The Effectiveness of Informed Search Algorithms in Solving the *Lights Out!* Game

Lindy Bustabad and **Jenny Zhong**
{libustabad,yuzhong}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

We investigate strategies for effective implementation of informed search algorithms to solve the *Lights Out!* puzzle game. We considered three algorithmic approaches to traversing the problem domain: Greedy Best-first search (GBFS), A^* search, and Monte Carlo tree search (MCTS) algorithms. From the results of our experimentation, we found that the A^* search algorithm outperformed GBFS which in turn outperformed MCTS in finding solutions to *Lights Out!* game boards of size 2 by 2 up to 5 by 5. The MCTS algorithm performed poorly, producing high path lengths in solving game boards.

1 Introduction

Lights Out! is an electronic handheld game released by Tiger Electronics in 1995. The game consists of a 5 by 5 grid of buttons, each of which conceals an LED. Figure 1 shows the handheld game device.

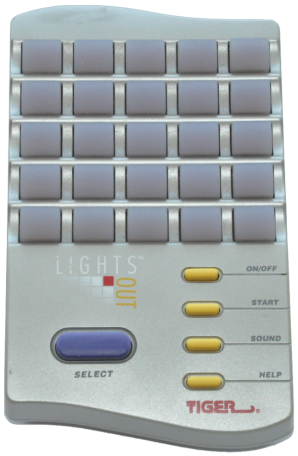

Figure 1: *Lights Out!* handheld game by Tiger Electronics. (Mulholand 2019)

To begin, a predetermined pattern of lights are initially switched on by the handheld device, and the remaining lights are dark. The goal of the game, as one might expect, is to turn all of the lights out. When the player presses a button, that button and every directly adjacent button is toggled—if the light is on, it is switched off and vice versa. Pressing a button that is not on an edge of the grid creates a cross-like pattern of light toggles. The maximum number of lights that can be turned off or on with any button press is 5. Figure 2 shows one progression of game play.

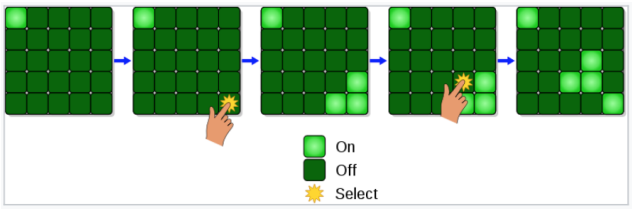

Figure 2: Pressing a button toggles it and its adjacent buttons in a cross-like pattern. (Wikipedia 2020)

The original *Lights Out!* game contained a set of one thousand solvable patterns (Delgado 2007). Tiger Electronics developed multiple versions of the popular game, including *Mini Lights Out* which had a 4 by 4 grid and *Lights Out Deluxe* which had a 6 by 6 grid.

The *Lights Out!* game grid can be represented as a binary matrix, with a 1 in a given location corresponding to that light being on and a 0 in a given location corresponding to that light being off. A solution to the puzzle can be found using linear algebra. Though at first glance a simple game, *Lights Out!* is compelling in part because of its surprisingly complex game theory. Researchers Anderson and Feil proved that only certain configurations are "winnable" by examining the column space of different input matrices (Anderson and Feil 1998).

By devising strategies for which lights to turn off, the game's goal state can be reached with fewer button presses. One common method for guaranteeing a "win" is known as *chasing the lights*. This strategy involves pressing all buttons that have a light on in the row above, starting with the second row of the grid. This approach guarantees all lights in the above row are turned off. The player continues down the grid with each subsequent row until there only remain lights in the bottom row. At this point, the player can utilize a lookup table for the pattern of the lights in the bottom row

to determine what button presses are needed in the top row. Finally, the player continues by "chasing the lights" down the grid again to win the game. Figure 3 provides this lookup table for button presses. While this approach achieves victory, it is still not optimal in terms of the number of moves required for success.

