Search: Lights Out

Make sure to turn off all the lights.

Arthur Przech - g4przech - Coding / Report
Shuai Zhou - c2zhoush - Evaluation / Results
Gallen Guritno - c4guritn - Report / Editing

CSC384 Project - Lights Out!

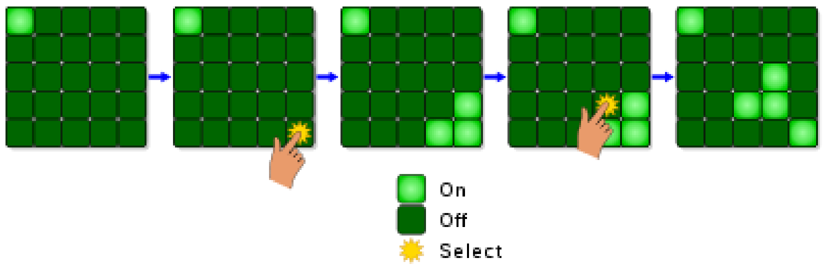
Arthur Przech - g4przech - Coding / Report
Shuai Zhou - c2zhoush - Evaluation / Results
Gallen Guritno - c4guritn - Report / Editing

April 9, 2016

Project Motivation/Background

Project Type: Search

The lights out game is an interesting problem that appears well suited to be solved using the search techniques. The game is played on a $(n \times n)$ board where each cell of the board can be in one of two states, "on" or "off". The goal of the game is to make every cell in the board be "off", hence the name lights out. To accomplish this task the player can press any cell in the board to change flip the value of the cells as well as all the cell that are adjacent to the cell (excluding the diagonally adjacent).



We thought that this would be a good problem for search because the problem had actions that lead to new states that could be the solution state of all cells being off. After working with the problem we found three issues that make this problem not suitable for search, the first is that search space grows extremely quickly as the board increases in size, the second issue with using search to solve the problem is that in some cases there is not going to be a solution for some sizes of boards given certain board configurations. The final issue with the problem is that it has a fast (much faster than any search) solution using linear algebra.

One interesting aspect of the problem is how it can be simplified given the ideas used to solve the problem using linear algebra. The problem is also interesting that it can be formulated in such a way that cycle checking does not need to occur.

Methods and Evaluation

We initially formulated the problem in a very general way. We created our successor states by going adding the state where each possible cell was pressed. This is a problem because as the branching factor of the problem grows quadratically with the size of board. When looking for information on the problem we found the solution to the problem using linear algebra. When reading the resource we found some interesting observations to the problem that helped us improve our model of the game.

The idea is that pressing a cell twice is the same thing as not pressing the cell at all. This means that in a solution to the problem a cell can will be at most pressed once. This helps in the search problem because as the algorithm adds more presses to its current working solution each call to successors will produce one fewer possible successor states because it will not have to consider the state where the cell is pressed twice. This removes the need to prune paths where a cell is pressed twice in both consecutive and non-consecutive presses of the same cell.

We looked at two heuristics for this problem that both used the same idea but they impact the search is different ways. The first heuristic is that the number of presses if each presses removed the most number of lights from the board which is 5. The second heuristics is the the number of presses if each press removed at most one light, or in other words the number of lights that are on.

The first heuristic will give us the path to the solution with the fewest number of presses since it is admissible because any solution can at most toggle off 5 lights in any press. Our heuristic assumes that every remaining press is optimal in turning off 5 lights while in reality those press could remove fewer lights.

The second heuristic will not be admissible as can be seen by the 3x3 lights out solution below:

0	1	0
1	1	1
0	1	0

This problem can be solved with one press to the center cell. However the second heuristic will guess that the problem will be solved in 5 presses.

These heuristics try to avoid searching on presses that lead to more lights being turned on. This can be a problem because sometimes more light needs to be turned on so that a pattern can be created so that more lights can be turned off later. States that turn off more lights but will not lead to a solution are given priority in the search over the ones that form a state with more light.

Forming a heuristic around states that generate patterns is not viable for because sometimes patterns need variable amount of presses for them to form. One could search for these possible patterns but that defeats the purpose of a heuristic because generating the heuristic will be overly time consuming.

We found that the second heuristic converges to a solution faster than the first one. We contribute this to the fact that given a start state that has few lights turned on may have a solution that requires many presses. The first heuristic would focus on searching paths with few presses preventing it from finding the solution quickly. The second heuristic is not admissible so it is more likely to search for a non optimal solution earlier, leading to a less extensive search on shorter paths to a solution.

While working with the problem we found that many initializations of the 4 by 4 puzzle and some 5 by 5 cannot be solved at all. This is make searching on these puzzle not viable because one would need to search the entire state space of the problem. As the size of the board grows in size the branching factor of the problem grow quadratically and the max length of any solution grows quadratically as well.

This leads to a problem of trying to evaluate the performance of searches because it not really possible to tell if a searching going to take a long time or if there is not going to be any solution at all.

This leads us to the third problem of the puzzle: the lights out problem can be solved using linear algebra. This is done by thinking of the problem as a linear combination of matrices where the matrices are the outcome of pressing a single cell of an empty board.

