In The Name of God

Navid Zare

Evolutionary Computing
Assignment 4: Genetic Programming - Evolving a Maze Solver

1. Introduction

Autonomous navigation in structured environments represents a fundamental challenge in artificial intelligence and robotics. Traditional approaches rely on explicit algorithms such as A* or Dijkstra's method, which require complete environmental knowledge and produce deterministic solutions. In contrast, evolutionary approaches offer the possibility of discovering novel navigation strategies through adaptation and selection.

Genetic Programming (GP), extends evolutionary computation to the domain of program synthesis. Rather than evolving fixed-length parameter vectors, GP evolves tree-structured programs capable of expressing complex conditional logic. This representation is particularly well-suited to reactive control tasks, where agent behavior depends on environmental perception.

1.1 Problem Clarifications

Several important clarifications from official assignment notes informed the implementation:

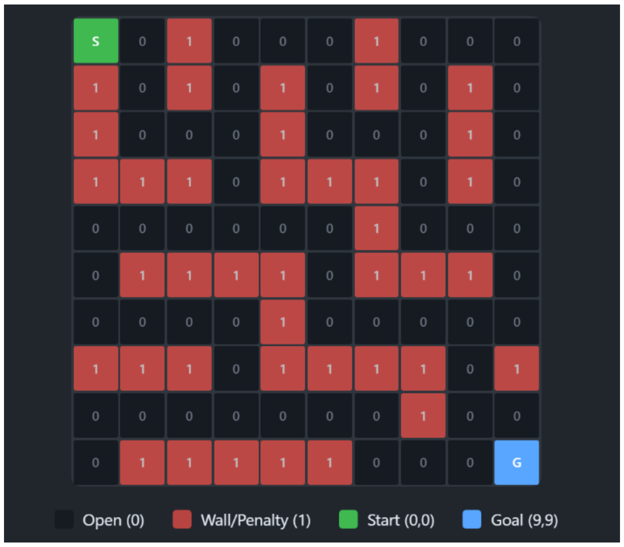
1. Movement actions (UP, DOWN, LEFT, RIGHT) occupy leaf nodes, while conditional sensing operations occupy internal nodes.
2. Penalty cells (in assignment called walls) permit traversal with associated fitness costs; they do not block movement.
3. The fitness function applies universally, including when the goal is reached. Zero fitness requires optimal path length with no penalties.
4. Initial population filtering prevents trivially optimal solutions from appearing in generation zero.

2. Problem Formulation

2.1 Environment Specification

The navigation environment consists of a 10×10 grid where each cell is classified as either open (value 0) or a penalty cell (value 1). The agent begins at position (0,0) in the top-left corner and must navigate to the goal at position (9,9) in the bottom-right corner. The Manhattan distance between start and goal is 18 cells, establishing a theoretical minimum path length.

The maze topology presents a non-trivial navigation challenge. Penalty cells are distributed throughout the grid in a pattern that blocks direct diagonal traversal, requiring the agent to navigate around obstacles. The specific configuration contains 34 penalty cells, creating multiple decision points where the agent must choose between alternative paths.



S	0	1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0	1	0
1	0	0	0	1	0	0	0	1	0
1	1	1	0	1	1	1	0	1	0
0	0	0	0	0	0	1	0	0	0
0	1	1	1	1	0	1	1	1	0
0	0	0	0	1	0	0	0	0	0
1	1	1	0	1	1	1	1	0	1
0	0	0	0	0	0	0	1	0	0
0	1	1	1	1	1	0	0	0	G

Figure 1: The Maze Grid

2.2 Fitness Function

The fitness function is formulated as a minimization objective combining four components:

$$F = s + 2d + 10w + 5l$$

s (steps): Base cost for each movement taken
d (distance): Manhattan distance to goal; weighted ×2 to encourage goal-seeking behavior
w (walls): Penalty cell traversals; weighted ×10 to create strong selective pressure against wall hits

l (loops): Revisited positions; weighted ×5 to discourage backtracking

This weighting scheme ensures wall avoidance takes precedence over path optimization while still rewarding efficiency.

3. Methodology

3.1 Representation

The GP system employs a tree-based representation where programs encode reactive navigation policies. The representation distinguishes between two node types:

Terminal Set: Four movement actions {UP, DOWN, LEFT, RIGHT} occupy leaf positions. When execution reaches a terminal node, the corresponding movement is performed.

