HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

-----ൠ📖ൠ-----

PROJECT REPORT

Modified light bulbs problem

Class: Data science & AI 01 – K64

Group 15

1. Nguyễn Hoàng Anh – 20194417
2. Nguyễn Minh Châu – 20194420
3. Chu Hoàng Dương – 20194429
4. Phạm Như Thuấn – 20194456

Introduction

In the course Introduction to AI, we have studied about several kinds of Intelligent Agents such as simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents, knowledge-based agents. Among them, goal-based agents, consider future actions and the desirability of their outcomes. The problem-solving agents is one kind of goal-based agents, which is the topic we choose for our programming subject.

In this report, we present the modified light bulbs problem and propose some methods for solving this problem. We will also focus on the analysis including analyzing the problem, explaining your choices, analyzing the results of different algorithms.

Contents

1. Presentation of the subject.

- Given a mxn matrix, each element represents the state of a light bulb (ON = 1, OFF = 0), every bulb's state is randomized initially. (m, n is given)

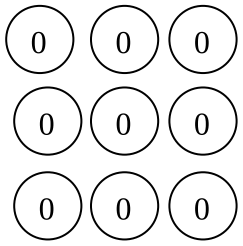


Figure 1. an initial state

- If we change the state of a bulb, the states of all adjacent bulbs of that bulb are also changed.
- If a bulb is "turned on" more than k times (k is given), it will break down and the whole network is down.
- The goal is to turn on every bulb in the network with a minimum number of moves (if possible) and return the process of doing it. If not possible, return Failure

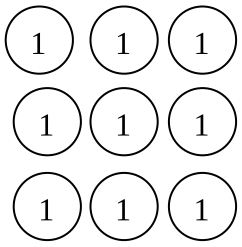


Figure 2. goal state

2. Description of the problem.

2.1 Detailed description of the problem.

-The environment is known, observable, discrete and deterministic

→ the solution to a problem is always a fixed sequence of actions.

-If we apply searching algorithm, the branching factor will be mxn (there are mxn actions to take for each state).

2.2 Problem formulation

- Initial state: zero matrix mxn or mxn matrix with random value 0 and 1.
- Actions/Transition model: (state1, press_button(i,j) -> state2)
- Goal test: check if the whole network is ON, or x = 1 ∀x ∈ matrix
- Path cost: each button pressed costs 1, the path cost is the total button pressed in the solution

3. Selecting the algorithms to be used for solving the problem

For solving the problem, we apply 3 algorithms: iterative Deepening Limit Search, Backtracking Algorithm and Greedy Algorithm.
The reasons why we chose this algorithm are:
- IDS is complete and optimal because each action costs 1 so we do not need to be concerned about the optimization of the algorithm, and due to its iterative property, if there is a solution, IDS is able to find it
- Backtracking algorithm is also complete and optimal because it is a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign, moreover we will iterate the depth of the algorithm to guarantee its optimization.
- Normal DFS may not be complete if there is a loop of action happening, it is also not optimal and may take many unnecessary actions.
- We are not able to find a proper heuristic function for this problem so we do not apply A* search.
- Compared to BFS, IDS and Backtracking are better for this problem because we found that the depth that gives the solution is usually large, and the branching factor is relatively large (mxn), and BFS has to finish

generating all state at relatively large depth, and it has to do it for every depth, so we expect BFS to cost much more time than our algorithms.

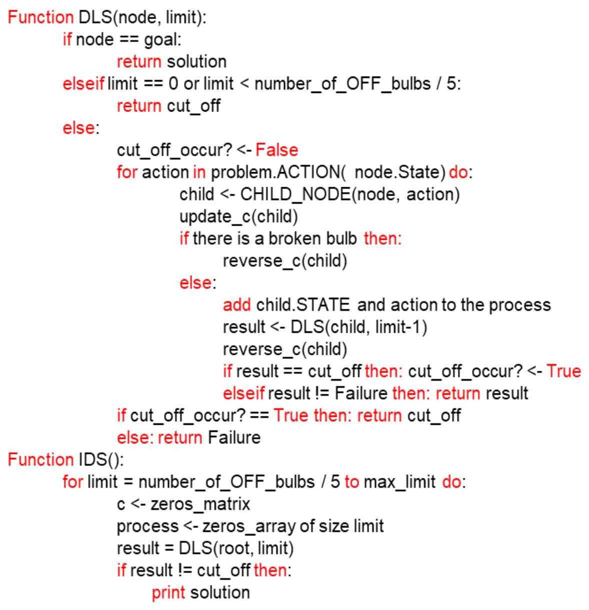
- It is too difficult to estimate a good depth limit for DLS, because the limit depends not only on the size of the matrix but also the initial state, for example a 3x3 matrix initially all 0 costs 5 steps, while a 4x4 matrix initially all 0 costs only 4 steps.
- We choose the Greedy algorithm even though it may not be complete, or optimal because it runs extremely fast compared to any searching algorithm above and be able to solve some problem instances with large size with the solution at deep levels, which cannot be found by searching algorithms.

