Illustrative study of Artificial Intelligence Algorithms

Prof. Ajay Talele[1], Jairaj Jangle[2], Yashwant Kapgate[3], Mayur Deogade[4,] Prajwal Langde[5]
[1,2,3,4,5] *Vishwakarma Institute of Technology, Pune*

***Abstract*- This paper illustrates some Artificial Intelligence algorithms such as Heuristics, Breadth-first search(BFS), Depth First Search(DFS), Cryptarithmetic Problem, A* Problem. Water Jug problem is a famous problem in the field of Artificial intelligence and the method used in our project is Heuristics. The BFS and DFS Algorithm is implemented in our project. In this paper, we proposed a solution to some of the basic AI problems and then optimized by using UI. We also showed in each algorithm the solution is reached faster than similar algorithms. Additionally, we have also developed a graphical user interactive application to visualize the above-mentioned algorithms.**

Index Terms- Artificial intelligence, Heuristics, Water Jug Problem, Cryptarithmetic Problem, Breadth First Search(BFS), Depth First Search(DFS), A*.

I. INTRODUCTION

It has become necessary to develop some methods to solve a certain type of problems which can not be efficiently solved by conventional computer programming. Let us take an example of a chess playing program. Theoretically, there are around 10^{27589} legal moves possible in a game of chess. Now it is practically impossible to program a machine to handle such a large number of possible moves. Thus, it is necessary to develop a machine which is able to play the game of chess in a way similar to humans. Thus came the evolution of Artificial Intelligence algorithms.

There is no such term called "Unified Artificial Intelligence Algorithm". There are many artificial intelligence algorithms developed but it is not necessary that one algorithm can solve any sort of problem. Thus it becomes necessary to be able to recognize which algorithm to be applied to which problem. In this paper, we have presented four types of artificial algorithms namely Heuristics, Depth-First, Breadth-First Search, Cryptarithmetic problem-solving algorithm, A* algorithm. In addition to the comparative discussion of the above-mentioned algorithms, we have also developed an application for visualizing the above-mentioned algorithms. This gives the user an easier understanding of the functioning of each algorithm. In the application, we have added animations as each algorithm transits through steps of the algorithms.

Section I: Introduction introduces the paper topic. Section II: Heuristic Algorithm discusses the analysis of heuristic algorithm and how water jug problem can be solved using heuristics. Section III: Cryptarithmetic problem discusses the crypt-arithmetic problem in depth and how it can be solved. Section IV: Depth-First Search explains DFS algorithm with diagrams and how it proceeds and the same goes for the next section, Section V: Breadth-First Search. Next, Section VI: A* algorithm discusses the types of pathfinding algorithms and discusses A* algorithm in depth with pseudo code. Section VII: Result provides some screenshot of our application with an explanation of each.

II. HEURISTICS ALGORITHM

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of shortcut utilized for optimality, completeness, accuracy, or precision for speed. A Heuristic (or a heuristic function) analyzes search algorithms. At each branching step, it evaluates the information that is available and makes a decision on which branch is to be followed. It does so by ranking alternatives. The Heuristic is any device that is often effective but does not guarantee to work in every case.

Solving the Water Jug Problem

Heuristic solutions rely upon problem-solving methods resulted from practical experience, they are rules of thumb that try to estimate an acceptable, computationally effective solution, as close as possible to the optimal one. However, they do not

present the guarantee that a solution will be obtained at all. They are only believed to work in most situations. There are specific heuristics that are applicable to particular a situation, and general heuristics that are applicable to a large range of problems. Let us consider a heuristic for the generalized water jugs problem: if the rest is lower than the capacity of jug A, the problem has no solution (obvious); if the rest is a multiple of the greatest common divisor of A and B, the problem has no solution (the solution results from combining the differences between A and B; e.g. with A=6 and B=3, it is impossible to obtain a rest equal to 2, because all the differences will be multiples of three); since the solution of this problem is generally based on the water exchange between the two jugs in order to obtain the differences, there are two strategies that lead to the solution for any problem. Let valA and valB be the quantities of the two jugs, respectively. Let rest be the quantity that is to be obtained in the first jug.

