Generating optimal dungeons

John Bush

August 19, 2016

1 Introduction

In games or puzzles, the players' experiences are based upon the perceived difficulty, their results, and particularly their sense of manipulating the outcome. Described by one puzzle designer[1],

> Ancient Greek has this amazing word "aporia" which means "to be at a loss" or "to lack the resources needed" and they talk about it a lot ... partially, I think, because it feels so good to get out of that feeling

From a design perspective, this can lead to contradictory constraints. Giving the player the sense that they are at a loss while allowing them to feel that they are overcoming it by their own resources often comes down to managing the difficulty for the player in a manner invisible to them.
This project is devising a manner of implementing a game scenario. Multiple players are navigating a "dungeon" composed of many rooms linked by unidirectional portals, from an entry point to an exit point. The players' perceptions should be that the dungeon is vast, that the feel urgency and a sense of momentum, and most importantly that their choices are influencing the outcome. At the same time, the level of difficulty must be managed so that is both challenging and solvable. The scenario should be replayable but always in a different configuration.

2 Background

This problem of arrangement is typical of graph theory. By representing the rooms as nodes of a graph and the portals between them as edges different arrangements can be compared and an algorithm for generating a scenario within constraints can be implemented. The chief constraints are,

- Difficulty is roughly proportional to the number of rooms visited by the players. If each edge is weighted as one unit, the difficulty can be expressed as the length of the *path* the players take. Accordingly, possible paths should have upper and lower bounds on length.
- The scenario should offer players' choices in the routes they take. So the graph should include nodes with multiple edges. At the same time, too many choices can be overwhelming to players and slow the pacing of the scenario. Accordingly, the *out-degree*, or ways to exit the node, have been limited to an arbitrarily selected maximum value of 2.
- A directed graph is one in which nodes are joined by *arcs*, which can be traversed in only one direction. This will discourage players from loitering and simplify the problem of bounding path length, as well as being appropriate to the "magic portals" the scenario uses.
- There should be many rooms to offer the players a sense of scale. However, if players were to visit every room, it would be exceptionally difficult. An *Euler path*, a path that traverses every arc exactly once, or a *Hamiltonian path*, a path that visits every node, are both definitely not optimal. Excluding a Hamiltonian path is an exceptionally difficult problem (NP-complete), but there are types of graphs that are known to have no Hamiltonian or Eulerian path.
- A *cycle* within the graph, or a path that visits the same node twice, is not desirable. This would add to the maximum possible path length as well as being unexciting for the players.
- The origin node should *connect* to every node in the tree — a room that can't be reached by the players is useless. Thus the graph should be *spanning*, connects to every node, from the *root* of the origin node. The same would apply to the exit node if the direction of every arc were reversed.

[1] Morse, Lindsay. (2016). *On The Definition and Importance of Actual Puzzles in Escape Rooms*. Retrieved from http://www.puzzlebreak.us/blog/2016/4/20/on-the-definition-importance-of-actual-puzzles-in-escape-rooms

3 Mathematical Techniques

A ***directed acyclic graph*** is a finite directed graph with no directed cycles, and meets all of these constraints. A particularly useful property of directed acyclic graphs is that longest and shortest paths can be computed in linear time. The graph itself was generated in two stages.

1. Build a directed spanning tree rooted in the origin node, whose greatest depth (distance from the origin node) is equal to the desired minimum path length M. That node at greatest depth (distance from origin) is to be the exit node. For a directed spanning tree, the following are equivalent[1]:
   - T is a directed spanning tree of G rooted at r.
   - The in-degree of r in T is 0, the in-degree of every other vertex in T is 1, and T is acyclic.
   - The in-degree of r in T is 0, the in-degree of every other vertex of T is 1, and there are directed paths in T from r to all other vertices.
2. Add routes until every node in the graph has a path to the exit node.