3.1 Iterative Deepening Limit Search (IDS)

We apply the IDS algorithm into this problem:

- Goal check criteria is to check the state of the network is all on
- Beside the condition limit = 0, we have another condition limit is less than the number of off bulbs divided by 5 to cut off earlier.
- problem.ACTION(node.State) is a list of buttons to press for each state.
- After taking any action, we update the counting matrix C to check whether there is broken bulb or not, if there is we do not take that action and reverse C to its previous state, if there is not, we take that action for real and go deeper by 1 level.
- DLS repeats this process until we can find the solution or we reach the depth limit or we detect failure.
- IDS will call DLS iteratively by increasing limit by 1 after every DLS called.

IDS pseudo code



```
Function DLS(node, limit):
    if node == goal:
        return solution
    elseif limit == 0 or limit < number_of_OFF_bulbs / 5:
        return cut_off
    else:
        cut_off_occur? <- False
        for action in problem.ACTION( node.State) do:
            child <- CHILD_NODE(node, action)
            update_c(child)
            if there is a broken bulb then:
                reverse_c(child)
            else:
                add child.STATE and action to the process
                result <- DLS(child, limit-1)
                reverse_c(child)
                if result == cut_off then: cut_off_occur? <- True
                elseif result != Failure then: return result
        if cut_off_occur? == True then: return cut_off
        else: return Failure
Function IDS():
    for limit = number_of_OFF_bulbs / 5 to max_limit do:
        c <- zeros_matrix
        process <- zeros_array of size limit
        result = DLS(root, limit)
        if result != cut_off then:
            print solution
```

3.2 Constraint satisfaction programming (Backtracking.)

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

Moreover, thanks to the lemma below, the light bulb problem became a modified version of combinations enumerating problem:

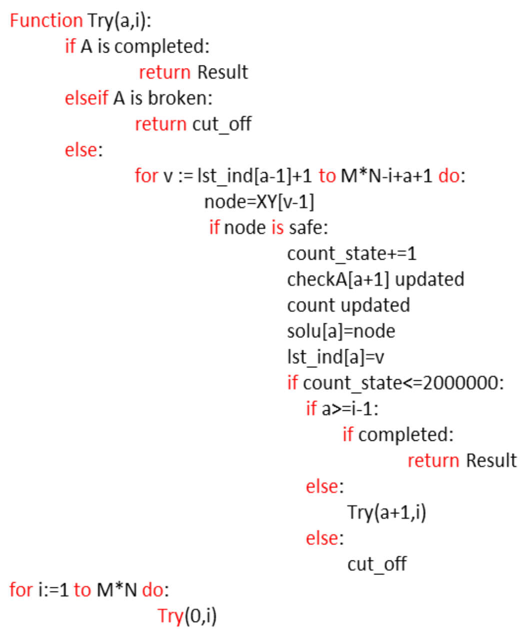
+We have the set of all buttons.

+We have S (a set of integers from 1 to mxn, each number is an index of a node).

+Use Backtracking algorithm to enumerate the subsets of S.

+With each subset of S, we have a list of indexes of buttons. Based on the lists of indexes, we can find the optimal solution for the problem.

