Maze Runner: Mathematical Explanations and Algorithms

Benjamin Ocampo

August 12, 2024

1 Maze Generation using Depth-First Search

The maze generation method employed is based on a randomized depth-first search (DFS) algorithm. This method generates a perfect maze, which is a maze with exactly one path between any two points. The algorithm can be described in both a mathematical sense and with a clear, step-by-step procedure.

1.1 Mathematical Description

Let the maze be represented by a grid $G = (V, E)$, where V is the set of vertices (or cells) and E is the set of edges connecting adjacent cells. The goal is to create a spanning tree of G, where every cell is reachable from any other cell, and there are no cycles (i.e., every pair of cells is connected by exactly one path).

The maze is generated by performing a randomized DFS on the grid. The process can be described as follows:

1. Start at an initial cell $v_0 \in V$.
2. Mark the current cell v_i as visited.
3. Randomly select an unvisited neighboring cell v_j.
4. Add the edge (v_i, v_j) to the tree and move to v_j.
5. If all neighbors of v_i are visited, backtrack to the previous cell.
6. Repeat steps 2-5 until all cells in V are visited.

This process results in a tree that spans all the vertices of G, creating a maze with a single path between any two cells.

1.2 Algorithm

The maze generation algorithm can be formally described as follows:

Algorithm 1 Maze Generation using Depth-First Search

Input: Grid $G = (V, E)$
Output: Spanning tree $T \subseteq G$, representing the maze
Choose an initial cell $v_0 \in V$
Mark v_0 as visited
Initialize a stack $S \leftarrow \{v_0\}$
while S is not empty **do**
    $v_i \leftarrow$ top element of S
    Let $N(v_i)$ be the set of unvisited neighbors of v_i
    if $N(v_i)$ is not empty **then**
        Randomly choose $v_j \in N(v_i)$
        Add edge (v_i, v_j) to T
        Mark v_j as visited
        Push v_j onto S
    else
        Pop v_i from S
    end if
end while
return T

1.3 Explanation of the Algorithm Steps

1. **Initialization:** Start with an initial cell v_0 and mark it as visited. Push this cell onto the stack S.

2. **Neighbor Selection:** At each step, consider the current cell v_i and its unvisited neighbors $N(v_i)$.

3. **Random Selection:** Randomly select an unvisited neighboring cell v_j and add the edge (v_i, v_j) to the spanning tree T.

4. **Visit and Stack Update:** Mark v_j as visited and push it onto the stack S.

5. **Backtracking:** If all neighbors of the current cell are visited, backtrack by popping the stack until an unvisited neighbor is found.

6. **Termination:** The algorithm terminates when the stack S is empty, indicating that all cells have been visited and the spanning tree T represents the complete maze.

1.4 Custom Modifications to the Maze Algorithm

The traditional maze generation algorithm creates a structure with a single solution path and many dead ends, providing a challenging but deterministic environment. However, to introduce variability and increase the number of potential solutions, we apply a modification that introduces randomness into the maze structure.

After the initial maze generation process, where the maze is formed by a spanning tree of paths connecting the entrance to the exit, we introduce a randomization step that evaluates each cell designated as a wall. For each wall cell, a random number generator is used to determine whether that wall should be converted into a path. This process can be described mathematically and algorithmically as follows:

1.4.1 Random Path Conversion

Let W denote the set of all cells in the maze grid that are classified as walls. For each cell $w \in W$, generate a random number r from a uniform distribution $U(0,1)$. Define a threshold probability $p_{\text{convert}} \in (0,1)$ that determines the likelihood of converting a wall into a path. The rule for conversion is:

$$\text{If } r \leq p_{\text{convert}}, \text{ then } w \text{ is converted to a path.}$$

This probabilistic approach introduces multiple alternative paths within the maze, thus increasing the number of potential solutions that a solver might find. The parameter p_{convert} controls the density of additional paths: a higher p_{convert} leads to more paths and a less challenging maze, while a lower p_{convert} maintains more of the original wall structure, preserving difficulty.

1.4.2 Algorithmic Steps

1. **Initial Maze Generation:** Generate the maze structure using a depth-first search (DFS) or other maze generation algorithm, resulting in a maze composed of paths and walls.
2. **Random Evaluation:** For each wall cell $w \in W$, generate a random number $r \in U(0,1)$.
3. **Conversion Decision:** Compare the random number r with the threshold p_{convert}:
   - If $r \leq p_{\text{convert}}$, convert w into a path cell.
   - If $r > p_{\text{convert}}$, retain w as a wall.
4. **Finalization:** After evaluating all wall cells, finalize the maze structure with the new paths introduced by this random conversion process.

1.4.3 Impact on Maze Complexity

The random conversion of walls to paths increases the complexity of the maze by introducing multiple solution paths. This modification enhances the maze's exploratory nature, as it presents more potential routes from the entrance to the exit, encouraging more diverse problem-solving strategies. However, careful tuning of p_{convert} is necessary to balance the maze's difficulty, ensuring that it remains solvable while offering a richer set of possible paths.