Function Set (Internal Nodes):

Eight conditional operators, each with two children (true branch, false branch):

Wall-sensing conditions:

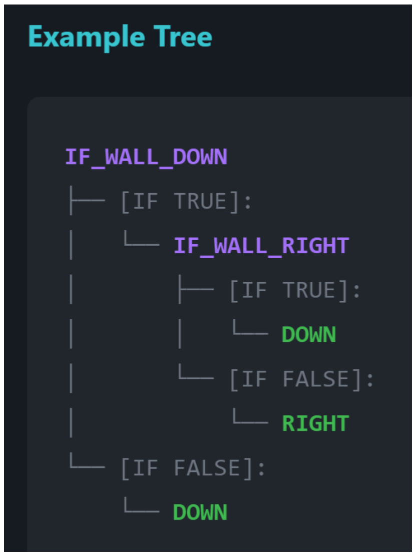
- IF_WALL_UP
- IF_WALL_DOWN
- IF_WALL_LEFT
- IF_WALL_RIGHT

Returns true if the adjacent cell is a penalty cell or out of bounds

Goal-sensing conditions:

- IF_GOAL_UP
- IF_GOAL_DOWN
- IF_GOAL_LEFT
- IF_GOAL_RIGHT

Returns true if the goal lies in that direction relative to the agent



Example Tree

```
IF_WALL_DOWN
├── [IF TRUE]:
│   └── IF_WALL_RIGHT
│       ├── [IF TRUE]:
│       │   └── DOWN
│       └── [IF FALSE]:
│           └── RIGHT
└── [IF FALSE]:
    └── DOWN
```

Figure 2 Example Tree

This tree says:

1. Check if there's a wall below me
2. If YES → check if there's a wall to my right
    - If YES → move DOWN (through the wall, accepting penalty)
    - If NO → move RIGHT
3. If NO wall below → move DOWN

3.2 Population Initialization

The initial population of 200 individuals is generated using the ramped half-and-half method. This approach divides the population equally between full trees (where all branches extend to maximum depth) and grow trees (where branches may terminate early). Tree depths are ramped from 2 to the maximum initial depth of 4, ensuring structural diversity.

A critical implementation detail involves filtering initial individuals. Any tree achieving fitness below 25 is rejected and regenerated. This threshold prevents the population from containing near-optimal solutions at initialization, ensuring that meaningful evolutionary improvement can be observed and measured.

3.3 Selection and Variation

Parent Selection: Fitness-proportional selection (roulette wheel) determines mating opportunities. Since fitness is minimized, values are inverted before calculating selection probabilities. Individuals with lower fitness receive proportionally higher selection probability.

Crossover: Subtree crossover operates with probability 0.9. Two parents exchange randomly selected subtrees, producing two offspring. Depth limits (maximum 8) are enforced; crossovers that would exceed this limit are rejected.

Mutation: With probability 0.1, offspring undergo mutation. Two operators are available with equal probability: subtree mutation replaces a randomly selected subtree with a newly generated tree, while point mutation changes a single node's label to another from the same set (terminal or function).

Survivor Selection: Generational replacement produces each new population from offspring, with elitism preserving the single best individual across generations.

3.4 Agent Simulation

Each individual's fitness is evaluated by simulating agent navigation. The simulation proceeds for a maximum of 60 steps or until the goal is reached. At each step, the GP tree is executed from the root: conditions are evaluated based on current position, and the appropriate branch is followed until a terminal action is reached. The agent then moves in the specified direction.

Movement into penalty cells is permitted but recorded for fitness calculation. Attempted movement outside grid boundaries leaves the agent in place while incrementing the step counter and recording a penalty. This design allows the algorithm to learn boundary awareness through evolution.

4. Experimental Results

4.1 Evolutionary Dynamics

The GP system was executed for 100 generations with the specified parameters. Figure 1 presents the fitness progression, tracking both the best individual and population average across generations.