Backtracking pseudo code



```
Function Try(a,i):
    if A is completed:
        return Result
    elseif A is broken:
        return cut_off
    else:
        for v := lst_ind[a-1]+1 to M*N-i+a+1 do:
            node=XY[v-1]
            if node is safe:
                count_state+=1
                checkA[a+1] updated
                count updated
                solu[a]=node
                lst_ind[a]=v
                if count_state<=2000000:
                    if a>=i-1:
                        if completed:
                            return Result
                    else:
                        Try(a+1,i)
                else:
                    cut_off
for i:=1 to M*N do:
        Try(0,i)
```

3.3 Greedy algorithm

With greedy algorithms, one attempts to construct a solution in stages. At each stage a decision is made that appears to be the best at that stage. This decision is made based on a greedy criterion. A decision made in one stage is not changed in a later stage, so each decision should assure feasibility. Greedy algorithms have intuitive appeal, however sometimes they fail to compute an optimal solution. Before designing a greedy algorithms, we first need to identify 5 features of a problem :

1. A candidate set from which an optimal solution should be built
2. A selection function (greedy criteria) which at each stage selects from the candidate set the most promising element that has not yet been selected
3. A feasibility function checks whether adding a particular candidate satisfies the feasibility constraints, if not discards the candidate and never considers it again
4. A solution function that checks whether the current set of chosen candidates is a complete solution 5. An objective function which assigns value to each partial or complete solution.

=> Apply to our problem

1.The mxn buttons constitute the candidate set.

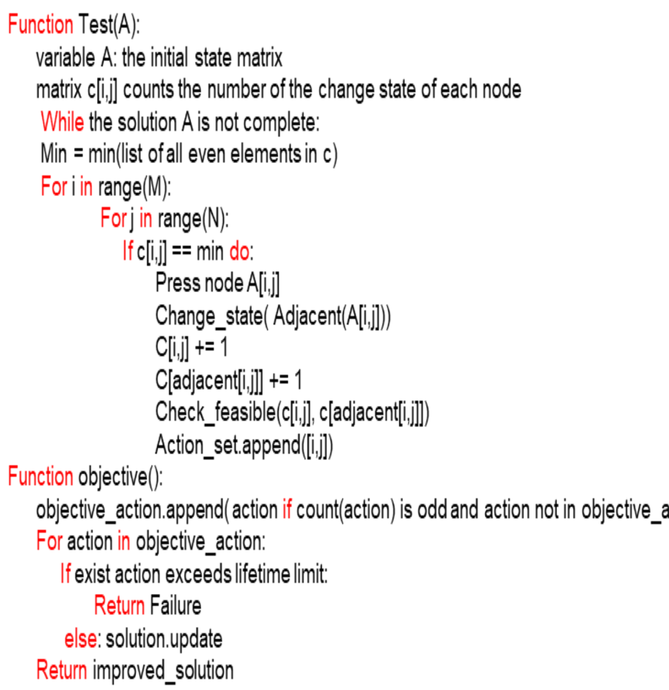
2.Greedy criteria: among the candidate whose state has been changed even times, selcect the one with minimum times of state changing.

3.Feasibility function: Consider the button I, the number of state changes has not exceeded the changing limit.

4.Solution: all the button’s values are 1.

5.Objective function: cut off unnecessary buttons by removing the duplicate buttons and buttons appear even times in the solution to

Greedy pseudo code



```
Function Test(A):
    variable A: the initial state matrix
    matrix c[i,j] counts the number of the change state of each node
     While the solution A is not complete:
     Min = min(list of all even elements in c)
     For i in range(M):
             For j in range(N):
                 If c[i,j] == min do:
                     Press node A[i,j]
                     Change_state( Adjacent(A[i,j]))
                     C[i,j] += 1
                     C[adjacent[i,j]] += 1
                     Check_feasible(c[i,j], c[adjacent[i,j]])
                     Action_set.append([i,j])
Function objective():
    objective_action.append( action if count(action) is odd and action not in objective_a
    For action in objective_action:
       If exist action exceeds lifetime limit:
           Return Failure
       else: solution.update
    Return improved_solution
```

4. Implementing the algorithms to be used for solving the problem

4.1 Main difficulty:

For our problem, we have faced some difficulties

- ❖ For the IDS algorithm, it is tricky to implement the action transition model, we noticed that the number bulbs whose states are changed when a certain button is pressed is different for different buttons (or action in short). For example, if we press a button in the middle of the network, it changes the state of 5 bulbs, but if we press a button on one edge, it changes 4 bulbs, and that number is 3 for a button in the corner. Moreover, if not every bulb has another bulb above it (or a left bulb, or a right bulb, or a bulb below). If we implement this by using normal IF-THEN, there are many cases to take into account. Instead, we decide to extend the parent state to (m+2)x(n+1) bulbs, we treat all "new" bulbs as normal bulbs, but they will not have the breaking down constraint, by doing this every "old" bulb will have exactly 4 neighboring bulbs, and we just need to change the states of 5 bulbs for every actions, after that we will take out the mxn matrix in the middle to be our next child state.
- ❖ For Backtracking, this problem is considered as listing permutation with repetition of a set of action with mxn entries, which is expected to cost $O((mxn)^{mxn})$.
- ❖ The biggest issue we encountered with IDS and Backtracking is that they are both time consuming algorithms: both runs with huge time complexity. Even with the breaking down constraint to detect failure sooner, it still takes too much time for data with large sizes, or for

some specific data whose sizes are relatively small but the solution is at a deep level.

