

| **Title:** Implementation of Informed search algorithm( GBFS/A\*) |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Comparison and analysis of informed search algorithms

**Expected Outcome of Experiment:**

| **Course Outcome** | **After successful completion of the course students should be able to** |
| --- | --- |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**](http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf)
4. [**http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**](http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html)
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf** [**http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**](http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf)
6. [**http://wiki.answers.com/Q/Informed\_search\_techniques\_and\_uninformed\_search\_techniques**](http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:** Problem solving, state-space trees, problem formulation, goal based agent architecture

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

The AI researchers have come up many algorithms those operate on state space tree to give the result. Goal based agent architectures solve problems through searching or planning. Depending on availability of more information other than the problem statement decides if the solution can be obtained with uninformed search or informed search.

Its fact that not all search algorithms end up in giving the optimal solution. So, it states the need to have a better and methodological approach which guarantees optimal solution.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Heuristic, Informed search, greedy best first search, A\* search

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Informed searching techniques**

* Greedy best first search
* A\*

**Chosen Problem statement:**

**8 puzzle problem**

The 8 puzzle problem is a classic problem in artificial intelligence and computer science that involves a small 3x3 grid with eight numbered tiles and one empty space. The goal is to rearrange the tiles from a given initial configuration to a desired goal configuration by sliding the tiles into the empty space.

States: numeric board with one blank tile.

Initial State:Given problem

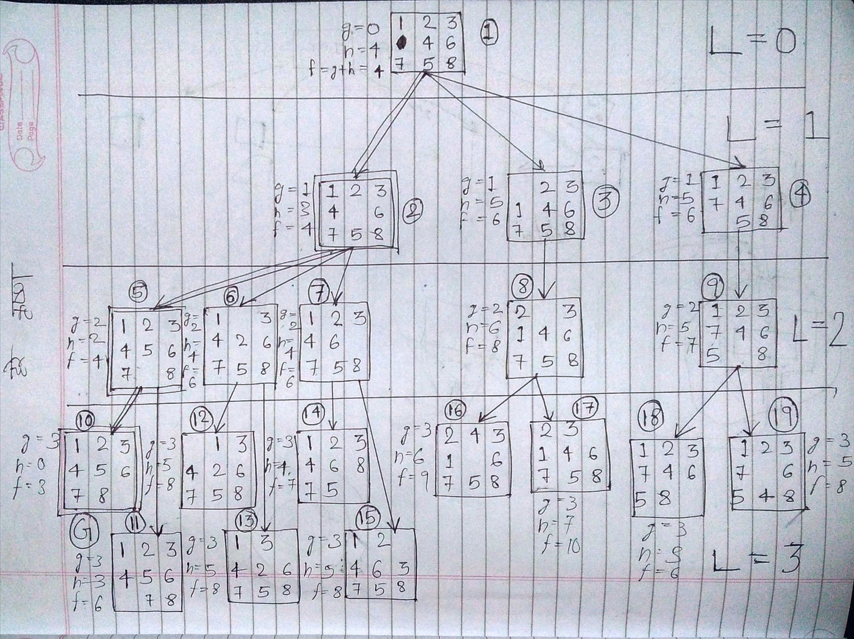
Actions: Shift tile left, right, top or bottom

Transition Model: Returns the board with a tile shifted.

Goal test: all numbers are in order

Path cost: Path cost to go from one node to the next.

Heuristic: Number of displaced tiles.

**State-space tree :**

**Solution with of GBFS/ A\* on the state-space tree:**

**A\* algorithm**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

15 tile game

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class Main

{

public static void main(String[] args) {

System.out.println("Hello World");

int [][] array ={{1,2,3,4},{5,6,7,8},{9,-1,10,12},{13,14,11,15}};

/\*swap(array,3,2,3);

print(array);

unswap(array,3,2,3);

print(array);\*/

System.out.println(check(array));

solve(array,-1,100,0);

}

//i column

// j row

public static int solve(int [][]array,int lastOp,int lastError,int level){

int p=-1;

int q=-1;

//find position of blank

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

if(array[i][j]==-1){

p=i;

q=j;

}

}

}

int maxError = 10000;

int[][] arraycpyminerror=copy(array);

int mink=-1;

for(int k=1;k<=4;k++){

int[][] arraycpy=copy(array);

if(IsComplementary(k,lastOp)==true){

continue;// dont repeat

}

int swapped=swap(arraycpy,p,q,k);

if(check(arraycpy)==1){

System.out.println("Answer->>");

print(arraycpy);

return 0;

//soln

}

int error = check(arraycpy) + level;

// check with least cost

if(error<maxError){

//print(array);

//System.out.println("Error reduced from "+lastError+" to "+check(arraycpy));

//print(arraycpy);

//System.out.println("Next->>");

arraycpyminerror = arraycpy;

mink = k;

maxError = error;

}

// print(arraycpy);

}

System.out.println("Level "+level);

print(arraycpyminerror);

if(solve(arraycpyminerror,mink,check(arraycpyminerror),level-1)==1){

return 0;

}

return -1;