The directed spanning tree in the first stage is generated by randomly adding new nodes to the graph. Since the in-degree (for nodes other than the origin) is 1 for every node (other than the origin), and by arbitrary constraint the maximum out-degree is 2, this spanning tree is an unbalanced binary tree. Thus it follows:

1. The graph will have a minimum size of $M + 1$ nodes, if it produced a simple sequence of nodes (of in-degree 1 and out-degree 1) from origin to exit.
2. The maximum size is 2^M, which would occur if a perfect full binary tree of height $M - 1$ were created in the first stage of the algorithm, before a path of desired length were created.

While these outcomes are undesirable from a practical standpoint, it is not difficult to check the size of the newly created graph and discard it if the size is not within the preferred bounds.

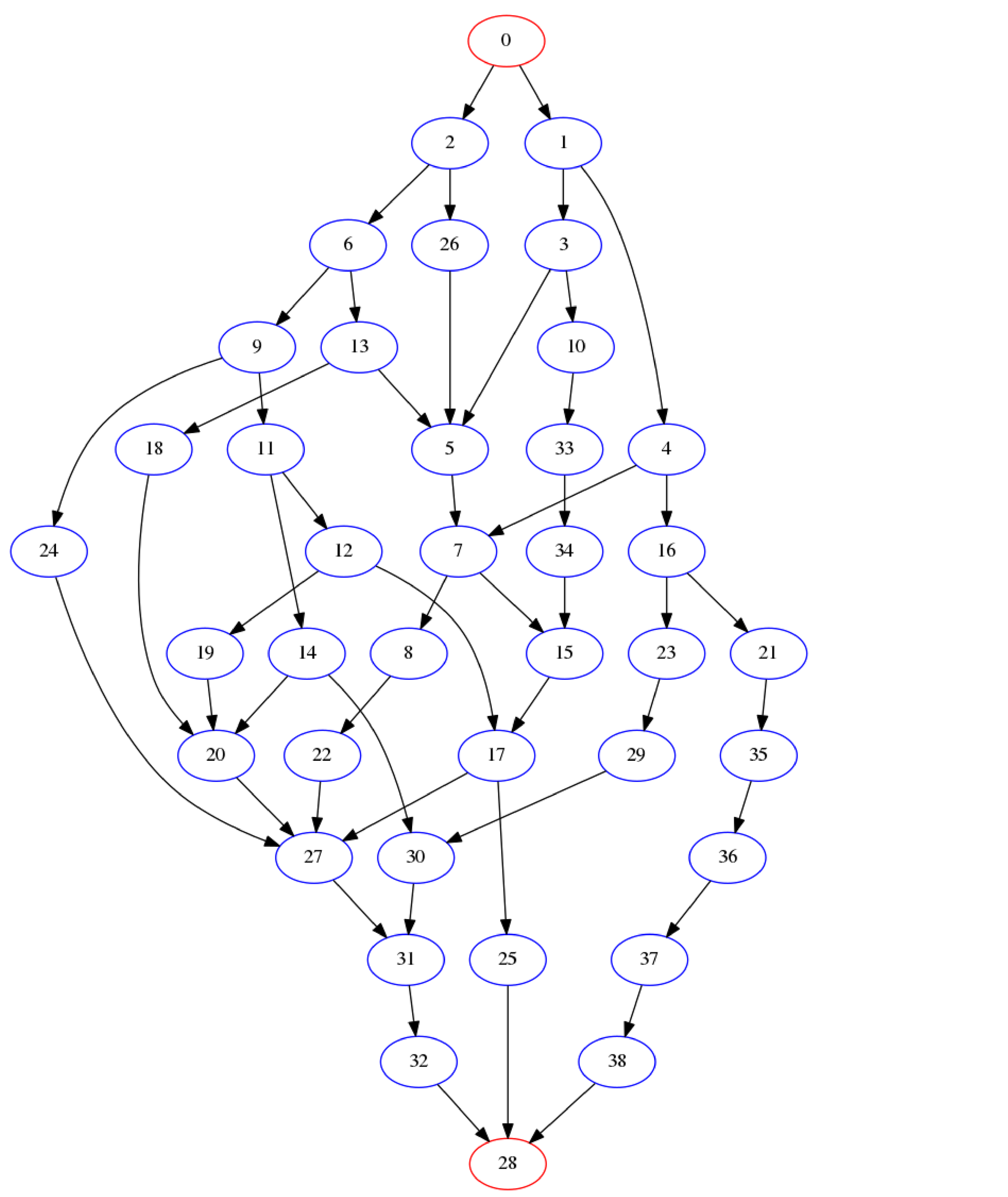

Figure 1: Diagram of graph, generated with maximum path length of 9

[1] Zwick, Uri (2013). *Directed Minimum Spanning Trees.* Retrived from https://www.cs.princeton.edu/courses/archive/spring13/cos528/directed-mst-1.pdf