❖ In term of greedy algorithm:

  The main difficulty for greedy algorithms is to find the proper greedy criteria (selection function). We propose some selection functions. For example, one criteria is iterating through the matrix and finding the first 0 elements, then changing the state of this bulb and its adjacency until a complete solution is found (no 0 elements anymore). After testing, we choose the selection as above because it can solve more problems than others (more complete) and the time cost is also very small, but for some specific problem, the result cannot be found or it is not optimal, then we have to come up with a method to process this result.

4.2 A helpful lemma

To reduce the time cost, we came up with a lemma:

- Lemma: If we can find a solution for this problem which is a sequence of buttons to press, the permutation of this sequence is also a solution.
- Proof:

  - Since the initial state of every bulb is known. The state of a bulb depends on how many times it is changed: if the state is changed by an even number of times, it stays the same; if it is changed by an odd number of times, it is changed from OFF to ON and vice versa.

  - Assume we have a solution, says S = {b_1, b_2, …, b_n}.

  - Thus, we know the position of b_1, b_2, …, b_n and all the affected bulbs by pressing any bi (i =1,2, …, n) in the network.

- For every bulb j, let T_j be the number of times its state is changed by doing S (in its order), and t_{ji} is a variable such that $t_{ji} = 1$ if pressing button bi changes the state of bulb j, $t_{ji} = 0$ otherwise

- For all bulbs j in the network, we have $T_j = t_{j1} + t_{j2} + t_{j3} + \ldots + t_{jn}$ (press bi in the order of S), and due to the commutative properties of the summation, T_j stays the same for any permutation of S

Then the lemma is proved!

- Application: if we can find a solution $S = \{b_1, b_2, b_3, \ldots, b_n\}$ we know that S is the optimal solution iff it contains no duplicate buttons (buttons that have the same position in the network). Let us say S has 2 duplicate buttons b_i and b_j, then $S' = \{b_i, b_j, b_1, b_2, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{j-1}, b_{j+1}, \ldots, b_n\}$ is also a solution, but if we press b_i and b_j, since they are identical, the state of the whole network will not change, then $S'' = \{b_1, b_2, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{j-1}, b_{j+1}, \ldots, b_n\}$ is also solution which costs 2 less steps. **This tells us we should avoid pressed buttons in the algorithm, and if there exists an optimal solution, its length cannot exceed mxn. This can cut off much unnecessary time.**

4.3 Algorithm improvement

- ❖ Iterative deepening limit search
    - o WeWith all methods above, we are able to save a lot of time and can deal with data with solutions at up to 12th level of depth, which we could not do before. want the buttons in the solution to be ordered in terms of the order they appear in the network (from left to right, from top to bottom), and due to the lemma, the action list we can take for a certain state will begin

from the button after the buttons that lead us to that state (the previous transition models).

- We noticed that a button can turn on maximum of 5 bulbs so if limit (the remaning number of action to take) is less than the number of current OFF bulbs divided by 5, we cut off right away.

❖ Backtracking

As mentioned above, with this lemma, the light bulb problem becomes a modified version of combinations enumerating problems. The lemma helps us to reduce complexity of Backtracking algorithm from $O((mxn)^{mxn})$ to $O((2^{mxn}))$, it is now just listing subsets of the action set with mxn entries. Although the Backtracking algorithm is still time consuming, it can solve almost all problems with data size 4x5 or lower.

❖ Greedy algorithm

Thanks to the lemma, after finding the solution, we can improve it to be better. We apply this lemma by selecting the action that appears odd times in the initial solution and removing all the duplication. After that, we get an "better" solution, but still can be non-optimal in some casesour solution is considerably improved .