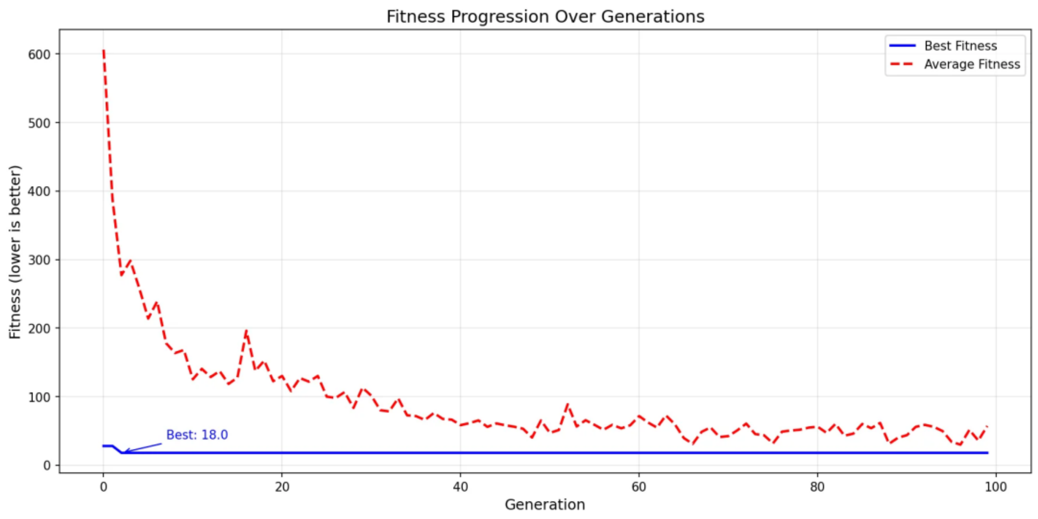


Figure 3: Fitness progression over 100 generations showing best (solid) and average (dashed) fitness values.

The results reveal rapid initial convergence followed by sustained optimization. The best fitness improved from 28.0 in generation 0 to the optimal value of 18.0 by generation 10, representing a 36% improvement within the first 10% of the evolutionary run.

The population average exhibited more dramatic improvement, decreasing from 606.3 initially to approximately 57.2 by generation 99—a reduction exceeding 90%.

The persistent gap between best and average fitness indicates maintained population diversity throughout evolution. Average fitness fluctuations between generations 40-99 (ranging from 42.4 to 71.7) suggest ongoing exploration of the solution space even after optimal solutions were discovered.

Table 1: Selected Generational Statistics

Generation	Best Fitness	Mean Fitness
0	28.0	606.3
10	18.0	125.3
50	18.0	47.2
99	18.0	57.2

4.2 Optimal Solution Analysis

The best evolved solution achieves fitness 18, corresponding to the theoretical minimum given the maze topology. Figure 4 illustrates the path traversed by this solution.

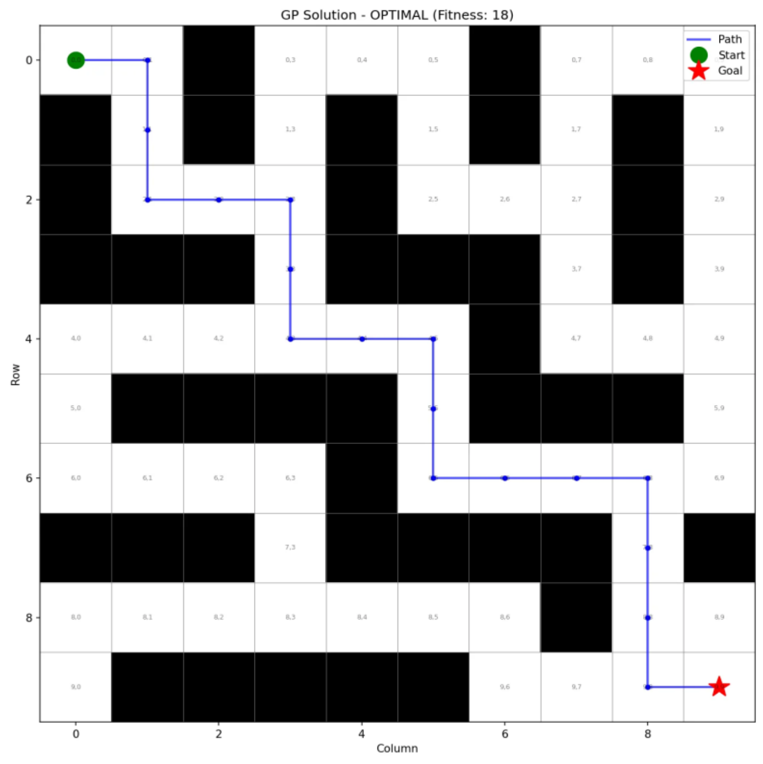


Figure 4: Optimal solution path (fitness = 18). The agent navigates from start (green circle) to goal (red star) without traversing any penalty cells.