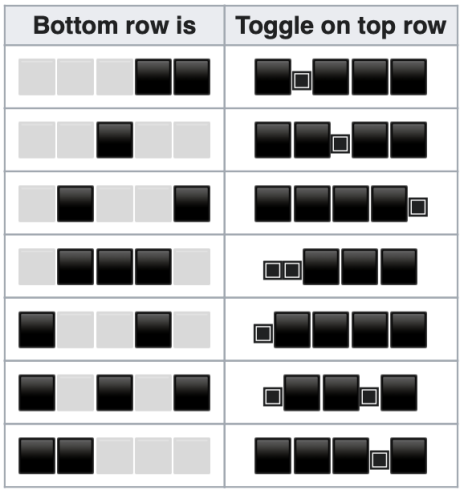

Figure 3: Corresponding lights to toggle in the top row based on light pattern in the bottom row.
(Wikipedia 2020)

We implemented three algorithms to solve the *Lights Out!* game and evaluated their respective abilities to solve the puzzle in as few moves as possible. These three approaches include a Greedy Best-first search algorithm, an A^* search algorithm utilizing a heuristic function, and a Monte Carlo tree search algorithm. We tested these approaches with grids of various sizes in order to determine the most effective search algorithm in playing *Lights Out!*.

This paper will document necessary background on the problem domain of *Lights Out!* and the structures of different search algorithms in Section 2. We will describe our implementations and experimental setup in Section 3. Finally, we will document our experimental results and draw conclusions on the performance of the three algorithmic search approaches in Sections 4 and 5.

2 Background

Initialization of Game Board States in *Lights Out!*

The problem search space for *Lights Out!* reaches large numbers, as each game board of size n by n produces a total of 2^{n^2} possible board states since each of the n^2 buttons on the board can be either turned on or off. Not all of these possible starting board states are considered "winnable" in *Lights Out!*. This explains why Tiger Electronics programmed one thousand guaranteed-solvable light patterns into each handheld device. According to Anderson and Feil, only 1 in 4 boards of size 5 by 5 are able to be solved.

Search Algorithms

A *search algorithm* is an algorithm that solves a search problem by retrieving a goal state within the search space of a problem domain. Search algorithms can be *uninformed*, which navigate a problem domain without any additional knowledge of the search space, or *informed*, which use problem-specific knowledge to guide algorithmic search of a problem domain. Informed search algorithms require access to a *heuristic function* which estimates the goodness of successor nodes to find the most promising path to a goal state.

Greedy Best-First Search Algorithm

The Greedy Best-first search (GBFS) algorithm is an informed search algorithm that explores a graph's search space by expanding the node whose estimated distance to the goal state is the smallest—hence, the name "greedy." The evaluation function for the algorithm is:

$$f(n) = h(n)$$

where n is the next node on the path, and $h(n)$ is the heuristic function that estimates the cost of the cheapest path from node n to the goal state. The GBFS algorithm does not use past knowledge and subsequently is not complete. Thus, there is always a risk of traversing a path that does not reach a goal state. In general, the GBFS algorithm is not optimal, meaning that the path found is not always the optimal path.

A^* Search Algorithm

The A^* search algorithm is another informed search algorithm that explores a graph's search space by finding a path to the goal state with the smallest cost or shortest distance. The algorithm minimizes the evaluation function

$$f(n) = g(n) + h(n)$$

where n is the next node, $h(n)$ is the heuristic function, and $g(n)$ is the cost of the path from the initial state to the node n. In comparison to GBFS, A^* uses past knowledge of the search space, $g(n)$, to evaluate which nodes to expand. Because A^* uses an admissible heuristic function, the search algorithm is optimal and complete. Thus, it will always find the optimal path to the goal state.

Monte Carlo Tree Search Algorithm

The Monte Carlo tree search (MCTS) algorithm is another informed search algorithm. The MCTS algorithm, developed in the early 2000s, was initially used to play board games like chess and Go. MCTS utilizes random sampling of the search space. The game is simulated to its end using random moves, and the moves that yield the best results are given greater weights, making them more likely to be selected in future simulations.

In MCTS, the various possible game states are represented in a game tree, and the search is conducted in four distinct steps. The first step of MCTS is known as *selection*, whereby the tree is traversed from the root node (representing the current game state) to a node that has yet to be fully

expanded, i.e. the algorithm has yet to expand at least one of its children.

