PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DHANKAWADI, PUNE

ARTIFICIAL INTELLIGENCE & ROBOTICS MINI-PROJECT REPORT ON

"HILL CLIMBING ALGORITHM"

SUBMITTED BY

NIKHIL JAIN	41425
ATHARVA JAGTAP	41423
VAIBHAV MARATHE	41433

Under the guidance of
Prof. D.D.Kadam

DEPARTMENT OF COMPUTER ENGINEERING Academic Year 2020-21

HILL CLIMBING ALGORITHM

PROBLEM STATEMENT:

Hill climbing algorithm is one of the common applications of artificial intelligence. Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. Hill Climbing is mostly used when a good heuristic is available. Use Heuristic Search Techniques to implement Hill Climbing Algorithm

ABSTRACT:

Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

H/W &S/W REQUIREMENTS:

OS: Windows 10/ Ubuntu LTS
RAM: 4GB
Hard Drive: 500 GB
Eclipse
Required Java libraries

INTRODUCTION:

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, **mathematical optimization problems** implies that hill-climbing solves the problems where we need to maximize or

minimize a given real function by choosing values from the given inputs. Example-Travelling Salesman Problem where we need to minimize the distance traveled by the salesman.

- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in a **reasonable time.**
- A **heuristic function** is a function that will rank all the possible alternatives at any branching step in a search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

OBJECTIVE:

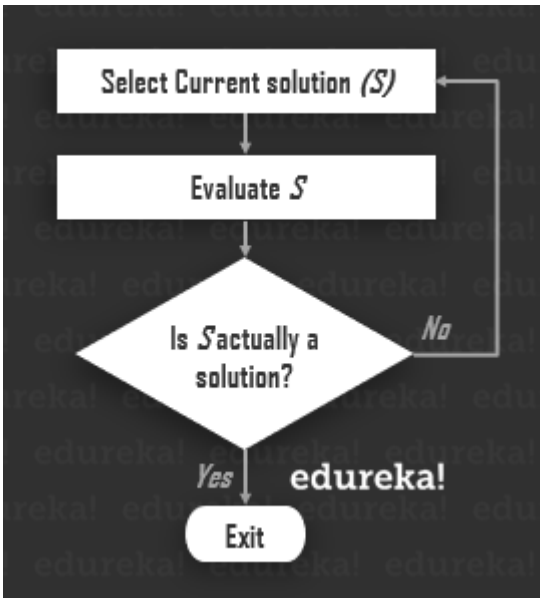
We will be able to:

1. Learn and implement Hill Climbing Algorithm
2. Use Heuristic Search Algorithm

SCOPE:

Hill Climbing is an iterative algorithm that can be used to find the weights θ for an optimal policy. It is a relatively simple algorithm that the Agent can use to gradually improve the weights $\boldsymbol{\theta}$ in its policy network while interacting with the Environment. As the name indicates, intuitively, we can visualize that the algorithm draws up a strategy to reach the highest point of a hill, where $\boldsymbol{\theta}$ indicates the coordinates of where we are at a given moment and G indicates the altitude at which we are at that point

SYSTEM ARCHITECTURE:



SOURCE CODE:

HillClimbingAlgorithm.java

```
import java.util.*;

public class HillClimbingAlgorithm {
	state gstate, cstate, sstate;
	Scanner sc = new Scanner(System.in);
	ArrayList<state> ngb = new ArrayList<state>();

	HillClimbingAlgorithm() {
		gstate = new state();
		cstate = new state();
		sstate = new state();
	}

	void display(state s) {
		int k = 0;
		for (int j = 0; j < 3; j++) {
			for (int i = 0; i < 3; i++) {
				System.out.print(" " + s.arr[k]);
				k++;
			}
			System.out.println();
		}
```

```
}

void input() {

	System.out.println("Enter the start state");
	for (int i = 0; i < 9; i++) {
		sstate.arr[i] = sc.nextInt();
	}

	System.out.println("Enter the goal state");
	for (int i = 0; i < 9; i++) {
		gstate.arr[i] = sc.nextInt();
	}

}

int h(state s) {
	int hvalue = 0;
	for (int i = 0; i < 9; i++) {
		if (s.arr[i] != gstate.arr[i]) {
			hvalue++;
		}
	}

	return hvalue;
}

int blpos(state s) {
	for (int j = 0; j < 9; j++) {
		if (s.arr[j] == 0) {
			return j;
		}
	}
	return 0;
}

void Movegen(state s) {
	int p = blpos(s);
	// int temp=0;
	ngb.clear();
	if (p % 3 != 0) {
		state n1 = new state(s);
		// temp=n1.arr[p];
```

```
                n1.arr[p] = n1.arr[p - 1];
                n1.arr[p - 1] = 0;
                n1.h = h(n1);
                ngb.add(n1);
        }

        if (p < 6) {
                state n1 = new state(s);
                // temp=n1.arr[p];
                n1.arr[p] = n1.arr[p + 3];
                n1.arr[p + 3] = 0;
                n1.h = h(n1);
                ngb.add(n1);
        }
        if (p > 2 && p < 9) {
                state n1 = new state(s);
                // temp=n1.arr[p];
                n1.arr[p] = n1.arr[p - 3];
                n1.arr[p - 3] = 0;
                n1.h = h(n1);
                ngb.add(n1);
        }
        if (p % 3 != 2) {
                state n1 = new state(s);
                // temp=n1.arr[p];
                n1.arr[p] = n1.arr[p + 1];
                n1.arr[p + 1] = 0;
                n1.h = h(n1);
                ngb.add(n1);
        }

}

int lowestscore() {
        int i = 0, min = 999;
        for (int j = 0; j < ngb.size(); j++) {
                if (min > ngb.get(j).h) {
                        min = ngb.get(j).h;
                        i = j;
                }
        }

        return i;
}
```

```
	state hillclimbing() {
		int low = 0, done = 0;
		state n, nn;
		sstate.h = h(sstate);
		sstate.paraent = null;
		n = sstate;
		Movegen(n);
		low = lowestscore();
		nn = ngb.get(low);
		display(n);
		System.out.println();
		while (nn.h < n.h) {
			display(nn);
			System.out.println();
			nn.paraent = n;
			n = nn;

			Movegen(n);
			low = lowestscore();
			nn = ngb.get(low);
		}
		return nn;

	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		HillClimbingAlgorithm ob = new HillClimbingAlgorithm();
		ob.input();
		ob.hillclimbing();

	}

}
```

state.java

```
class state {
	int arr[];
	state paraent;
	int h = 0;

	state(state s1) {
		this.arr = new int[9];
		for (int i = 0; i < s1.arr.length; i++) {
			this.arr[i] = s1.arr[i];
		}

	}

	state() {
		arr = new int[9];
	}
}
```

TEST CASES:

Enter the start state:
1 0 3
8 2 4
7 6 5
Enter the goal state:
1 2 3
8 0 4
7 6 5

Output:
1 0 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5

CONCLUSION:

Thus, we have used the Heuristic Search Technique to implement Hill Climbing Algorithm.

REFERENCES:

1. https://www.edureka.co/blog/hill-climbing-algorithm-ai/
2. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/
3. https://www.javatpoint.com/hill-climbing-algorithm-in-ai