5. Comparing the results of the algorithms used for solving the problem

5.1 Providing quantitative performance indicators

❖ Total number of problem instances: 95

Percentage where the algorithm successfully solved the problem:

- IDS: 83/95 = 87.36%
- Backtracking: 82/95 = 86.31%
- Greedy:75/95 = 78.94%

❖ Average time and space complexities observed in practice given in this the table in the link below:

Data experiment.xlsx

5.2 Explaining these results

- If any of our algorithms finds a solution, it will return Found , for IDS and Backtracking if there is no solution for an instance, they will return Failure.

- IDS and Backtracking are both time consuming as mentioned above, so we decide to stop those algorithm after 2 millions states generated for IDS and 2 millions entries in subsets generated for Backtracking, in this cases, they will return Time Out

- Greedy algorithm may not find the solution, which it does in some instances, the completedness of it depends on the "goodness" of the greedy criteria. Because Greedy algorithm is all about making decision without coming back (no undo) it may keep making wrong decisions repeatedly, so we have to estimate when to stop it. We did this by making a limit for the number of times the state of a bulbs is changed. Problem with this is when the algorithm returns Failure we do not know that it is because there is no solution or our greedy criteria is not good enough.

- We conclude that the IDS and Backtracking algorithm are more complete than Greedy algorithm because the Greedy criteria is not suitable for all the instances of the problem while IDS and Backtracking always find the solution if there exists one.

- Greedy algorithm on average runs much faster than other 2 algorithms, IDS on average runs in longest time, Backtracking usually gives out solution and detects failure faster than IDS, but not as fast as Greedy algorithm, this is reasonable because their time complexities in theory is ordered in the same way.

- IDS and Backtracking have the same space complexity in theory. Since both algorithms are BFS-fashion, which can be implemented by queue, the maximum length of the queue, which is O(mxnxd) is the number of states stored in memeory, each state cost O(mxn) so the total space complexity is $O((mxn)^2xd)$

Criterion	Iterative Deepening	Backtracking	Greedy
Complete?	Yes	Yes	No
Time	$O(mxn^d)$	$O(2^{mxn})$	O(mxnxk)
Space	$O((mxn)^2xd)$	$O((mxn)^2xd)$	O(mxn)
Optimal?	Yes	Yes	No

6. Conclusion and possible extensions.

- In conclusion, we think that we have finished our project well, eventhough there are still many issues need to be tackled. However, we found that searching algorithms cost too much time.

- By achieving this project, we are able to understand clearly how some searching algorithms work, how to analysis their time and space complexity, how to choose proper seacrhing algorithm for problem like this, and most importantly how to implement those algorithm.

- We are able to come up with some methods that can improve the time costs of our algorithm. Thanks to those methods, our programs cost much less time than they did without them.

Our main problem is still time cost:

-If we had more time, we would have wished to be able to find some heuristic functions to reduce time cost even further, also we would have wished to find a better greedy criteria for our Greedy algorithm, we actually came up with several, and then chose the one that gives most complete results, but it is still not complete in every cases. We really want to improve greedy algorithm because it gives out results really fast and can deal with instances that searching algorithm can not.

- We think that our probem can also be solved by building linear model and use Linear programming, it is a really good algorithm and does not cost as much time as searching. But it is not a searching algorithm and we did not have enough time.

7. List of tasks

7.1 Programming tasks

- Implementing Iterative Deepening Limit Search: Nguyễn Minh Châu
- Implementing Backtracking Algorithm: Chu Hoàng Dương
- Implementing Greedy Algorithm: Nguyễn Hoàng Anh
- Generating random instances of the problem: Phạm Như Thuấn

7.2 Analytic tasks

- Nguyễn Hoàng Anh:choosingselecting Greedy algorithm to solve this problem, doing slides in power point for our presentation, writing parts, making the demo video, writing parts 3.3, 5, 6, 7, 8 in, designing the report.
- Nguyễn Minh Châu: proposing our subject, choosingselecting IDS algorithm to solve this problem, writing parts 1, 2, 3.1, 4.1, 4.3, and 6 in the report.
- Chu Hoàng Dương: choosing Backtracking algorithm to solve this problem, proposing and proving the lemma, writing parts 3.2, 4.2 in the report
- Phạm Như Thuấn: result statistics, recording video, analyzing the the table of data result

8. List of bibliographic references

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 3rd edition (2009)
- Stuart J. Russell and Peter Norvig, Artificial Intelligence, A Modern Approach, 3rd edition (2010)