When a button is pressed twice there is no net change. This means that all arithmetic done on the matrices is done under modulo 2. This also leads to the fact that the order in which buttons are pressed does not matter.
To find a solution this we create a special n^2 by n^2 matrix made up of the toggle matrices (the linear combination), and multiply the solution vector where each entry 1 corresponds to the a cell that need to be pressed, this multiplication is equal to the the initial problem being solved. So get back the solution the system needs to be solved under modulo two arithmetic. We got this information from Simon Fraser University lecture[1]. We found an implementation of this method from pmneila [2] on Github in python, we took his implementation and ported it to Python 3. We used their implementation to instantiate versions of the problem that we know were solvable.

Comparatively the solution using linear algebra is extremely quick and is much faster on then solving the problem using search.

We took the idea that a cell does not need to pressed no more than once in any solution and encoded it into are implementation of search. When we are generating successor states for searching on the problem we did not successor that would lead to the same cell being pressed more than once. This gives us the advantage that as the solution path grows the branching factor decreases While there is a decrease in the branching factor this decrease is within a constant factor because the branching factor goes from n^2 to 0 which is equal to $\frac{n(n+1)}{2}$ which is still grows quadratically in terms of the board size.

We also encoded these ideas in our checking of cycles, when we generate a hash state to compare on, we do this on a representation of the presses made so far rather than the current state of the board. This is because the order in which the presses occur does not matter. If the states were hashed on the current representation of the board this would lead to issues occurring because a path could be created to a state of the board using a press that is needed to get from that state of the board to the solution. If the states of the board was hashed then the failed path would prevent a path that would lead to the solution.

Comparison of Search Algorithms Performances

The following tables summarize the overall performances of each of the search algorithms used in the project, measured with respect to runtime and the overall number of nodes expanded. Note that we were only able to reasonably generate data for all 4 search algorithms on a 3*3 board size. Only best-first search was viable to run on board sizes 4*4, 5*5 and above simply because the other algorithms would take too long to run due to the sheer number of the search space.

breath-first initial ONs	runtime	nodes-expanded
1	0.22	3615
3	2.02	35122
6	15.88	260954
9	0.94	19886

depth-first initial ONs	runtime	nodes-expanded
1	29.38	518331
3	8.16	125744
6	0.01	209
9	1.03	18878

best-first initial ONs	runtime	nodes-expanded
1	0.01	1089
3	0.03	247
6	0.05	129
9	0.17	2759

a* (first heuristics) initial ONs	runtime	nodes-expanded
1	0.13	1918
3	0.77	10450
6	13.28	172090
9	0.77	11770

a* with second heuristic initial ONs	runtime	nodes-expanded
1	0.01	182
3	0.05	1009
6	1.21	14930
9	0.10	1808

Based on the data of the tables and on the simple fact that we could not use any of the other algorithms to solve the bigger boards, the best-first search algorithm is the best for our purposes. It is consistently the fastest because the focus of this problem is to simply obtain the final solution of all lights being off, and there is much less focus on trying to get to that final state within as few button presses as possible. This fact is also what makes a* not as ideal, although in general a* is still undeniably better than the uninformed search algorithms. It is interesting to note that best-first will run the same way regardless of which heuristics is used, because both are simply constant multiples of the other. The difference between the two heuristics is only significant in a*, where the actual cost of each button press has influence on the evaluation function.

It is trivial to observe that the runtimes of each search algorithm is directly proportional to the corresponding total number of nodes expanded during the search. More importantly, however, our results above is a fine example of how, even with a less than perfect heuristic, informed searches are far more superior than uninformed searches.

Finally, observe that the initial number of lights that is on at initialization does not necessarily correspond with how long the actual search is going to take, or how many nodes are going to be expanded. This is also evident in the tables. For instance, it can be seen that having 6 lights initially on lead to a* expanding 172090 nodes in total whereas increasing that number to 9 resulted in a significantly less expansion of 11770 nodes.

Conclusion

Initially the lights out problem seemed like a very viable candidate for a search problem. However, we found issues where a solution may not exist and for large board sizes, finding a solution will take a large amount of time. Search method is viable for smaller boards up to 3*3 boards, and 4*4 boards on a lesser extent. In these cases, the branching factors and the maximum path length has yet to explode in its quadratic nature. We have learned on the other hand how this same problem has an elegant solution using the linear algebra constructs. If we were to do this project again in the future, we would look more into specific initializations of the problem, e.g. all lights being turned on only in one of the two halves of the board, no isolated lights, etc. Then we shall look into generating specific heuristics that would work well in those specific kinds of initializations of the board and compare them with our general heuristics.

Citations and References

1. Jamie Mulholland, Lecture 24: Lights Out Puzzle, Simon Fraser University, http://people.math.sfu.ca/ jtmulhol/math302/notes/24-Lights-Out.pdf

2. Github: pmneila, Python Lights Out Solver Using Linear Algebra, lightsout.py, https://github.com/pmneila/Lights-Out/blob/master/lightsout.py