The optimal solution exhibits several noteworthy characteristics. The path length of 18 moves matches the Manhattan distance, indicating no wasted movement. Zero penalty cell traversals demonstrate that evolution successfully discovered wall-avoidance behavior through selective pressure alone. The absence of revisited positions shows efficient spatial exploration.

The evolved decision tree reveals an intelligent navigation strategy. The root condition IF_WALL_DOWN determines primary movement direction: when the cell below is clear, the agent moves downward (toward the goal). When blocked, secondary conditions check for rightward movement opportunities or fall back to alternative directions. This hierarchical decision structure balances goal-seeking behavior with obstacle avoidance.

4.3 Comparative Analysis of Suboptimal Solutions

Analysis of suboptimal solutions provides insight into the fitness landscape and the role of different fitness components. Two representative suboptimal solutions were examined, with fitness values of 58 and 98 respectively.

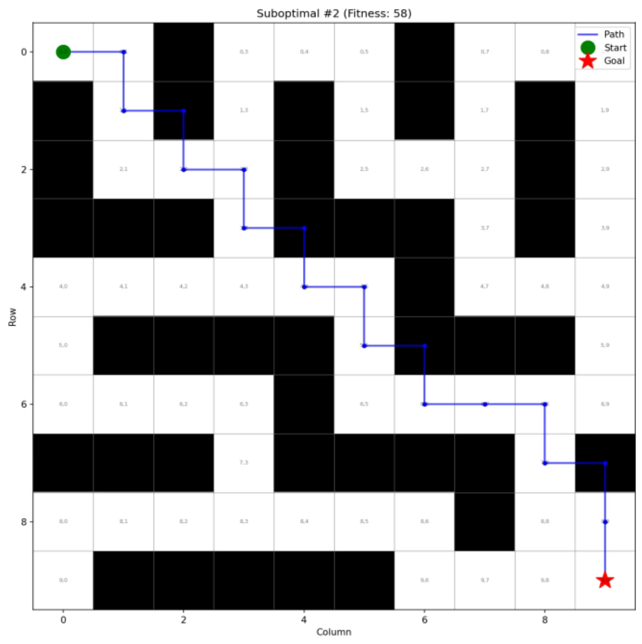


Figure 5: Suboptimal solution with fitness 58 (4 penalty cell traversals).

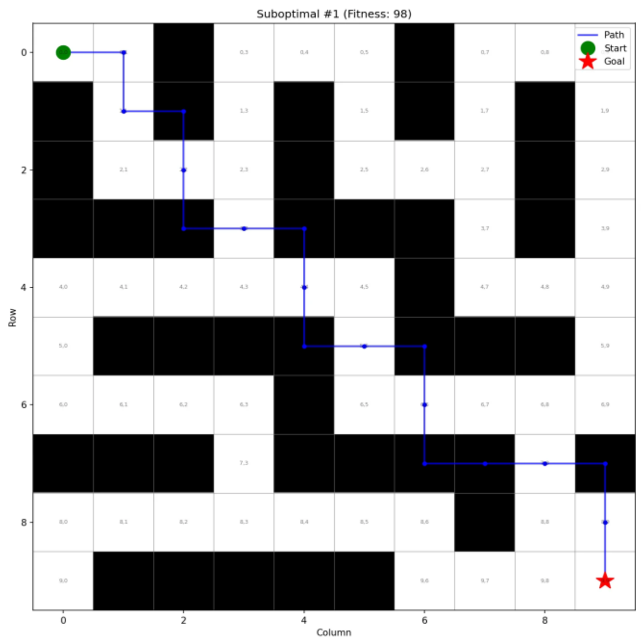


Figure 6: Suboptimal solution with fitness 98 (8 penalty cell traversals).

Table 2: Fitness Component Analysis Across Solutions

Solution	Steps	Distance	Wall Hits	Loops	Fitness
Optimal	18	0	0	0	18
Suboptimal A	18	0	4	0	58
Suboptimal B	18	0	8	0	98

Table 2 reveals that all three solutions achieve identical step counts (18) and reach the goal (distance = 0). The fitness differential arises entirely from penalty cell traversals: Suboptimal A crosses 4 penalty cells contributing 40 points (10 × 4), while Suboptimal B crosses 8 penalty cells contributing 80 points. This confirms that the wall penalty weight dominates fitness differentiation among goal-reaching solutions.