4 Model

As mentioned above, building the graph was done in two stages. Taking the desired minimum path length M as a parameter: The first stage was accomplished according to the following algorithm. The graph is begun with an origin node, n_O. Two child nodes (n_1, n_2) are added to it. The following is repeated until the longest path from the origin is equal to M.

1. Make a list of all nodes that can accept an adjacent node. The maximum number of nodes is arbitrarily fixed to be 2. This is to keep the size of the graph within manageable limits, and makes the game play more engaging for the players as they are not overwhelmed by choices at any given time.
2. Randomly select one node from this list. Create a new node adjacent to this selected node.
3. Check the graph for a node of the desired distance.

The resulting graph is a directed spanning tree. However, including the origin and exit nodes, only $M + 1$ connect to the exit node.
The algorithm of the second stage repeats until every node connects to the exit node.

1. Similarly to above, make a list of all nodes that
   - Can accept another adjacent node.
   - Do not connect to the exit node.
2. Randomly select one node from this list. This is the node that will be the *start* of a new arc.
3. Make a list of all nodes that
   - Do connect to the exit node.
   - Are of an equal or greater distance from the origin node, compared to the start node.

   Randomly select one node from this list to be the *finish* of a new arc. By always linking nodes that are of similar distance from the origin, paths of very long length are discouraged, as we aren't zig-zagging from the bottom of the tree back to the top.
4. Calculate the shortest paths from the origin to start nodes, and from the finish to exit nodes (in my implementation, using Djikstra's algorithm). If adding an arc from the start node to the finish node will create a path from the origin to the exit nodes that is of length less than M, it will violate the constraints of the graph. So:
   - Create a new node adjacent to the start node.
   - Make this the new start node.
   - Repeat until adding an arc from the new start node to the finish node will not violate the constraints of the graph.
5. Add an arc from the start node to the finish node.
6. Check to see if all nodes connect to the exit node.

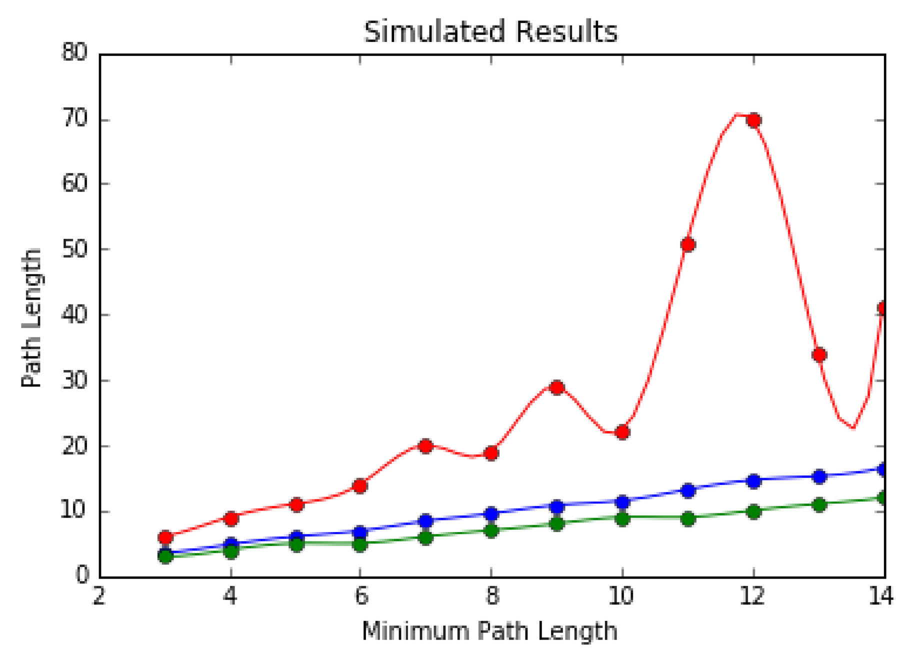

Figure 2: Results of simulations, demonstrating observed path lengths of simulated players for graphs generated by different parameters. Red and green refer to the upper and lower bounds observed, and blue refers to the average over 1000 simulations of 10 different graphs.