}

public static boolean IsComplementary(int k, int dir){

if(dir==1 && k==2){//up

return true;

}

if(dir==2 && k==1){//down

return true;

}

if(dir==3 && k==4){//left

return true;

}

if(dir==4 && k==3){//right

return true;

}

return false;

}

public static int unswap(int [][]array,int i,int j,int dir){

if(dir==1){//up

return swap(array,i-1,j,2);

}

if(dir==2){//down

return swap(array,i+1,j,1);

}

if(dir==3){//left

return swap(array,i,j-1,4);

}

if(dir==4){//right

return swap(array,i,j+1,3);

}

return 0;

}

public static int swap(int [][]array,int i,int j,int dir){

int temp=-5;

if(dir==1){//up

if(i==0){

return 1;

}

temp = array[i-1][j];

array[i-1][j]=-1;

}

if(dir==2){//down

if(i==3){

return 1;

}

temp = array[i+1][j];

array[i+1][j]=-1;

}

if(dir==3){//left

if(j==0){

return 1;

}

temp = array[i][j-1];

array[i][j-1]=-1;

}

if(dir==4){//right

if(j==3){

return 1;

}

temp = array[i][j+1];

array[i][j+1]=-1;

}

array[i][j]=temp;

return 0;

}

public static int check(int[][] array){

int error=0;

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

if(array[i][j]!=4\*(i)+j+1) {//check values

// System.out.println(j+""+i+" "+array[i][j]);

error++;

//wrong

}

}

}

return error;

}

public static int [][] copy(int [][]array){

int [][] newarr = new int [array.length][array[1].length];

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

newarr[i][j]=array[i][j];

}

}

return newarr;

}

public static void print(int[][]array){

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

System.out.print(array[i][j]+",");

}

System.out.println();

}

}

}

**Greedy**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15 tile game

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class Main

{

public static void main(String[] args) {

System.out.println("Hello World");

int [][] array ={{1,2,3,4},{5,6,7,8},{9,-1,10,12},{13,14,11,15}};

/\*swap(array,3,2,3);

print(array);

unswap(array,3,2,3);

print(array);\*/

System.out.println(check(array));

solve(array,-1,100,0);

}

//i column

// j row

public static int solve(int [][]array,int lastOp,int lastError,int level){

int p=-1;

int q=-1;

//find position of blank

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

if(array[i][j]==-1){

p=i;

q=j;

}

}

}

int maxError = 10000;

int[][] arraycpyminerror=copy(array);

int mink=-1;

for(int k=1;k<=4;k++){

int[][] arraycpy=copy(array);

if(IsComplementary(k,lastOp)==true){

continue;// dont repeat

}

int swapped=swap(arraycpy,p,q,k);

if(check(arraycpy)==1){

System.out.println("Answer->>");

print(arraycpy);

return 0;

//soln

}

int error = check(arraycpy) ;

// check with least cost

if(error<maxError){

//print(array);

//System.out.println("Error reduced from "+lastError+" to "+check(arraycpy));

//print(arraycpy);

//System.out.println("Next->>");

arraycpyminerror = arraycpy;

mink = k;

maxError = error;

}

// print(arraycpy);

}

System.out.println("Level "+level);

print(arraycpyminerror);

if(solve(arraycpyminerror,mink,check(arraycpyminerror),level-1)==1){

return 0;

}

return -1;

}

public static boolean IsComplementary(int k, int dir){

if(dir==1 && k==2){//up

return true;

}

if(dir==2 && k==1){//down

return true;

}

if(dir==3 && k==4){//left

return true;

}

if(dir==4 && k==3){//right

return true;

}

return false;

}

public static int unswap(int [][]array,int i,int j,int dir){

if(dir==1){//up

return swap(array,i-1,j,2);

}

if(dir==2){//down

return swap(array,i+1,j,1);

}

if(dir==3){//left

return swap(array,i,j-1,4);

}

if(dir==4){//right

return swap(array,i,j+1,3);

}

return 0;

}

public static int swap(int [][]array,int i,int j,int dir){

int temp=-5;

if(dir==1){//up

if(i==0){

return 1;

}

temp = array[i-1][j];

array[i-1][j]=-1;

}

if(dir==2){//down

if(i==3){

return 1;

}

temp = array[i+1][j];

array[i+1][j]=-1;

}

if(dir==3){//left

if(j==0){

return 1;

}

temp = array[i][j-1];

array[i][j-1]=-1;

}

if(dir==4){//right

if(j==3){

return 1;

}

temp = array[i][j+1];

array[i][j+1]=-1;

}

array[i][j]=temp;

return 0;

}

public static int check(int[][] array){

int error=0;

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

if(array[i][j]!=4\*(i)+j+1) {//check values

// System.out.println(j+""+i+" "+array[i][j]);

error++;

//wrong

}

}

}

return error;

}

public static int [][] copy(int [][]array){

int [][] newarr = new int [array.length][array[1].length];

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

newarr[i][j]=array[i][j];

}

}

return newarr;

}

public static void print(int[][]array){

for(int i =0;i<4;i++){

for(int j=0;j<4;j++){

System.out.print(array[i][j]+",");

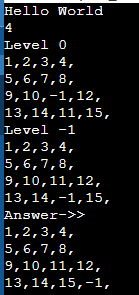
}

System.out.println();

}

}

}

****

**Comparison of performance of Greedy and A\* Algorithm:**

For our problem statement, both greedy and A\* algorithms are equally good. Since the path levels for the graph have equal g values, A\* and greedy search algorithms have similar implementation.