Strategy 1: while (valA != rest) { if (valB != B && valA == 0) FillA(); if (valB == B) EmptyB(); Pour FromAtoB(); }

Strategy 2: while (valA != rest) { if (valA != A && valB == 0) FillB(); if (valA == A) EmptyA(); PourFromBtoA(); }

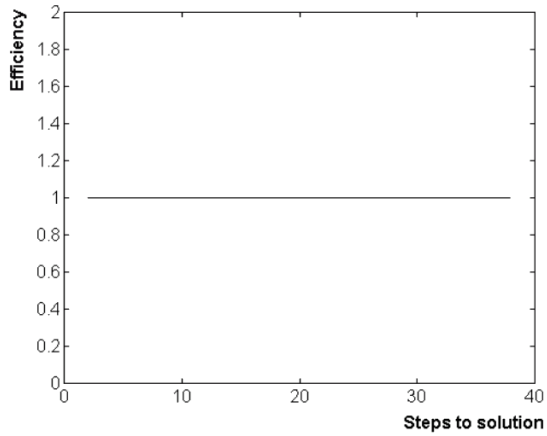

Fig. 1: Efficiency of the Heuristic method

In this way, building a solution tree is not required and solving is done without additional memory costs. However, since they are heuristic solutions, the adequate strategy should be determined from case to case. For example, if A=6, B=5 and the rest is 2, the first strategy finds the solution in 6 steps, while the latter does in 14 steps. If A=11, B=6 and the rest is 7, the first strategy finds the solution in 24 steps, and the latter in only 8.

III. CRYPT ARITHMETIC PROBLEM

Cryptarithmetic problem can be referred to as a puzzle which consists of arithmetic operator commonly addition where the digits are replaced by English alphabets. The main Motto of the cryptarithmetic problem is to map the letters to some digits ranging from 0 to 9 with the constraints provided by arithmetic that no two alphabets can have the same digit or value from 0 to 9. All the alphabets given in the problem statement should have a unique digit different from one another and also the arithmetic operator should be considered while evaluating the solution[5]. This problem was very popular during the 1930s in the Sphinx which is a Belgium journal of recreational mathematics [6]. One of the best solutions to the Cryptarithmetic problem was published in July 1924 by Henry Dudeney in the issue of Strand Magazine [7] which is shown in Fig 2.

```
  SEND
+MORE
------
MONEY
```

Fig 2. Cryptarithmetic problem example

In the example above, the arithmetic operator addition is used for assigning the digits to letters in a way such that the solution exists and is arithmetically correct. Here the solution can be O=0, M=1, Y=2, E=5, N=6, D=7, R=8 and S=9 which gives the correct solution for the problem. Hence the result can be as shown in Fig. 2.

```
 9567
+1085
-----
10652
```

Fig 3. An acceptable solution to the problem in Fig.2

Pseudo code for Crypt arithmetic problem

1. Start by examining the rightmost digit of the topmost row, with a carry of 0.[8]
2. If we are beyond the leftmost digit of the puzzle, return true if carry doesn't exist, otherwise false
3. If we are currently trying to assign a char in one of the addends

- If char already assigned, just recur on the row beneath this one, adding value into the sum
- If not assigned, then, for (every possible choice among the digits not in use) make that choice and then on the row beneath this one, if successful, return true
  if unsuccessful, unmake assignment and try another digit
- Return false if none of the assignment works

4. Else if try to assign a char in the sum
5. If char assigned & matches correct, recur on next column to the left with carry if success return true,
6. If char is assigned & if it doesn't matches, return false
7. If char is unassigned & if the correct digit is already used, return false
8. If char unassigned & correct digit unused, assign it and recur on next column to left with carry, If success return true
9. Return false to trigger backtracking

Constraints of the Cryptarithmetic problem are as follows:

1. The arithmetic operations are in decimal numbers only so digits should vary from 0 to 9; therefore, there must be a maximum of ten different letters in the overall strings which is being used.
2. All the same letters of the alphabets should use the same unique digit and no two different letters of the alphabets should be bounded to the same decimal digit.
3. As the words will represent numbers, the first letter of the word cannot be assigned to zero.
4. The resulting numbers should satisfy the problem, meaning that the result of the first two numbers (operands) under the specified arithmetic operation (plus operator) should be the third number.

IV. DEPTH FIRST SEARCH (DFS)

Depth-First Search (DFS) is one of this searching method that is included of blind search. That is, the search is done by ensuring all vertices have been visited, but without the exact solution or an intermediate solution. Especially for AI (Artificial Intelligence) problems searching or tracking is one method to solve general problems, Searching is a process of looking for a solution of a problem through a set of possible state space (state space) [9].
The main factor which is owned by the DFS algorithm has its ability to find the nodes or vertices that have not visited in depth, and data structure as per Cormen et al. Stack have a unique role in DFS that served to 'remember' or store a value in the algorithm reaches a vertex particular. The DFS algorithm uses the idea of backtracking so it's called a recursive algorithm also. Initially exhaustive searches of all the nodes by going ahead is done else by backtracking. This recursive nature of DFS can be implemented using stacks[10]. Nodes are visited as per Fig.3.

