two types of maze were used: maze with no blockers and spiral-like maze. Maze with no blockers configuration is considered as the worst case because algorithm pruning will take no effect in this maze type. Algorithm pruning is effective for mazes which consist of a lot of blockers that means the amount of visited cells are reduced and so does the program runtime. However, this is not the case in no blocker mazes. On the other hand, spiral-like maze may not be the worst possible case but it serves as a special case with an interesting problem to be evaluated, because in determining the expected escape time, we need to know the most optimal movement probabilities beforehand. In the case

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
29797023	2022-07-11 16:19:26	Sheinna	Not So Blind Escape	**accepted**	2.37	5.3M	CPP14
29796855	2022-07-11 15:32:36	Sheinna	Not So Blind Escape	**accepted**	2.39	5.4M	CPP14
29796749	2022-07-11 15:06:50	Sheinna	Not So Blind Escape	**accepted**	2.43	5.3M	CPP14
29796747	2022-07-11 15:06:21	Sheinna	Not So Blind Escape	**accepted**	2.37	5.2M	CPP14
29796501	2022-07-11 13:42:23	Sheinna	Not So Blind Escape	**accepted**	2.85	5.3M	CPP14
29795795	2022-07-11 10:30:46	Sheinna	Not So Blind Escape	**accepted**	2.50	5.4M	CPP14
29795535	2022-07-11 09:22:47	Sheinna	Not So Blind Escape	**accepted**	2.39	5.3M	CPP14
29795526	2022-07-11 09:21:43	Sheinna	Not So Blind Escape	**accepted**	2.39	5.4M	CPP14
29795515	2022-07-11 09:19:38	Sheinna	Not So Blind Escape	**accepted**	2.39	5.3M	CPP14
29681233	2022-06-13 11:40:20	Sheinna	Not So Blind Escape	**accepted**	2.48	5.2M	CPP14

Fig. 9. Validity Test Examined by Sphere Online Judge

Users accepted	Submissions	Accepted	Wrong Answer	Compile Error	Runtime Error	Time Limit Exceeded
5	99	37	41	5	9	7

DATE	USER	RESULT	LANG
2022-07-11 15:06:21	Sheinna	**accepted**	CPP14
2022-06-03 09:26:42	Rully Soelaiman	**accepted**	CPP14
2021-02-18 06:55:45	rama_pang	**accepted**	CPP14
2020-06-20 08:33:54	suhash	**accepted**	CPP14
2019-11-18 05:02:51	[Rampage] Blue.Mary	**accepted**	CPP

Fig. 10. Statistics of all previously submitted solutions and list of accepted users examined by Sphere Online Judge

of spiral-like maze, the escape path must also have a spiral-like path and thus the movement probabilities between each directions should be quite similar. With similar movement probabilities, the agent will need the longest time to reach the destination cell, thus it can be referred as the worst case if the focus of the challenge leans more towards escaping the maze quickly rather than saving program runtime to find the most optimal escape route. Illustration of a 8×8 spiral-like maze is shown in Figure 11. After choosing two worst case test scenario, we would like to know how the trend of the program runtime for each test case looks like with expanding maze size. For this performance evaluation, we chose $N \times N$ maze size with one unit of increase starting from $N = 2$ to 8. Table I and II show the program runtime when solving no blocker maze and spiral-like maze. The data above were obtained by executing the implemented approach on a local PC ten times for each maze type.

Figure 12 shows the scatter chart of every ten trials' average runtime value in no blocker mazes and spiral-like mazes with different sizes. It is shown that the program runtime in both no blocker maze and spiral-like maze are exponentially increasing with the maze size. Especially for no blocker mazes, the trend is growing much more significantly compared to spiral-like mazes. This is due to the fact that the proposed hybrid method is highly dependant on the maze cells that can be visited. Hence, no blocker maze will obviously have more cells to be examined, compared to spiral-like mazes. In 8×8 no blocker maze, the program ran

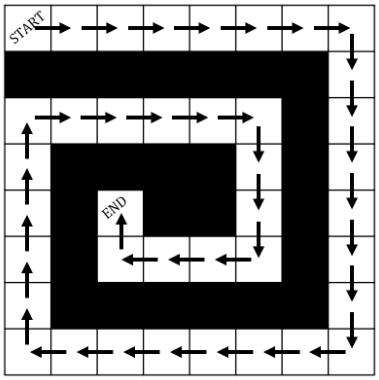

Fig. 11. Illustration of spiral-maze sized 8×8. It is shown that the escape path consists of 6 steps to the North, 12 steps to the East, 10 steps to the South, and 10 steps to the West. Number of steps needed for each direction are more or less the same, thus the optimal movement probabilities are nearly 0.25 for each direction.