Notably, Suboptimal B employs an extremely simple decision tree with only three nodes: IF_WALL_LEFT at the root with RIGHT and DOWN as children. This minimal structure successfully reaches the goal but lacks the sophistication to avoid penalty cells. The contrast with the optimal solution's more complex tree illustrates the relationship between tree complexity and behavioral refinement.

5. Findings

5.1 Effectiveness of the Evolutionary Approach

The experimental results demonstrate that GP can effectively discover optimal navigation strategies for the maze environment. The rapid convergence to optimal fitness within 10 generations suggests that the representation and operator design are well-suited to the problem. The combination of wall-sensing and goal-sensing primitives provides sufficient environmental information for the evolution of sophisticated navigation behaviors.

A key design decision validated by this work is the treatment of penalty cells as soft constraints rather than hard barriers. By allowing agents to traverse penalty cells with associated fitness costs, the system demonstrates that appropriate behavior can emerge from evolutionary pressure without explicit rule encoding. This approach aligns with the principle that evolutionary systems should discover solutions rather than have them prescribed.

5.2 Role of Fitness Function Design

The multi-component fitness function proves essential to guiding evolution. The high weight assigned to penalty cell traversals (10×) creates strong selective pressure for wall avoidance. Analysis of suboptimal solutions confirms that this component dominates fitness differentiation among solutions that successfully reach the goal. Without this weighting, solutions might converge on direct paths that ignore obstacles.

5.3 Population Diversity and Exploration

The sustained gap between best and average fitness throughout evolution indicates that population diversity is maintained despite elitism. This diversity is important for continued exploration and guards against premature convergence. The fluctuations in average fitness observed in later generations suggest that genetic operators continue to introduce variation even after optimal solutions are found.

The initial population filtering mechanism (rejecting fitness below 25) serves an important experimental purpose: it ensures that optimal solutions are discovered through evolutionary improvement rather than random initialization. The observation that generation 0 achieved best fitness of 28 confirms this mechanism operates as intended.

6. Conclusion

This work demonstrates the successful application of Genetic Programming to the maze navigation problem. The system evolves decision tree programs that encode reactive navigation policies, achieving optimal performance through evolutionary search. Key findings include:

5. Optimal solutions (fitness 18) were discovered by generation 10, demonstrating rapid convergence of the GP system.
6. Penalty cell avoidance emerged from selective pressure without explicit constraints, validating the soft-constraint approach.
7. The weighted fitness function effectively balanced multiple objectives, with penalty cell weight dominating differentiation among goal-reaching solutions.
8. Population diversity was maintained throughout evolution, supporting continued exploration of the solution space.

The implementation adheres to all assignment specifications, including ramped half-and-half initialization, fitness-proportional parent selection, generational survivor selection with elitism, subtree crossover, and mutation operators. The clarifications regarding tree structure (actions in leaves), penalty cell traversability, and universal fitness application are incorporated into the design.

Appendix A: Algorithm Parameters

Table A1: Complete Parameter Configuration

Parameter	Value
Population Size	200
Maximum Generations	100
Initial Maximum Tree Depth	4
Maximum Tree Depth (Runtime)	8
Crossover Probability	0.9
Mutation Probability	0.1
Maximum Steps per Episode	60
Initialization Fitness Threshold	≥ 25
Grid Dimensions	10 × 10
Start Position	(0, 0)
Goal Position	(9, 9)

Appendix B: Primitive Sets

Table B1: Terminal and Function Sets

Type	Primitive	Description
Terminal	UP	Move agent one cell upward (row - 1)
	DOWN	Move agent one cell downward (row + 1)
	LEFT	Move agent one cell leftward (col - 1)
	RIGHT	Move agent one cell rightward (col + 1)
Function	IF_WALL_UP	True if cell above is penalty or boundary
	IF_WALL_DOWN	True if cell below is penalty or boundary
	IF_WALL_LEFT	True if cell to left is penalty or boundary
	IF_WALL_RIGHT	True if cell to right is penalty or boundary
	IF_GOAL_UP	True if goal row < agent row
	IF_GOAL_DOWN	True if goal row > agent row
	IF_GOAL_LEFT	True if goal col < agent col
	IF_GOAL_RIGHT	True if goal col > agent col