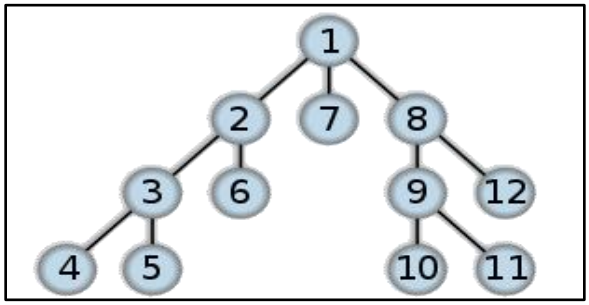

Fig. 3: Order in which nodes are visited. (DFS)

The DFS algorithm can be described as :

- Start from a starting node and push all its adjacent nodes into a stack.
- Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
- Check if the nodes that are visited marked then repeat this same process until the stack is empty.

Checking the nodes that are visited will prevent you from visiting the same node more than once. If marking is not done then the same nodes are visited more than once, then this algorithm may end up in an infinite loop.
For algorithms like finding connected components, topological sorting, generating words in order to plot the limit set of a group, finding bi-connectivity in the graph includes depth-first search as a building block [11].

V. BREADTH FIRST SEARCH (BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.

It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
In this algorithm breadthwise traversing of the graph is done as follows:

1. First, move horizontally and visit all the nodes of the current layer
2. Move to the next layer

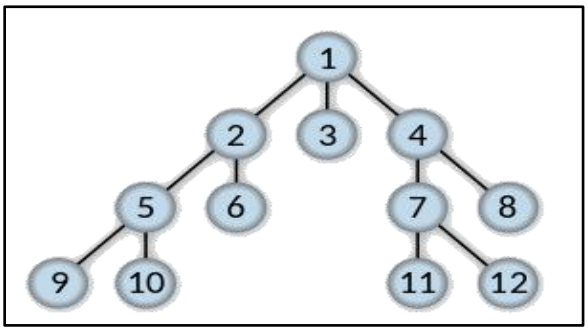

Fig. 4: Order in which nodes are visited(BFS)

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of the same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order[12]. Nodes are visited as per Fig.4.
To make this process easy, a queue is used to store the node and mark it as 'visited' until all its neighbors (vertices that are directly connected to it) are marked[11][13]. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neighbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.
Breadth-first search can be used to solve many problems in graph theory such as Copying Garbage collection, Cheney's algorithm, construction of the *failure function* of the Aho-Corasick pattern matcher, Testing bipartiteness of a graph.

VI. A* ALGORITHM

A pathfinding algorithm searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with the intent of finding the most optimal route. Pathfinding algorithm consists of two phases
1. Finding the path between two given nodes.
2. Finding the most optimal path between the two nodes.

Types of pathfinding algorithm:
1. Exhaustive.
2. Eliminatory.

Let's first look at what Exhaustive algorithms are.

1. Exhaustive pathfinding algorithms

The most primitive pathfinding algorithms are Breadth-first and Depth-first. This algorithm finds the path between two nodes by exhausting all the possibilities, starting from a given node. These algorithm run in O(|V| + |E|), or linear time, where V is the number of vertices, and E is the number of Edges between vertices.
Another yet complicated pathfinding algorithm for finding the optimal path is the Bellman-Ford algorithm, which yields the time complexity of O(|V||E|), or quadratic time.
However, it is not necessary to examine all the possible paths to find the optimum one.

2. Eliminatory pathfinding algorithms

The eliminatory algorithms consist of Dijkstra and A* pathfinding algorithms. In this project, A* algorithm is implemented due to the high time cost of Dijkstra and the inability of Dijkstra to evaluate negative edge weights.

A star algorithm:
A* is a variant of Dijkstra's algorithm. A* assigns a weight to each open node which is equal to the weight of the edge to that node in addition to the approximate distance between that node and the finish. This approximate distance is found by the heuristic 1 and represents a minimum possible distance between that node and the end. This enables it to eliminate longer paths once an initial path is found. If there exists a path of length x between the start and finish, and the minimum distance between a node and the finish is greater than the distance x, that node does not need to be examined. A* uses this heuristic to improve on the behavior relative to Dijkstra's algorithm. When the heuristic evaluates to zero, A* is equivalent to Dijkstra's algorithm[15]. As the heuristic estimate increases in value and gets closer to the true distance, A* continues to find optimal paths but runs faster because it needs to examine fewer nodes. When the value of the heuristic

is exactly the true distance, A* examines the fewest nodes.