TABLE I
PROGRAM RUNTIME IN NO BLOCKER MAZES (IN SECONDS)

#	Maze Size						
	2×2	3×3	4×4	5×5	6×6	7×7	8×8
1	0.002	0.007	0.022	0.067	0.170	0.386	0.845
2	0.002	0.007	0.022	0.103	0.296	0.890	1.236
3	0.003	0.007	0.027	0.085	0.188	0.433	1.031
4	0.002	0.008	0.032	0.122	0.244	0.457	0.965
5	0.002	0.008	0.023	0.070	0.156	0.388	0.927
6	0.003	0.009	0.032	0.086	0.199	0.539	1.031
7	0.003	0.011	0.046	0.088	0.204	0.524	1.015
8	0.002	0.007	0.024	0.068	0.218	0.489	0.971
9	0.005	0.012	0.036	0.077	0.207	0.461	1.283
10	0.003	0.009	0.040	0.076	0.157	0.465	1.257

TABLE II
PROGRAM RUNTIME IN SPIRAL-LIKE MAZES (IN SECONDS)

#	Maze Size						
	2×2	3×3	4×4	5×5	6×6	7×7	8×8
1	0.004	0.008	0.015	0.042	0.069	0.179	0.384
2	0.006	0.009	0.017	0.048	0.092	0.254	0.394
3	0.003	0.006	0.011	0.031	0.077	0.194	0.418
4	0.004	0.006	0.010	0.033	0.087	0.201	0.346
5	0.005	0.011	0.027	0.069	0.112	0.267	0.465
6	0.006	0.010	0.020	0.048	0.083	0.233	0.389
7	0.005	0.009	0.018	0.041	0.092	0.159	0.429
8	0.003	0.009	0.015	0.041	0.103	0.222	0.534
9	0.009	0.020	0.031	0.044	0.087	0.219	0.471
10	0.006	0.009	0.018	0.035	0.073	0.172	0.386

for an average of 1.0561 seconds, while in spiral-like maze, it took only an average of 0.4216 seconds which is half of the time needed for solving no-blocker maze.

In live environment, the proposed method is also submitted ten times in order to check on its performance consistency. Figure 13 shows a bar chart of all ten submissions' execution time measured by Sphere Online Judge, with an average execution time of only 2.456 seconds out of 16 seconds, the maximum time limit given. If we further observe this bar chart pattern, only one trial run (6^{th}) took ± 40 milliseconds longer than the remaining 9 trials. Therefore, this 6^{th} trial

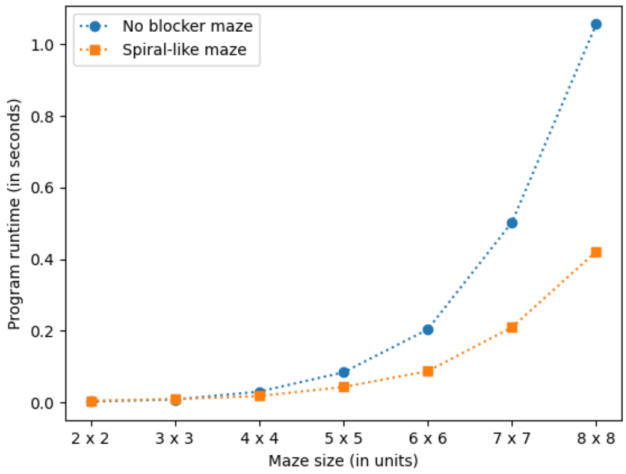

Fig. 12. Average program runtime in different type of mazes. Program runtime in both mazes are exponentially increasing, with larger multiplier in no blocker mazes.

