Claire Whittington

Instructor Shaheen

CS 470 Artificial Intelligence

09 February, 2025

Boggle Part 2: Writeup

A smart search algorithm is needed when solving a Boggle board because the number of possible sequences on a board will grow exponentially with each increase in size. My algorithm uses a Depth-First Search (DFS) approach to solve any board without needing to look at every combination. I chose to implement DFS over BFS because I think it's more intuitive for making pathways through the board.

My algorithm works by exploring each letter on the board and recursively traveling to adjacent letters in order to build words. A key feature of the DFS algorithm involves backtracking, which lets it pursue other paths after encountering a dead-end.

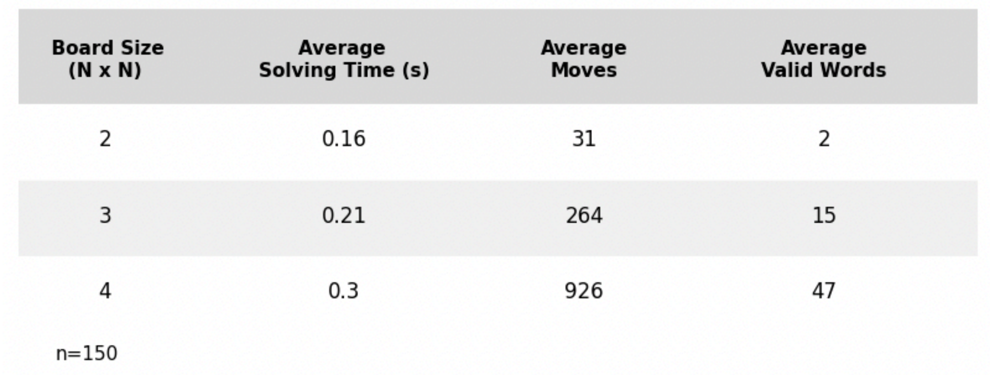
My program is optimized in such a way where all paths are checked against a hashed set of valid prefixes derived from the list of valid Boggle words. Invalid paths are discarded while recognized words are stored in a list. The algorithm organizes all words alphabetically by length and prints a summary to the console upon completion.

In tracking the performance of my program, I've noticed the beginnings of an exponential relationship between size and computational complexity of the board. This is made very clear when analyzing the total moves and words for each board size, N. From N=2 to N=4, the average number of moves grows quickly from 31 to 926 (Figure 1.1). Similarly, the average number of valid words grows from 2 to 47. From these observations, we can deduce that larger boards offer more opportunities to form words while also requiring significantly more computation. This increase in computational power is reflected in Figure 1.2, where time increases somewhat significantly between N=2 and N=4. I predict that as board size increases, we will see an exponential increase in time complexity due to the exponential increase in moves (specifically for N=5, I predict that my program will take between 0.4 to 0.5 seconds).

Since each letter can be followed by any other, I estimate that the total number of possible combinations is around $N^2!$. This simple equation accounts for the exponential nature of the size of the board multiplied by all word lengths of every size up to N^2. However, there will be a large discrepancy between the number of valid words and possible combinations. This is due to the fact that the vast majority of random letter combinations do not form real words. This trend is reflected in the data, where N=4 is predicted to have around 2 trillion possible letter combinations but has an actual average of about 47 valid words.

In a time-limited Boggle competition, I would want my program to maximize efficiency by prioritizing certain letters and combinations over others. For example, the combination of ‘ST’ would be a higher-value play than ‘VU’ because it has a higher probability of validly chaining to its surrounding letters. A strategy like this would fall under heuristic probability because it focuses on likely solutions rather than all solutions. By relying on shortcuts, implementing letter prioritization would be a great way to adjust my program to compete in time-limited competitions.