Simulating a player's route through a graph is much easier than creating the graph, computationally. In practice, many playthroughs can be simulated for a specific graph in order to identify those with the most desirable characteristics.
The simulation algorithm that was implemented is very similar to a random walk. A list is made to track those arcs which were visited and the simulated player's initial position is set to the origin node. The following is repeated until the simulated player reaches the exit.

1. Randomly select one of the arcs that start in the currently occupied node and does not already exist in the list of visited arcs. If all arcs that start in the currently occupied node have already been visited, select one these visited arcs at random.
2. For the selected arc, the start node is the current position. Replace it with the finish node.
3. Check to see the new position is the exit node.

5 Solution

Figure 1 shows a graph that was generated by this algorithm for a minimum path length of 9. To see how the algorithm scales, simulations were repeated for graphs of minimum path length of 3 to 15. Ten different graphs were generated for each minimum path length, and a player simulation was conducted on each graph 1000 times. Figure 2 shows the results with observed upper and lower bounds (in red and green, respectively) and the average observed path length. Figure 3 shows the same data compared to the average number of nodes in the generated graph for the respective parameters. It can be seen that while the lower bound increases linearly with respect to the minimum path length, the upper bound increases much more rapidly, and the total number of nodes in each graph increases exponentially.
The algorithm does reliably generate graphs that fit the constraints and are certainly practical if simulations are run and the more unsuitable graphs are discarded.

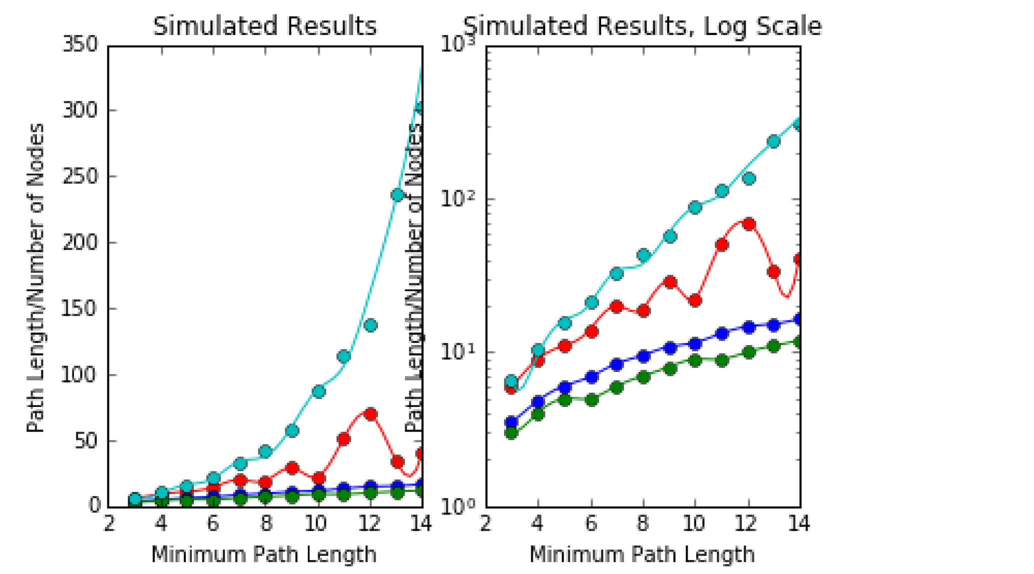

Figure 3: Results of simulations, demonstrating observed path lengths of simulated players for graphs generated by different parameters. Red and green refer to the upper and lower bounds observed, blue refers to the average path length, and cyan refers to the average number of nodes over 1000 simulations of 10 different graphs.