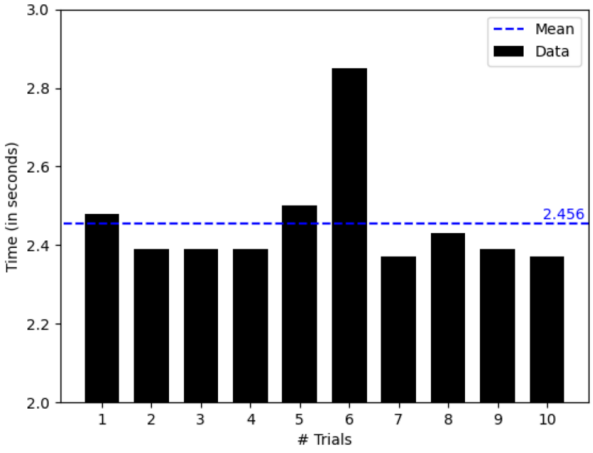

Fig. 13. Timewise Performance Measurement by Sphere Online Judge

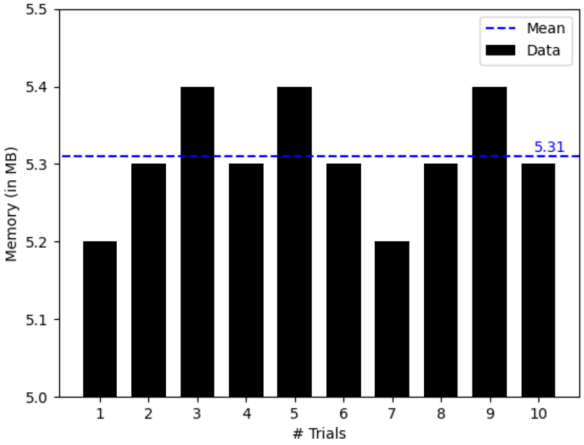

Fig. 14. Spacewise Performance Measurement by Sphere Online Judge

can be considered as an outlier caused by the server load inconsistency during certain busy time [20] [21].

2) Memory Usage: As mentioned in the beginning of this section, the proposed approach memory usage examination is done by Sphere Online Judge that shows how many resources were used when executing submitted program. In average, our proposed approach only needs 5.31 MB of resources

TABLE III
NUMBER OF HOPS NEEDED USING INTUITIVE VS RANDOM APPROACH

Maze Size	No Blocker		Spiral-like	
	Intuitive	Random	Intuitive	Random
4×4	57	73	35	42.5
5×5	59	75.8	40	43.3
6×6	59	74.9	40	48.8
7×7	59	77.4	42	47.4
8×8	59	75.4	47	50.2

which is only 0.3% of the maximum memory limit given, shown in Figure 14. In conclusion, our hybrid approach is indeed efficient spacewise.

C. Heuristic Method Examination

In this section, the efficiency of our proposed heuristic method in solving different maze configurations and sizes will be analysed. As explained in earlier part of this paper, our heuristic method uses a loop to optimize the state in each iteration. The more iterations it needs to reach the optimal state, the less efficient it is. Thus, the number of iterations, also referred as hops, needed to reach the optimal state will be used to represent the efficiency of the heuristic method. This examination was carried out in local environment using the same $N \times N$ maze configurations that were used in the program runtime performance examination in Section III-B with 4×4 as the starting size. This is because smaller mazes most probably need less movement or even just one or two directions which is not an ideal representation to evaluate our heuristic method. In addition, this evaluation was also tested using two starting point selection approaches: intuitively or randomly, which was briefly mentioned in Section II-B. These starting point selection approaches are important to be evaluated to show how significantly faster the intuitive approach is compared to random approach in obtaining the most optimal movement probabilities.

Table III shows how many hops were needed to find the optimal movement probabilities using both intuitive and random approaches in no blocker mazes and spiral-like mazes, ranged from size 4×4 until 8×8. The result is quite contrary to the program runtime based on the maze dimension. Even though both intuitive and random approach shows an increase trend in hops against maze size, it can be seen that intuitive approach has a better performance compared to random approach, especially for no blocker maze type. This is because, in no blocker maze there are more variations of escape paths that can cause higher error rate in choosing optimal movement probabilities. In contrary, spiral-like maze only has exactly one variation of escape path, and thus the error rate is less. Our intuitive approach uses an almost "fair" probability for each direction (e.g. 0.2, 0.3, 0.2, 0.3), depending on the source and destination cell locations in the maze. Hence, it is more efficient to be used in a maze where the optimal movement probabilities' error rate is higher. Another negative side of using random starting point observed from the bar chart from Figure 15 and 16 is, regardless of its uniform average result, it actually has a quite significantly wide variance in every trial. By choosing randomly, the selected starting point can indeed be very near to the optimal state, but it can also be very far, it is a hit-or-miss situation. On the other hand, by choosing intuitively, the selected starting point may not be as near as the random approach's "100% hit" situation, but it can provide a more consistently accurate starting point. From both bar charts in Figure 15 and Figure 16, it can be concluded that choosing the starting point for our heuristic method intuitively is preferable.

