Optimal Grid Escape Homework

Salvatore Scotellaro

February 2026

1 Assignment and Report Structure

This report describes the work done to accomplish the assigned homework. Before entering the actual discussion about the resolution process, let's briefly recap the most important aspects of the assignment in the following few points.

- The environment is a $n \times n$ grid that can be traversed horizontally and vertically with an entrance and an exit door.
- Each cell of the grid can either contain wall or be a free cell.
- The goal is to find the shortest path from the entrance to the exit, if it exists with no walls in the middle.
- Whenever no path exists due to some wall presence, the goal becomes first to find the path breaking the least amount of walls, and secondarily reaching the exit traversing the least amount of cells.

Then, in the following sections of this document, we discuss the resolution process. In particular, this report is structured as follows.

- Section 2 describes most relevant parts of the code written for Clingo. This includes how the world has been modeled, most relevant predicates, constraints and the key idea underlying all admissible answer sets.
- Section 3 discusses experiments and results. This goes from description of instances' considered parameters, to metrics' definition and analysis of obtained values for such metrics.

2 ASP Model

To model the problem using Answer Set Programming (ASP), and solve it with Clingo, the following relevant aspects of the world are reflected in atoms and rules. It must be noted that only parts of the program requiring some sort of discussion or explanation are presented, while obvious ones are omitted.

2.1 Grid structure

Starting from the representation of the grid structure, it is sufficient to say that all cells and adjacency relationships are modeled with the following apposite predicates. Other ones, like $cell(X, Y)$, $free\_cell(X, Y)$, ecc. are assumed to be obvious.

- $neighbor(X1, Y1, X2, Y2)$: rule defining what it means for cells $(X1, Y1)$ and $(X2, Y2)$ to be neighbors. This is just a straightforward definition: two cells are neighbors if one can be obtained from the other adding or subtracting 1 from any of its coordinates.
- $passable(X, Y)$: predicate defining whether cell (X, Y) is passable or not, either because it is free or because there is a wall that can be broken.

2.2 Solution's Elements

At this point we move on the discussion of how other solution's elements and/or aspects are introduced in the program. In order, then, we present the modeling of wall breaking decisions, path construction and optimization criteria.

- **Wall Breaking Decisions**: for these, rule $break\_wall(X, Y) : wall(X, Y)$ is used. Clearly, decision depends on reachability of the exit door and on path length optimization result.
- **Path Construction**: to enforce path connectivity $connected(X, Y)$ predicate is used. This just ensures that any cell in the path has at least one neighbor connected to the entrance with some sub-path. Then, it is not possible for a cell to be in the path while not satisfying *connected* predicate's conditions.
- **Hierarchical Optimization**: in terms of optimization, as explained in the introduction, we consider two objectives: primary one minimizes walls broken, while secondary minimizes path length. To model such objectives just *count* operations are used respectively on $walls\_broken$ and in_path predicates.

2.3 Constraints

With problem's model defined through predicates and facts explained above, just the following four constraints must be added. Each of these is, then, briefly explained here below.

- **Entrance Inclusion Constraint**: a constraint used to impose that entrance is in the path, in the form : $-input\_door(X, Y),\ \neg in\_path(X, Y)$.
- **Exit Inclusion Constraint**: a constraint used to impose that exit is in the path, in the form : $-output\_door(X, Y),\ \neg in\_path(X, Y)$.

- **Connectivity Constraint**: constraint imposing for each cell in the path that it must be somehow connected to the rest and of the path, and so to the entrance, in the form : $-in\_path(X, Y),\ \neg\ connected(X, Y)$.

- **Path Neighbors Constraint**: constraint imposing that each non-start and non-end cell has exactly two neighbors. This is expressed in the form reported here below.

$$:-\ in\_path(X, Y),\ \neg input\_door(X, Y),\ \neg output\_door(X, Y),$$
$$\neg\#count\{X1, Y1 : neighbor(X, Y, X1, Y1),\ in\_path(X1, Y1)\} = 2.$$

2.4 Answers Sets Logic

Clarified problem's model and used constraints, the logic underlying answer sets found by the program can finally and easily be summarized in these points.

- Given for each cell its status (free or wall) and adjacency relations, it can be determined the subset of passable cells and neighbors for each cell.

- Entrance and exit are always included in solutions, since they are mandatory elements for each solution.