Path selection should both *explore* new paths to gain information and *exploit* known good paths using existing information. To do so, the algorithm should select child nodes using a *selection function* that prioritizes both exploration and exploitation. One selection function, named Upper Confidence Bounds applied to Trees (UCT) is:

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

where w_i is the node's number of simulations resulting in a win, s_i is the node's total number of simulations, s is the parent node's total number of simulations, and c is the exploration parameter. The term $\frac{w_i}{s_i}$ is the *exploitation term*, which represents the average win rate for a node. The term $\sqrt{\frac{\ln s_p}{s_i}}$ is the *exploration term*, which increases the less a node is chosen for simulation. The exploration parameter c controls the amount the equation favors exploration over exploitation (Liu 2017). Current literature suggests $\frac{\sqrt{2}}{2}$ is the standard value for c in the UCT function (Browne et al. 2012). However, this standard may not be the best choice for all game simulations. To maximize the UCT function, this number is often chosen empirically.

The step following selection is *expansion*, in which a child node is created from the node reached in selection. This child node must represent a game state that can be reached following a legal move made in the state represented by the leaf node. The third step of MCTS is *simulation*, which involves the game being played out to its conclusion from the child node created in expansion. A reward is given to every child node by calculating how close the output of the random decision is from the output needed to win the game. We will refer to this step as a *rollout*.

In the final step, known as *backpropagation*, information regarding the result of the simulation is passed up the tree from the child node back to the root node. As we perform iterations of MCTS, our algorithm learns more about the search space and which moves are preferable. Using weighted probabilities, one of the most promising moves will be selected. Figure 4 demonstrates how a search tree is grown through repeated application of the four steps of MCTS.

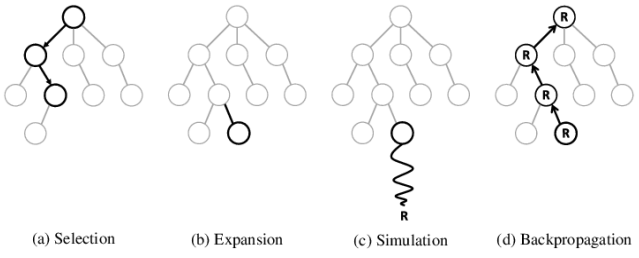

Figure 4: Steps of the Monte Carlo tree search algorithm. (James, Konidaris, and Rosman 2017)

3 Experiments

In our research, we implemented the GBFS, A^*, and MCTS algorithms to solve *Lights Out!* game boards of various sizes.

Initialization of Game Board States

To ensure that all initial game board states were solvable, we began with a solved board (no lights on) and randomly selected a number of moves between 1 and 10. For each of these moves, we randomly selected and toggled one of the n^2 buttons on the board to produce the initial game board state. This reverse-walk from a solved board state ensures that all boards passed to the search algorithms were indeed solvable.

Greedy Best-First Search Algorithm

To implement GBFS for the *Lights Out!* game problem domain, we designed a unique heuristic function, $h(n)$, that estimates the cost of all possible nodes and selects the cheapest node each time. The heuristic function evaluates the cost of each node by counting the number of remaining lights after individually toggling each button on the board. The "greedy" algorithm will choose to toggle the button that will turn off the greatest number of lights. The algorithm repeats this process until the goal state has been reached.

A^* Search Algorithm

As noted in Section 2, A^* is defined by two functions, $g(n)$ and $h(n)$. To design a heuristic function, $h(n)$, for the A^* algorithm, we relied on a few guiding principles. The heuristic considers that progress has been made any time it enters a state where fewer lights are on than were on in the previous game state. In any given state, the minimum number of steps remaining is the number of lights remaining divided by 5, since 5 is the maximum number of lights that can be turned off with a single button press. Using these principles, the function examines each possible move and awards a score to each, ultimately determining the optimal path to turn off all lights that requires the fewest moves. Our $h(n)$ function is

$$h(n) = n^2 - \frac{k}{5}$$

with k defined as the number of lights remaining.