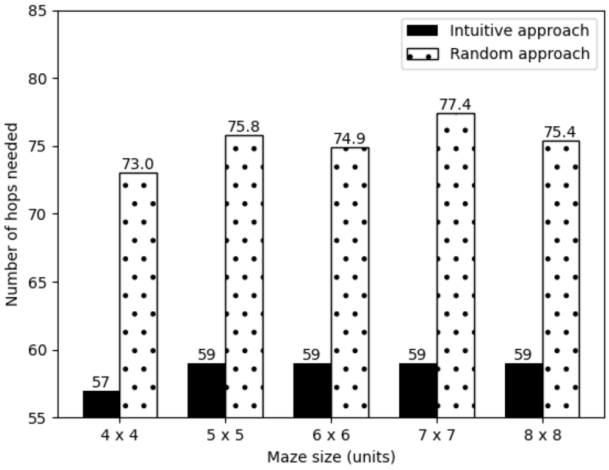

Fig. 15. Comparison between Intuitive and Random Approach in No Blocker Mazes. It is based on the number of hops needed to reach the optimal state. Intuitive approach data is uniform for each trial, thus was taken once. While random approach data is averaged from a total of 10 trials. Overall, intuitive approach performs more effectively compared to random approach by a large margin.

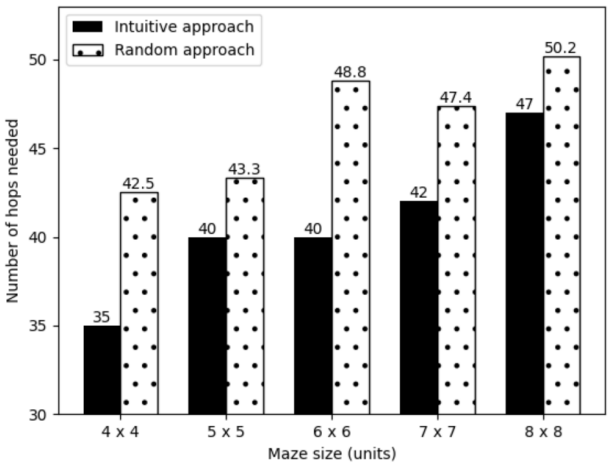

Fig. 16. Comparison between Intuitive and Random Approach in Spiral-like Mazes. Same as the previous figure (Fig. 15), intuitive approach data was retrieved once, while random approach data was retrieved from an average aggregation of 10 trials. In spiral-like mazes, intuitive approach also performs better compared to random approach.

For a better visual representation, Figure 17 shows a 3D-graph which illustrates how hill climbing states are updated in each iteration which we referred as hops, when solving a 8×8 no blocker maze with intuitively determined starting point. In this example, a total of 59 hops were needed, it can be seen that the hops are eventually leading to a certain coordinate which represents the optimal movement probabilities. The hop distances are becoming less as they approach the optimal value to satisfy the precision required. We then tried to use the same maze configuration but with

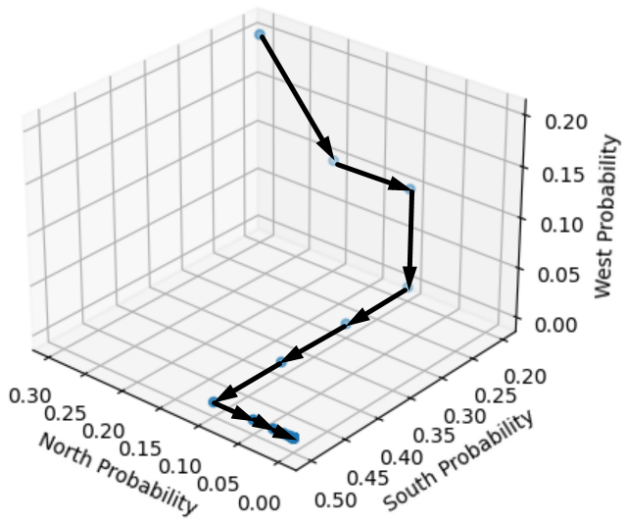

Fig. 17. Hill Climbing Hops Movement in a 8×8 No Blocker Maze with Intuitive Starting Point. A total of 59 hops were needed, almost 20 hops less compared to using randomly selected approach.