2 Mathematical Foundation of Simulated Annealing

Simulated Annealing (SA) is an optimization technique inspired by the annealing process in metallurgy. It is particularly effective for solving problems with a large search space and finding an approximate global minimum.

2.1 Objective Function

Let $f : S \to \mathbb{R}$ be the objective function we want to minimize, where S is the set of possible solutions. The goal is to find $s^* \in S$ such that:

$$f(s^*) = \min_{s \in S} f(s)$$

2.2 Temperature and Boltzmann Distribution

The algorithm is controlled by a parameter called the *temperature*, denoted as T, which decreases over time. At each step, the algorithm explores the solution space and decides whether to move to a new solution based on the change in the objective function, $\Delta f = f(s_{\text{new}}) - f(s_{\text{current}})$, and the current temperature T. The probability of accepting a worse solution is given by the Boltzmann distribution:

$$P(\Delta f, T) = \exp\left(\frac{-\Delta f}{T}\right)$$

where $\Delta f > 0$ indicates a move to a worse solution.

2.3 Cooling Schedule

The temperature T is gradually decreased according to a cooling schedule. A common cooling schedule is the exponential decay:

$$T_{k+1} = \alpha T_k$$

where $\alpha \in (0, 1)$ is the cooling rate and k is the iteration index.

3 Simulated Annealing Algorithm

The simulated annealing algorithm can be described as follows:

Algorithm 2 Simulated Annealing

Input: Initial solution s_{current}, initial temperature T_0, cooling rate α, stopping temperature $T_{\min}$
Output: Approximate solution s^*
$s_{\text{best}} \leftarrow s_{\text{current}}$
$T \leftarrow T_0$
while $T > T_{\min}$ **do**
    Generate a neighbor s_{new} of s_{current}
    $\Delta f \leftarrow f(s_{\text{new}}) - f(s_{\text{current}})$
    if $\Delta f \leq 0$ **then**
        $s_{\text{current}} \leftarrow s_{\text{new}}$
        if $f(s_{\text{current}}) < f(s_{\text{best}})$ **then**
            $s_{\text{best}} \leftarrow s_{\text{current}}$
        end if
    else
        Generate a random number $r \in [0, 1]$
        if $r < \exp\left(\frac{-\Delta f}{T}\right)$ **then**
            $s_{\text{current}} \leftarrow s_{\text{new}}$
        end if
    end if
    $T \leftarrow \alpha T$
end while
return s_{best}

4 Explanation of the Algorithm Steps

1. **Initialization:** Start with an initial solution s_{current} and set the initial temperature T_0.
2. **Neighbor Generation:** At each step, generate a new candidate solution s_{new} by making a small random modification to the current solution s_{current}.
3. **Cost Evaluation:** Compute the change in the objective function, $\Delta f = f(s_{\text{new}}) - f(s_{\text{current}})$.
4. **Acceptance Criterion:**
    - If $\Delta f \leq 0$, accept s_{new} as the current solution (it is better than or equal to the current solution).
    - If $\Delta f > 0$, accept s_{new} with a probability $P(\Delta f, T) = \exp\left(\frac{-\Delta f}{T}\right)$.

5. **Update Temperature:** Reduce the temperature T according to the cooling schedule.

6. **Termination:** Repeat the process until the temperature T falls below a predefined minimum value $T_{\min}$. The best solution found during the process is returned as the approximate global minimum.

5 Ant Colony Optimization Method

Ant Colony Optimization (ACO) is a probabilistic technique for solving computational problems that can be reduced to finding good paths through graphs. It is inspired by the behavior of ants in finding paths from their colony to food sources. The method is particularly effective for optimization problems such as the traveling salesman problem (TSP), where the goal is to find the shortest possible route that visits a set of cities and returns to the origin city.

5.1 Mathematical Description

Let the problem be represented by a graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges connecting the vertices. Each edge $(i, j) \in E$ has an associated pheromone level τ_{ij} and a heuristic value η_{ij}, which represents the desirability of choosing that edge based on problem-specific knowledge (e.g., the inverse of the distance between cities in TSP).

The probability $p_{ij}^k(t)$ of an ant k at vertex i moving to vertex j at time t is given by:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}$$

where:

- $\tau_{ij}(t)$ is the pheromone level on edge (i, j) at time t.
- η_{ij} is the heuristic value of edge (i, j).
- α is a parameter that controls the influence of the pheromone.
- β is a parameter that controls the influence of the heuristic information.
- N_i^k is the set of feasible neighbors of vertex i that ant k can move to.

As ants traverse the graph, they deposit pheromone on the edges they traverse. The amount of pheromone deposited by ant k on edge (i, j) is proportional to the quality of the solution found by the ant, typically inversely proportional to the length of the path:

$$\Delta\tau_{ij}^k(t) = \frac{Q}{L_k}$$

where Q is a constant and L_k is the length of the path found by ant k.