A* f cost calculation equation: f(n) = g(n) + h(n)

Where:

g is the cost of the present node from the star point excluding the unwalkable path.

h is the cost of the present node from the ending point including the unwalkable path.

f is the resultant cost.

```
OPEN //the set of nodes to be evaluated CLOSED
//the set of nodes already evaluated Add the start node
to OPEN
loop
current = node in OPEN with the lowest f_cost //path
has been found
remove current from OPEN add current to CLOSED
if current is the target node return
for each neighbor of the current node
if neighbor is not traversable or neighbor is in
CLOSED skip to the next neighbor
if new path to neighbor is shorter OR neighbor is not
in OPEN
set f_cost of neighbor
set parent of neighbor to current if neighbor is not in
OPEN
add neighbor to OPEN
```

Fig 5: A* pseudo code[14]

VII. RESULT

We have developed an application for this comparison. We have used QT platform to integrate various graphical functionalities with C++ as backend. Our application presents the user with various Artificial algorithms in an interactive demeanor. The application was developed on Linux x64 platform and we have also deployed a stand-alone executable package of our application which can run on any Linux x64 based machine, for this purpose we have used. AppImage container.

This application was developed with an objective to present the users new to Artificial intelligence subject, an interactive and graphical method of how various artificial algorithms functions. With this application one can readily understand the working of the above-discussed algorithms namely: Water Jug problem using Heuristics, Crypt-Arithmetic Depth-First search, Breadth-First search and A* with a very intuitive animation. Our application also lets the user enter the input values to initiate the algorithm working as an example the user can enter the initial water levels of the first jug and the second jug along with the volume of water that needs to be measured.

Following are some screenshots of our application.

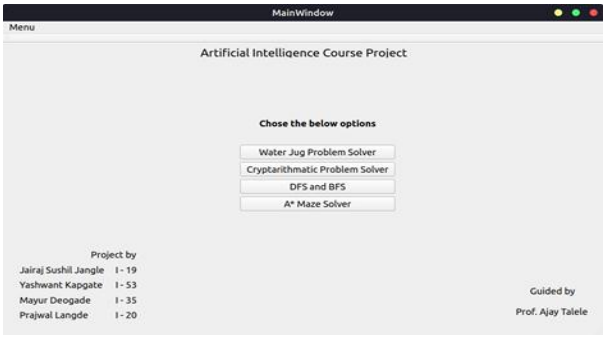

Fig. 5: Home screen of our application.

The Home Screen of our application consists of four buttons in the middle and a menu bar at the top, These four buttons let the user switch to various algorithms offered in our application. In the menu bar, the user can jump from any screen to another screen.

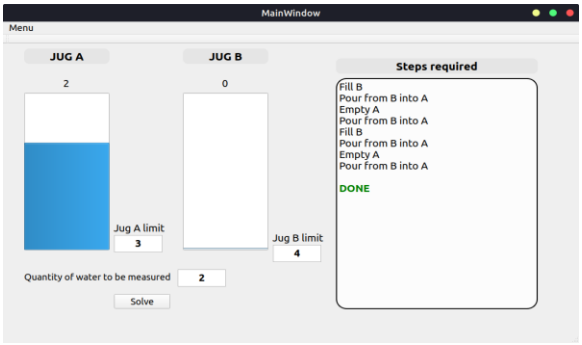

Fig. 6: Water Jug Problem

For developing the above water jug like graphics, we have used a QProgressBar element in the application. We have also added some animations when the algorithms transitions through steps.

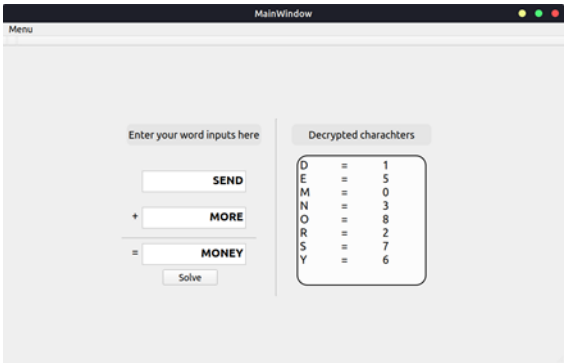

Fig. 7: Cryptarithmetic Problem

To make the Crypt-Arithmetic screen of the application, we have included three input fields in which the user can enter the input query to the cryptarithmetic problem. After clicking the Solve button at the bottom of the text fields, the application proceeds to solve the problem and displays the solution in the text box at the right side of the screen.