Monte Carlo Tree Search Algorithm

We used Paul Sinclair's Python package "mcts" as the basis for implementing MCTS for *Lights Out!* (Sinclair 2019). To maximize the UCT function, we ran the MCTS algorithm with 10 different 3 by 3 boards using values of exploration parameter c in the range 0.5 to 1.4. We recorded the average number of nodes expanded using each c value. Then, we determined the best exploration parameter to be the c value that minimized the number of nodes expanded throughout the search, implementing this value in our MCTS algorithm.

In searching the *Lights Out!* game tree, the MCTS algorithm will find the move(s) that lead to the quickest victories and select those moves, allowing the algorithm to learn which paths are most effective for turning off all the lights.

Traditional MCTS is designed for two-player games. Random rollouts result in a win, loss, or draw for a player and get backpropagated as such. However, in puzzle domains such as *Lights Out!*, eventually any solvable board game will result in a win for the player. Thus, we need a new way to evaluate expanded nodes. We began by evaluating nodes by the length of a random rollout, as this was closest to traditional MCTS. However, this proved a poor metric, as random rollouts had no way of converging to a solution in a reasonable time, and even moves that were close to being solved would not necessarily be solved efficiently when we play a random policy. Thus, we instead adopted a heuristic by which we can evaluate newly expanded nodes. In our implementation, terminal states were assigned a large reward of 10,000, and previously visited states were assigned small rewards of $-10,000$. All other states were assigned a reward equal to the difference between the board size and the number of lights remaining.

Experimental Trials

We ran the three search algorithms on 10 different solvable game boards for board sizes 2 by 2 up to 5 by 5. We recorded the average path length to a solution over the 10 trials for each algorithm.

4 Results

The average numbers of nodes expanded by the MCTS algorithm on a 3 by 3 board over 10 trials for various c values is presented in Figure 5.

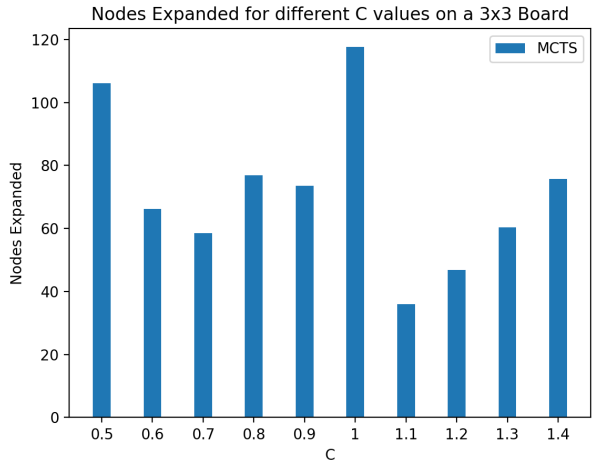

Figure 5: Average number of nodes expanded over 10 trials on a 3 by 3 board for different values of the exploration parameter, c.

As seen in Figure 5, setting c equal to 1.1 results in the shortest path length for MCTS. Thus, we used $c = 1.1$ in all other experiments.

The average numbers of moves taken by the GBFS, A^*, and MCTS algorithms over 10 trials for various board sizes is presented in Figure 6.

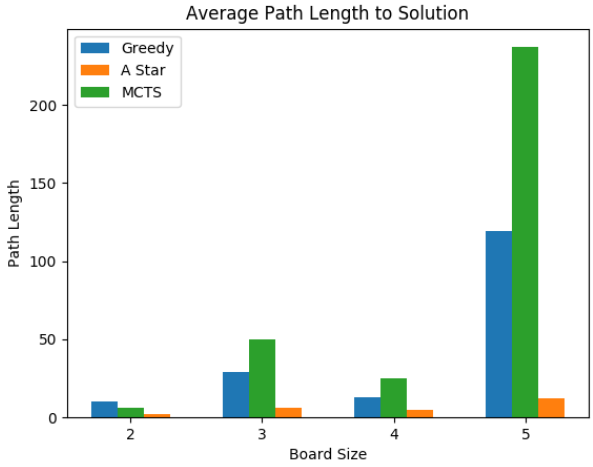

Figure 6: Average path length to solution over 10 trials for various board sizes solved using GBFS, A^*, and MCTS algorithms.