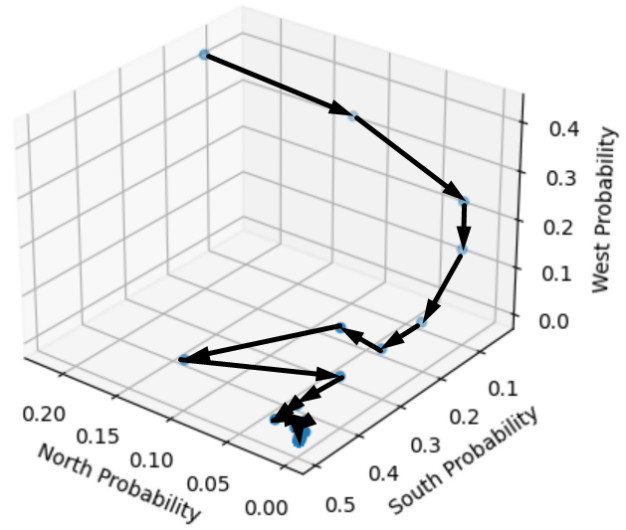

Fig. 18. Hill Climbing Hops Movement in a 8×8 No Blocker Maze with Random Starting Point. A total of 78 hops were needed in this illustration.

a randomly selected starting point, this eventually leads to the same optimal value after a total of 78 hops, as seen in Figure 18. Both of these 3D-graph illustrations further emphasize that by using hill climbing approach, no matter where the starting point is, will always eventually reach the same optimal state. What makes each starting point different is the number of hops needed to reach the optimal state, which is why in our proposed approach, we did not choose the starting point randomly, but rather used an intuitive approach to determine it.

$$e = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \tag{10}$$

To get a more quantitative evaluation on this heuristic performance evaluation, error value between the suggested coordinate in every iteration (x_1, y_1, z_1) and its expected optimal coordinate (x_0, y_0, z_0) will be calculated and illustrated in a line graph. The error value is calculated using Euclidean distance which is shown in equation (10). Euclidean distance is usually used to calculate the distance between several data

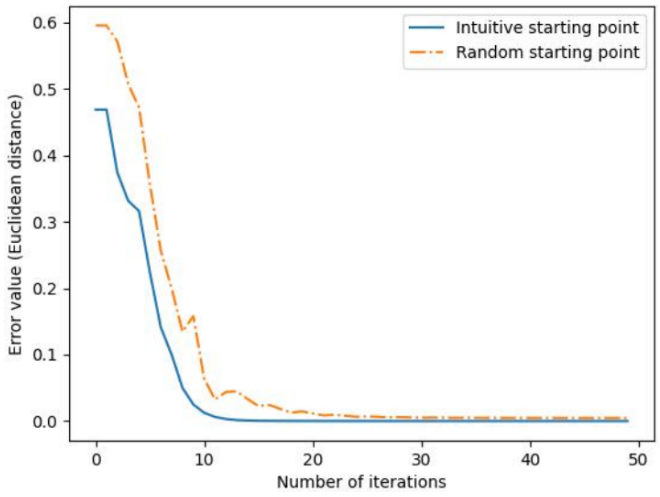

Fig. 19. Error value between given movement probabilities in each iteration with its optimal movement probabilities calculated using euclidean distance as the number of iterations increases.

items [22], which in our case can be applied as a metric to calculate error values. This error value should gradually decrease as the number of iterations increases and eventually reach convergence to prove that the heuristic method is working properly. Line graph shown in Figure 19 validates this hypothesis by showing that using hill climbing with either randomly or intuitively selected starting point, will result an exponentially decreasing trend of error value and finally reach convergence. For intuitively selected starting point, approximately after only 10 iterations, convergence are already reached. While for randomly selected starting point, approximately 25 iterations are needed to reach convergence, which also supports that intuitively selected starting point is a better approach as it reaches convergence faster compared to randomly selected starting point.