**Properties of A\* algorithm:**

A\* algorithm is an informed search algorithm that combines the advantages of both breadth-first search and greedy best-first search by using both the cost-so-far function (g) and a heuristic function (h) to guide the search towards the goal efficiently. Here are the key properties of the A\* algorithm:

1. **Completeness**: A\* algorithm is complete, meaning it is guaranteed to find a solution if one exists, given finite search space and appropriate heuristic function.
2. **Optimality**: A\* algorithm is optimal, meaning it finds the shortest path from the initial state to the goal state, provided that the heuristic function is admissible (never overestimates the cost to reach the goal) and consistent (satisfies the triangle inequality).
3. **Admissibility**: If the heuristic function used in A\* is admissible, it will always underestimate the cost to reach the goal, ensuring that A\* expands the fewest number of nodes necessary to find the optimal solution.
4. **Time and Space Complexity**: The time and space complexity of A\* algorithm depend on various factors, including the size of the search space, the quality of the heuristic function, and the implementation details. In general, A\* has exponential time complexity but is often more efficient than uninformed search algorithms due to its informed nature.
5. **Open and Closed Lists**: A\* algorithm maintains two lists during the search process: the open list, which contains nodes that are currently being considered for expansion, and the closed list, which contains nodes that have already been expanded. This helps prevent revisiting already explored nodes and ensures efficient exploration of the search space.

**Post lab Objective questions**

1. **A heuristic is a way of trying**
   1. To discover something or an idea embedded in a program
   2. To search and measure how far a node in a search tree seems to be from a goal
   3. To compare two nodes in a search tree to see if one is better than the other
   4. Only (a) and (b)
   5. Only (a), (b) and (c).

**Answer: e**

1. **A\* algorithm is based on** 
   1. Breadth-First-Search
   2. Depth-First –Search
   3. Best-First-Search
   4. Hill climbing.
   5. Bulkworld Problem.

**Answer: c**

1. **What is a heuristic function?** 
   1. A function to solve mathematical problems
   2. A function which takes parameters of type string and returns an integer value
   3. A function whose return type is nothing
   4. A function which returns an object
   5. A function that maps from problem state descriptions to measures of desirability.

**Answer: e**

**Post Lab Subjective Questions:**

1. **How best-first-search algorithm supports heuristic evaluation function?**

Best-first search algorithm utilizes a heuristic evaluation function to guide the search towards the most promising nodes in the search space. This function estimates the cost or value of reaching a goal state from any given node. By prioritizing nodes with lower heuristic values, the algorithm explores paths likely to lead to the goal more efficiently. Heuristic evaluation enables best-first search to intelligently select nodes, balancing exploration and exploitation to efficiently navigate complex search spaces. This approach is particularly effective in problem-solving scenarios where accurate cost-to-goal estimations are available, optimizing the search process towards finding solutions efficiently.

1. **Find a good heuristic function for following**
   1. **Monkey and Banana problem**
   2. **Travelling Salesman problem**

For the Monkey and Banana problem, a heuristic function could be the Manhattan distance between the monkey and the banana, considering obstacles if present. This heuristic provides an estimate of the minimum number of moves required for the monkey to reach the banana, guiding the search towards the shortest path.

For the Traveling Salesman Problem (TSP), a good heuristic function could be the nearest neighbor heuristic. This heuristic selects the nearest unvisited city from the current city at each step, approximating a solution.

1. **Define the heuristic search. Discuss benefits and short comings.**

Benefits of heuristic search:

Efficiency: Heuristic search algorithms can significantly reduce the search time by focusing on promising solutions, especially in large and complex search spaces.

Domain-specific knowledge: Heuristics leverage domain-specific knowledge to guide the search, allowing algorithms to make informed decisions and prioritize exploration.

Flexibility: Heuristic search algorithms can be tailored to different problem domains by adjusting the heuristics used, making them versatile for a wide range of applications.

Shortcomings of heuristic search:

Lack of optimality: Heuristic search algorithms do not always guarantee finding the optimal solution, as they make decisions based on estimations rather than exhaustive exploration.

Sensitivity to heuristics: The quality of the heuristic function greatly influences the performance of heuristic search algorithms. Poorly chosen heuristics can lead to suboptimal or inefficient solutions.

Complexity: Designing effective heuristics can be challenging and may require substantial domain knowledge. Additionally, implementing heuristic search algorithms correctly can be complex, especially for certain problem domains.

Risk of getting stuck: Heuristic search algorithms may get stuck in local optima or plateaus, especially if the chosen heuristics are not effective at guiding the search towards the global optimum.