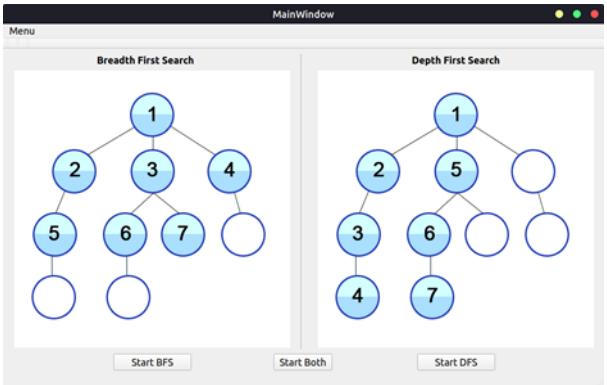

Fig. 8: Depth-First and Breadth-First search.

For presenting the DFS and BFS algorithm we have used a tree structure. In this tree, we have ten nodes for both DFS and BFS. After clicking on the Start button at the bottom of the tree diagram, our application shows how both the algorithms proceeds, DFS prioritizes depth while BFS prioritized Breadth.

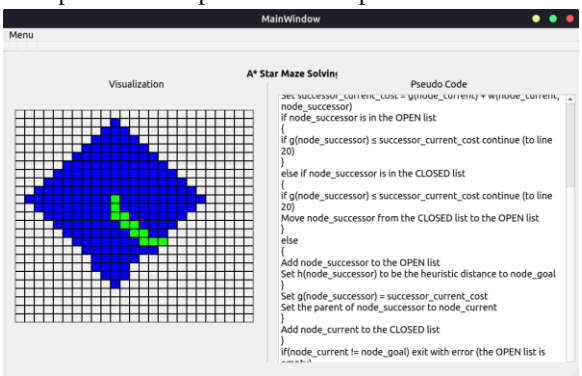

Fig. 9: A* Algorithm.

The final page of our application presents the A* algorithm. To present the A* algorithm we have added a map structure and added a start point and an end point. The blue region shows the searched region and the green blocks shows the final calculated path. On the right side, we have included the pseudo code of the A* algorithm.

VIII. CONCLUSION

In the paper, we have described various artificial intelligence: Heuristics, Cryptarithmetic solving an algorithm, Depth-First Search, Breadth-First Search, A* algorithm in brief with some application to which each of them can be applied. This paper gives an idea to recognize and relate algorithms to certain AI problems. We have also developed an application to give the user a graphical description of the working of each mentioned algorithm.

REFERENCES

[1] Harris R. Creative Problem Solving: A Step by Step Approach, Pyrczak, Los Angeles, 2002.

[2] Boldi P., Santini M., Vigna S. Measuring with Jugs, Theoretical Computer Science, no. 282, pp. 259-270, 2002.

[3] Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, New.

[4] A Heuristic for Solving the Generalized Water Jugs Problem Bulletin of the Polytechnic Institute of Iasi, tome LI (LV), section Automatic Control and Computer Science.

[5] Vinod Goel, Sketches of thought, MIT Press, 1995, pp. 87 and 88.

[6] Bonnie Averbach and Orin Chein, Problem Solving Through Recreational Mathematics, Courier Dover Publications, 2000, pp. 156.

[7] H. E. Dudeney, in Strand Magazine vol. 68 (July 1924), pp. 97 and 214

[8] Pseudo code for Crypt arithmetic problem from https://www.google.com/amp/s/www.geeksforgeeks.org/solving-cryptarithmetic-puzzles-backtracking-8/amp/

[9] Desiani, Anita and Arhami, Muhammad. 2006. Konsep Kecerdasan Buatan. Yogyakarta, Indonesia: C.V Andi Offset.

[10] Ms. Avinash Kaur, Ms. Purva Sharma, Ms.Apurva Verma, International Journal of Scientific and Research Publications, Volume 4, Issue 3, March 2014.

[11] Hyejeong Ryu and Wan Kyun Chung, ”Local Map-based Exploration using a Breadth-First Search Algorithm for Mobile Robots” Springer, INTERNATIONAL JOURNAL OF PRECISION ENGINEERING AND MANUFACTURING Vol. 16, No. 10, pp. 2073-2080

[12] Maciej Kurant, Athina Markopoulou, Patrick Thiran,” On the bias of BFS (Breadth First

Search)" 2010 22nd International Teletraffic Congress (lTC 22), IEEE, Amsterdam, Netherlands.

[13] Maryia Belova, Ming Ouyang " Breadth-First Search with A Multi-Core Computer", IEEE, Conference, Lake Buena Vista, FL, USA

[14] A star algorithm through https://www.youtube.com/watch?v=-L-WgKMFuhE by Sebastian Lague.

[15] For detail A* understanding: http://theory.stanford.edu/~amitp/GameProgramming/Heuristic