IV. Conclusion

In this paper, we apply a one-of-a-kind approach to solve a probability theory problem specifically in calculating the minimum expected escape time from a two-dimensional maze by using hybrid method. This hybrid approach is a fusion of two methods: deterministic and heuristic. The deterministic part is used for calculating the expected escape time using linear algebra based on each maze configuration and its movement probabilities. While the heuristic part is for process optimization of determining movement probabilities, such that the expected escape time is minimized.

The experimental results for this problem have shown that the proposed approach using hybrid method can give consistently valid answer while using efficient resources both timewise and spacewise. In future work, it might be possible to modify our approach to be applicable in more complex problems such as multi-level mazes or mazes with multiple entry and exit points.

References

[1] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Mathematical Expectation*. Prentice Hall, 2012, no. 9th ed., pp. 111–142.

[2] G. Micheli, S. Schraven, and V. Weger, "A local to global principle for expected values," *Journal of Number Theory*, vol. 238, pp. 1–16, 2022.

[3] M. Baron, *Probability and statistics for computer scientists*. Chapman and Hall/CRC, 2018.

[4] S. O. Judge. (2019) Not so blind escape. [Online]. Available: https://www.spoj.com/problems/NBLINDESC/

[5] M. Chen, C. Wu, X. Tang, X. Peng, Z. Zeng, and S. Liu, "An efficient deterministic heuristic algorithm for the rectangular packing problem," *Computers and Industrial Engineering*, vol. 137, p. 106097, 2019.

[6] K. He, W. Huang, and Y. Jin, "An efficient deterministic heuristic for two-dimensional rectangular packing," *Computers and Operations Research*, vol. 39, no. 7, pp. 1355–1363, 2012.

[7] D. Michel and A. Zidna, "A new deterministic heuristic knots placement for b-spline approximation," *Mathematics and Computers in Simulation*, vol. 186, pp. 91–102, 2021.

[8] A. Wahba, R. El-khoribi, and S. Taie, "A new hybrid model for energy consumption prediction based on grey wolf optimization," *IAENG International Journal of Computer Science*, vol. 49, no. 2, pp. 469–481, 2022.

[9] T. Cormen, T. Cormen, C. Leiserson, I. Books24x7, M. I. of Technology, M. Press, R. Rivest, C. Stein, and M.-H. P. Company, *Introduction To Algorithms*, ser. Introduction to Algorithms. MIT Press, 2001.

[10] S. Halim and F. Halim, *Breadth First Search (BFS)*. Lulu.com, 2013, no. v. 3, pp. 123–124.

[11] M. Nazarahari, E. Khanmirza, and S. Doostie, "Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm," *Expert Systems with Applications*, vol. 115, pp. 106–120, 2019.

[12] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[13] B. Wang, B. Lv, and Y. Song, "A hybrid genetic algorithm with integer coding for task offloading in edge-cloud cooperative computing," *IAENG International Journal of Computer Science*, vol. 49, no. 2, pp. 503–510, 2022.

[14] K. A.-R. Youssefi and M. Rouhani, "Swarm intelligence based robotic search in unknown maze-like environments," *Expert Systems with Applications*, vol. 178, p. 114907, 2021.

[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, ser. Always learning. Pearson, 2016.

[16] R. Graves and S. Chakraborty, "A linear objective function-based heuristic for robotic exploration of unknown polygonal environments," *Frontiers in Robotics and AI*, vol. 5, 2018.

[17] S. R. Labs. (2016) Sphere online judge: Clusters. [Online]. Available: https://www.spoj.com/clusters/

[18] ——. (2021) How to cope with spoj? [Online]. Available: https://www.spoj.com/tutorials/USERS/

[19] S. Yendri, R. Soelaiman, U. L. Yuhana, and S. Yendri, "Dynamic programming approach for solving rectangle partitioning problem," *IAENG International Journal of Computer Science*, vol. 49, no. 2, pp. 410–419, 2022.

[20] S. S. Mopuri. (2018) System design — online judge with data modelling. [Online]. Available: https://medium.com/@saisandeepmopuri/system-design-online-judge-with-data-modelling-40cb2b53bfeb

[21] Q. Zhang, X. Li, X. Xie, and J. Ou, "Design of experimental platform of data structure based on online judge," *Journal of Chemical and Pharmaceutical Research*, vol. 7, no. 3, pp. 241–2421, 2015.

[22] Z. Deng, "Research and application of webpage information recognition method based on knn algorithm," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 3, pp. 725–731, 2022.