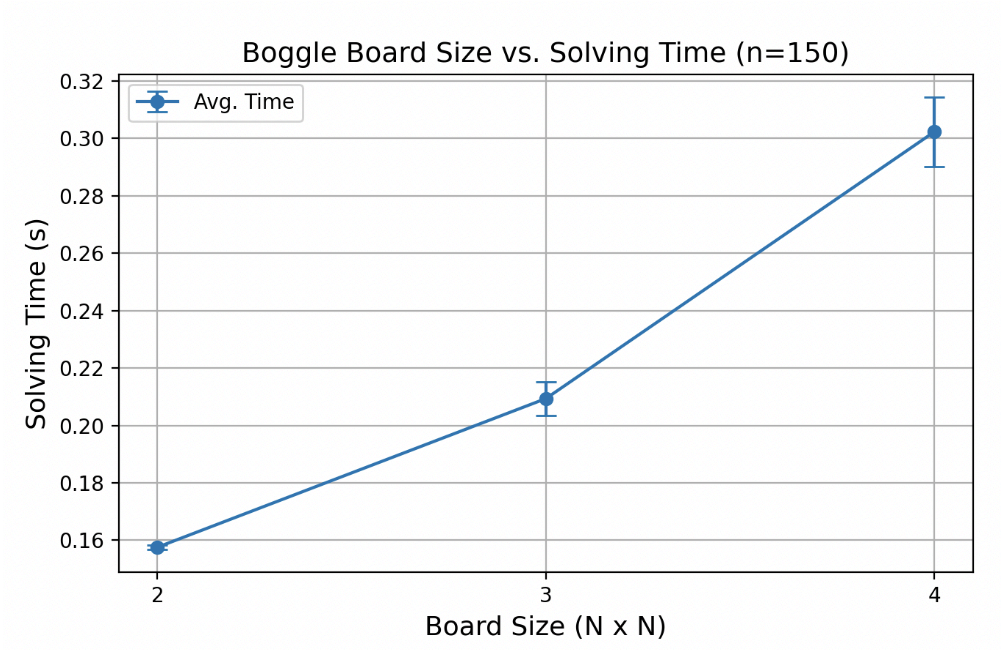
Figure 1.1: Program Summary



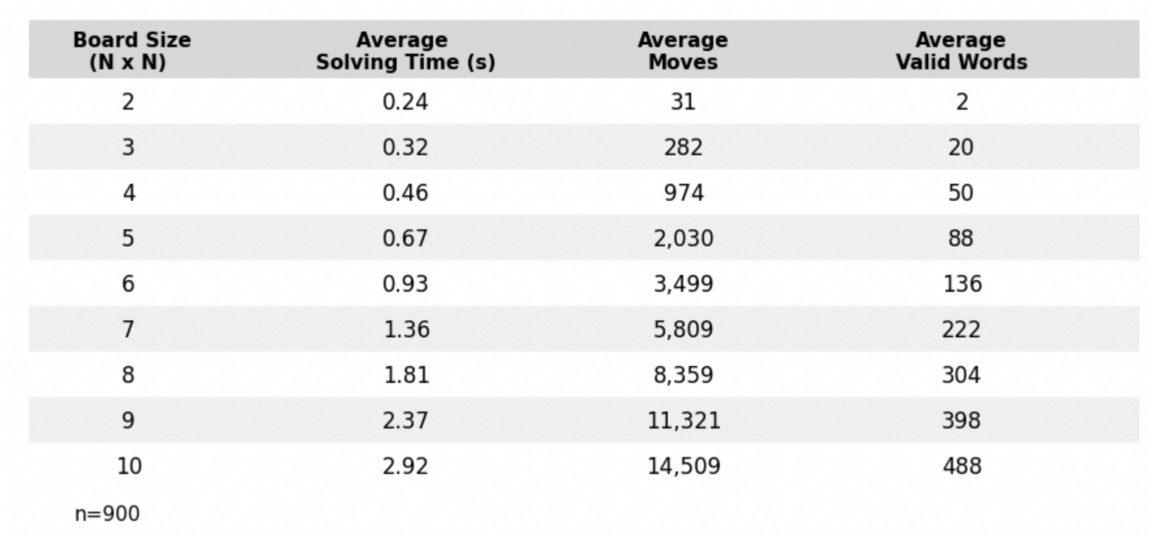
Board Size (N x N)	Average Solving Time (s)	Average Moves	Average Valid Words
2	0.16	31	2
3	0.21	264	15
4	0.3	926	47

n=150

Figure 1.2: Average Solving Time



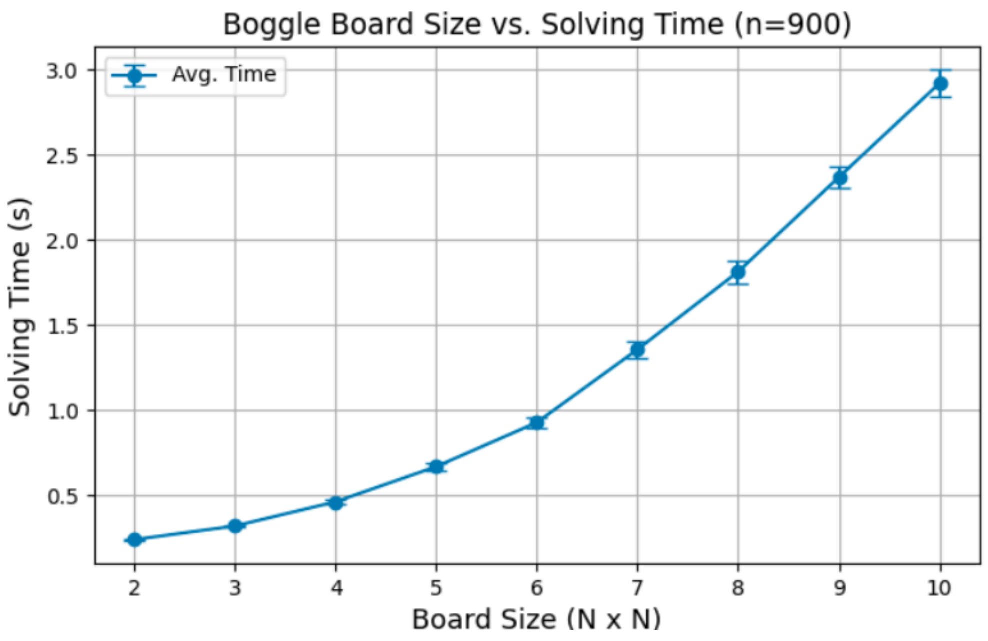
BONUS Figure 1.3: Program Summary up to Size 10



Board Size (N x N)	Average Solving Time (s)	Average Moves	Average Valid Words
2	0.24	31	2
3	0.32	282	20
4	0.46	974	50
5	0.67	2,030	88
6	0.93	3,499	136
7	1.36	5,809	222
8	1.81	8,359	304
9	2.37	11,321	398
10	2.92	14,509	488

n=900

BONUS Figure 1.4: Average Solving Time up to Size 10


Boggle Board Size vs. Solving Time (n=900)
Avg. Time
Solving Time (s)
Board Size (N x N)