- Using connectivity constraint and constraint on number of neighbors in a path, all remaining parts of any feasible path are determined.

- Finally, optimization on two-levels priority allows to choose best solution found.

3 Experiments and Results

Passing to experiments, it is important to premise that multiple instances of the problem have been solved varying multiple of their characteristics and evaluated according to several metrics. Then in this section, in order, we go through parameters varied for instances, metrics defined for evaluation and experiments results.

Parameters Starting from parameters varied to define different instances, the following relevant aspects have been taken in consideration.

- **Dimensions**: grid dimensions defining the dimensions of the world, and so of the search space for stable models.

- **Path Availability**: existence of a path connecting the entrance to the exit with no need to break walls.

Metrics Going into metrics, instead, in order to appropriately evaluate performances of the code on defined instances, the ones presented here below are taken in account.

- **Time**: time necessary to find the optimal answer set.
- **Stable Models Number**: number of answers generated while searching for the optimal answer set.
- **Optimality**: success or failure of the program to find the optimal answer set.
- **Path Length**: length of obtained path defined as number of cells in it.
- **Walls Broken**: number of walls that have been broken to reach the exit. Denoted as w_b.

Results Finally, obtained results are summarized in Table 1. For each of the instances cited in this Table, and so for each instance evaluated, there is its graphical representation at the very end of this document. Furthermore, each instance has been included in the GitHub folder (available at this link https://github.com/salvatoreScotellaro/Grid-Escape-Project) and can be used, as explained in the README available in the folder itself, to replicate the experiments.

Instance	Time (s)	Stable Models	Optimal	Path Length	Broken Walls
Test 1	0.005	1	Yes	9	0
Test 2	0.005	1	Yes	9	1
Test 3	0.006	1	Yes	9	1
Test 4	0.008	2	Yes	25	0
Test 5	0.006	1	Yes	11	2
Test 6	0.156	6	Yes	19	1
Test 7	0.608	14	Yes	25	0
Test 8	1.874	8+	Yes	29	0
Test 9	8.969	30+	Yes	33	0
Test 10	11.440	35+	Yes	39	0

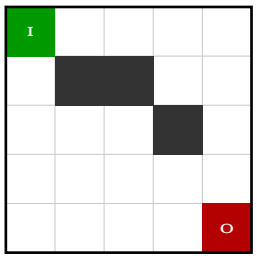
Table 1: Summary of experimental results for each test instance

About obtained results most considerations are somehow obvious. Anyway, all of these considerations are listed in this points.

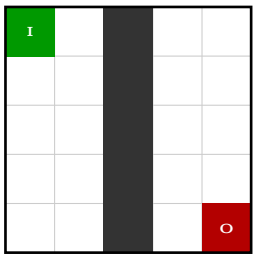
- The ASP program is able to solve efficiently most of the considered instances, regardless of their dimensions, which are still not big though. At this matter is relevant to specify that for the last four test instances the execution has been interrupted by hand as soon as a solution with optimal values was found. In fact, leaving the program free to run until a complete demonstration of the optimality of that models was found, made the

program run for too too long. This is also the reason why for that tests the number of models contains a +.

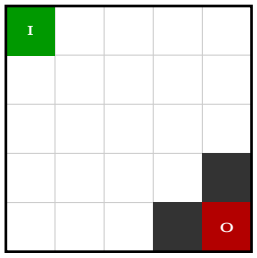
- The number of valid models generated while searching for the optimal solution is very small with respect to number of total theoretically possible ones.
- The program is always able to find the optimal solution on these instances. At this matter, though, it is important to say that also much bigger instances have been considered and in these cases the program even running for a bit does not reach the optimum. Obviously at some point it would, theoretically, reach it, but it is not clear how much time it would require.
- All values of path length and number of walls broken is coherent with the setting of the instance in which it is obtained. This means that, for example, in Test 2 we have one broken wall due to the presence of a complete vertical barrier, in Test 5 we have two walls broken because there is a double barrier or in Test 4 we have zero walls broken with a long path due to the labyrinthine structure.