The pheromone levels are updated after all ants have completed their paths. The update rule is given by:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

where:

- $\rho \in (0,1)$ is the evaporation rate, which controls the decay of pheromone over time.
- m is the number of ants.

The evaporation process ensures that the algorithm does not converge too quickly to a suboptimal solution by reducing the influence of previously laid pheromone trails.

5.2 Algorithm

The Ant Colony Optimization algorithm can be described step-by-step as follows:

1. **Initialization:** Initialize the pheromone levels $\tau_{ij}(0)$ on all edges $(i,j) \in E$ to a small positive value. Set the parameters α, β, ρ, and Q. Place m ants at random vertices in the graph.
2. **Construct Solutions:** Each ant constructs a solution by traversing the graph. Starting from its initial vertex, each ant k chooses the next vertex j based on the probability $p_{ij}^{k}(t)$, until it has visited all vertices or completed its tour (in the case of TSP).
3. **Evaluate Solutions:** Compute the length L_k of the path found by each ant k. This can be the total distance traveled in TSP or another objective function specific to the problem being solved.
4. **Update Pheromones:** After all ants have completed their tours, update the pheromone levels on the edges according to the pheromone update rule. This includes both the pheromone deposition by ants and the evaporation of existing pheromones.
5. **Check Termination:** If a termination condition is met (e.g., a maximum number of iterations is reached or the solutions converge), stop the algorithm. Otherwise, go back to step 2 and repeat the process.
6. **Return the Best Solution:** The best solution found by the ants is returned as the output of the algorithm.

5.3 Impact on Optimization

Ant Colony Optimization is a versatile and powerful algorithm, particularly well-suited for combinatorial optimization problems. The balance between exploration and exploitation is managed through the pheromone updating process, ensuring that the ants collectively search the solution space effectively while gradually focusing on the most promising areas. The algorithm's performance depends on careful tuning of the parameters α, β, ρ, and Q, as well as the problem-specific heuristic information η_{ij}.

5.4 Enhancing the Best Solution with Simulated Annealing

After the Ant Colony Optimization (ACO) process identifies a set of potential solutions, the best solution found by the ants is further refined using Simulated Annealing (SA). This step aims to enhance the quality of the solution by exploring the solution space more thoroughly, particularly by fine-tuning the paths based on pheromone trails.

5.4.1 Integration of Simulated Annealing with Ant Colony Optimization

Once the ants have completed their tours and the best path has been identified based on the highest pheromone concentration, the Simulated Annealing algorithm is applied to this path. The key idea is to use the pheromone levels as a guiding mechanism while allowing controlled random exploration to avoid local minima.

The process can be described mathematically as follows:

1. **Initial Path Selection:** Let P_{best} be the path identified by the ACO with the highest pheromone level. This path serves as the initial solution for the Simulated Annealing process.

2. **Objective Function:** The objective function $f(P)$ to be minimized is defined as the total cost of the path, which could be the distance, time, or any other problem-specific metric, possibly modified by pheromone levels. The goal is to find a path P^* such that:

$$f(P^*) = \min_P f(P)$$

3. **Annealing Process:** Starting from the initial path P_{best}, small perturbations are made to the path, such as swapping, reversing, or relocating segments of the path. Each new path P_{new} is accepted based on the simulated annealing acceptance probability:

$$P_{\text{accept}} = \exp\left(\frac{-\Delta f}{T}\right)$$

where $\Delta f = f(P_{\text{new}}) - f(P_{\text{current}})$ and T is the current temperature.

4. **Pheromone-Guided Exploration:** The perturbations are guided by the pheromone levels on the edges. Paths that follow stronger pheromone trails are favored, but the annealing process allows occasional deviations to explore potentially better alternatives.

5. **Temperature Schedule:** The temperature T is gradually reduced according to a cooling schedule, allowing the algorithm to focus more on exploitation of the best solutions found as the process progresses.

6. **Convergence:** The simulated annealing process continues until the temperature falls below a predefined threshold $T_{\min}$. The final path P^* is returned as the refined best solution, combining the global search capabilities of ACO with the local refinement of SA.

5.4.2 Advantages of Combined ACO and SA

The combination of Ant Colony Optimization with Simulated Annealing leverages the strengths of both algorithms. ACO provides a strong global search mechanism driven by collective intelligence, while SA offers a powerful local search capability to refine the best solutions. By applying SA to the pheromone-based paths, the algorithm effectively balances exploration and exploitation, leading to a higher likelihood of finding an optimal or near-optimal solution.

This combined approach is particularly useful in complex optimization problems where the search space is large and contains many local minima. The pheromone-guided annealing ensures that the refinement process is not purely random but is instead informed by the collective search history of the ants, leading to more efficient convergence on high-quality solutions.