The A^* algorithm found a solution in the least number of moves for each of the board sizes. The A^* algorithm consistently solved a given *Lights Out!* puzzle in under 10 moves on average over 10 trials.

The GBFS algorithm output the next lowest path lengths to solutions. Our implementation was able to solve a given puzzle in under 50 moves for boards up to size 4 by 4. For 5 by 5 boards, GBFS was able to find a solution in approximately 125 moves on average over 10 trials.

The MCTS algorithm output the highest path lengths of the three algorithms. Our implementation was able to solve a given puzzle in around 50 or less moves for boards up to size 4 by 4. For 5 by 5 boards, MCTS was able to find a solution in approximately 250 moves on average over 10 trials.

Predictably, larger boards have larger search spaces and tend to be harder to solve. In our trials, we may have generated some simpler boards for size 4 by 4 which might explain why we see a slight decrease in path lengths. Given sufficient trials and processing power, we would expect to see this decrease become more in line with the trends we see for 2 by 2, 3 by 3, and 5 by 5 boards.

The A^* algorithm successfully produced the optimal solution for each board. In comparison to the other two algorithms, A^* produced a solution about 6 times quicker than GBFS and 12 times quicker than MCTS. GBFS outperformed MCTS, and this is likely due to the design of our MCTS algorithm. Since random rollouts did not perform strongly in single-player game play, our reward function was designed to reward states based on the number of lights remaining and to penalize previously visited states. MCTS would likely perform better with a stronger heuristic. The current algorithm gains information about the search space significantly more slowly than by "greedily" taking the best move each time.

5 Conclusions

Our research implements and compares three informed search algorithms—Greedy Best-first search, A^*, Monte Carlo tree search—to solve the *Lights Out!* puzzle game. The A^* algorithm outperformed both the GBFS and MCTS algorithms, with MCTS performing extremely poorly in finding a solution to a given game board. This is likely due to weaknesses in heuristic reward design and implementation. Avenues for future research include investigating the effectiveness of other search algorithms, including but not limited to Iterative Deepening search and Iterative Deepening Depth-first search. Additionally, current research on the mathematics of determining solvable board states should be utilized to develop an algorithm that detects an unsolvable *Lights Out!* puzzle early in its search iterations. Researchers should also study the problem domain of *Lights Out!* boards wrapped on a torus, in addition to other effective heuristic choices for node evaluation in MCTS.

6 Contributions

L.B. wrote the game board Python class and the GBFS implementation. J.Z. wrote the A^* implementation. Both J.Z. and L.B. wrote the MCTS implementation. Both J.Z. and L.B. ran and recorded data from experiments. J.Z. wrote the introduction and background sections. L.B. wrote the experiments, results, and conclusions sections. Both J.Z. and L.B. wrote the abstract, contributions, and acknowledgements sections. L.B. prepared figures 1, 2, 3, 4, 5, and 6. Both authors proof-read the entire document.

7 Acknowledgements

We would like to thank Dr. Raghu Ramanujan for his assistance with project design and technical support throughout our research process.

References

Anderson, M., and Feil, T. 1998. Turning lights out with linear algebra. *Mathematics Magazine* 71(4):300–303.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Delgado, T. 2007. *COLUMN: 'Beyond Tetris' - Lights Out*. http://www.gamesetwatch.com/2007/01/column_beyond_tetris_lights_ou_1.php.

James, S.; Konidaris, G.; and Rosman, B. 2017. An analysis of monte carlo tree search.

Liu, M. 2017. *General Game-Playing With Monte Carlo Tree Search*. https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238.

Mulholand, J. 2019. *Lights Out*. https://www.sfu.ca/~jtmulhol/math302/puzzles-lo.html.

Sinclair, P. 2019. mcts. https://github.com/pbsinclair42/MCTS.

Wikipedia. 2020. *Lights Out (game)*. https://en.wikipedia.org/wiki/Lights_Out_(game).