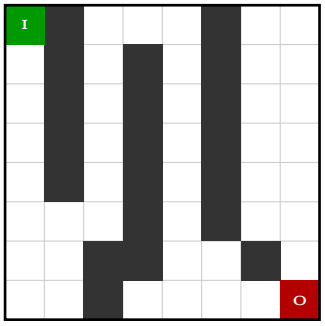
(a) Test 1 - 5×5 with Free Path



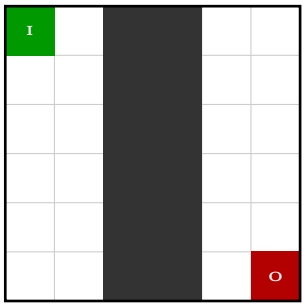
(b) Test 2 - 5×5 with Full Vertical Barrier



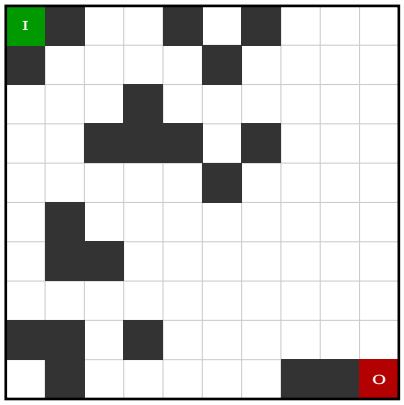
(a) Test 3 - 5×5 with Walled Exit



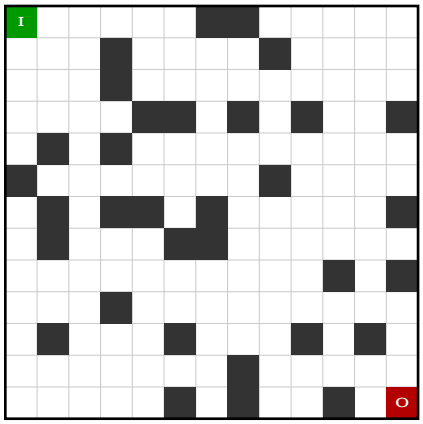
(b) Test 4 - 8×8 with Labyrinthine Structure



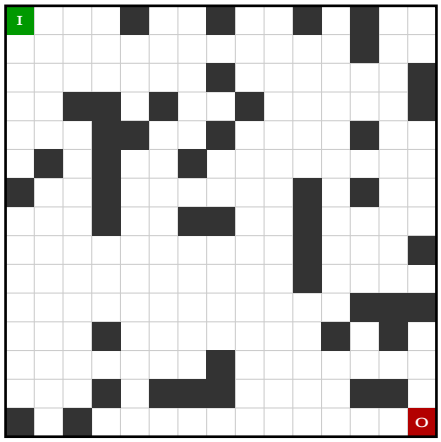
(a) Test 5 - 6×6 with Double Vertical Barrier



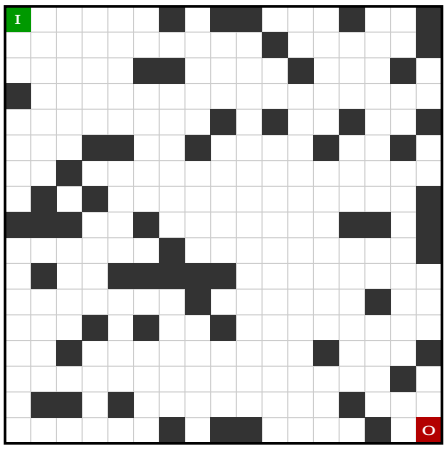
(b) Test 6 - 10×10 with Random Walls



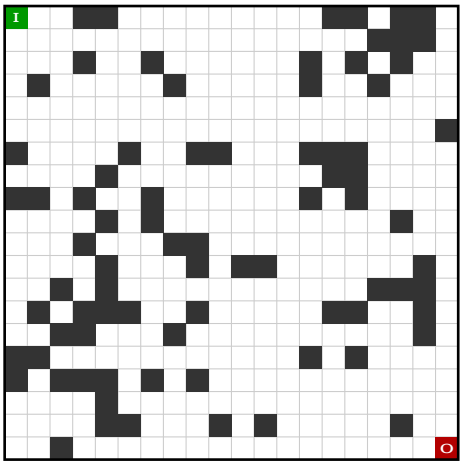
(a) Test 7 - 13×13 with Random Walls



(b) Test 8 - 15×15 with Random Walls



(a) Test 9 - 17×17 with Random Walls



(b) Test 10 - 20×20 with Random Walls