# Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem

Debasish Nayak (110cs0081)

**Department of Computer Science and Engineering**

**National Institute of Technology Rourkela**

**Rourkela - 769008, India**

# Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem:

*Thesis submitted in*

May 2014

*to the department of*

Computer Science and Engineering of

National Institute of Technology Rourkela

*in partial fulfillment of the requirements*

*for the degree of*

Bachelor of Technology

*in*

Computer Science and Engineering

*by*

Debasish Nayak

[Roll No: 110cs0081]

*Under the guidance of*

Prof. B. Majhi

# Declaration of Authorship

I hereby declare that all the work contained in this report is my own work unless otherwise acknowledged. Also, all of my work has not been previously submitted for any academic degree. All sources of quoted information have been acknowledged by means of appropriate references.

*Debasish Nayak*

Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769 008, India. www.nitrkl.ac.in

## Certificate

This is to certify that the project entitled *Analysis and Implementation of Admissible Heuristics in 8-Puzzle Problem* by *Debasish Nayak* is a record of his work carried out under my supervision in partial fulfillment of the requirements for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*.

*Bansidhar Majhi*

## Acknowledgment

I have taken a lot of deliberations in this venture. But it wouldn't have been conceivable without the help and backing of numerous people. I want to enlarge my true appreciation and thank them.

I take this opportunity to express my profound gratitude and deep regards to my guide Prof. B Majhi sir for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessings, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to all the professors of the Department of Computer Science and Engineering, NIT Rourkela for instilling in me the basic knowledge about the field that greatly benefitted me while carrying out the project and achieving the goal.

Finally, I dedicate the thesis to my parents for their love, support and encouragement without which this would not have been possible.

*Debasish Nayak*

# Abstract

N-puzzle problem has been one of the basic problem since the beginning of artificial intelligence. The most popular version of n-puzzle among people is 8-puzzle problem. It consists of an area divided into 3x3 grid containing 8 numbered (to identify) tiles and one empty grid. We are given an initial state and we have to reach the goal state which is also specified. In this project, we have used various informed search methods like a*algorithm, ida* algorithm to solve the puzzle. Various heuristic involved in the informed search like number of misplaced tiles, Manhattan distance were analyzed; Manhattan distance being one of the most popular ones. Drawbacks of the heuristics are mentioned and an improvement in Manhattan distance heuristic is implemented.

# Contents

**Declaration of Authorship**……………………………………………………….. 3

**Certificate**……………………………………………………………………. 4

**Acknowledgment**……………………………………………………………. 5

**Abstract**……………………………………………………………………... 6

**List of tables**………………………………………………………………. 9

**1.Introduction**…………………………………………………………… 10

1.1 8 Puzzle (The problem)…………………………………………… 10

1.2Objective………………………………………………………….. 10

1.3 Thesis Outline…………………………………………………….. 10

**2.Literature Review**……………………………………………………. 11

**3.Methods and Proposed Work**………………………………………. 13

3.1 Search Space and Search Trees…………………………….... 13

3.2 Informed Search……………………………………………….. 15

3.2.1 Heuristics……………………………………………... 15

3.2.2 Algorithms of Informed Search……………………………. 16

3.2.2.1 A star(A*) Algorithm……………………………... 16

3.2.3 Admissibility of an algorithm………………………………. 19

3.3 Detecting Unsolvable Puzzles…………………………………….. 20

3.3.1Odd Board Size (when √ (n+1) is odd)……………………… 22

3.3.2 Even board size (when √ (n+1) is even)…………………… 23

**4.Implementation**……………………………………………………….. 24

4.1A* algorithm as implemented……………………………………….. 25

4.2 IDA*(Iterative Deepening A*)……………………………………….. 26

4.3 Solvability Check…………………………………………………….. 27

4.4 Heuristics…………………………………………………………… 28

4.4.1 Number of Misplaced Tiles……………………………….. 28

4.4.2 Sum of Manhattan Distances………………………………. 29

4.5 Algorithm for Calculating the Linear Conflict 32

Heuristic…………...
4.7 Comparing the Performance of the Three Heuristics…………….. 35
4.8 Screenshots……………………………………………………….. 36
4.9 Dominating Heuristic…………………………………………... 39
**5.Conclusion**........................................................................................... 41

**Bibliography**……………………………………………………………… 42

## List of Figures

| | | |
|---|---|---|
| 3.1 | Fig.1 Search tree for 8-puzzle………………………………………….. | 14 |
| 3.3 | Fig.2 Problem with path length 31……………………………………… | 20 |
| 3.3 | Fig.3 Unsolvable puzzle examples……………………………….... | 21 |
| 3.3.1 | Fig.4 Unsolvable puzzle examples……………………………….. | 22 |
| 3.3.2 | Fig.5 Even board size solvability……………………………………. | 23 |
| 4.3 | Fig.6 Solvability Check screenshot……….....……………………... | 27 |
| 4.4.1 | Fig.7 Number of misplaced tiles………….…………………… | 28 |
| 4.4.2 | Fig.8 Sum of Manhattan distance……………………………………. | 29 |
| 4.4.2 | Fig.9 Solving using Manhattan distance…….……………………. | 30 |
| 4.4.2 | Fig.10 Shortcomings of Manhattan distance…..…………………….. | 31 |
| 4.8 | Fig.11 Implementation of missing tile heuristic.…………………. | 36 |
| 4.8 | Fig.12 Sum of Manhattan distance heuristic…….……………….... | 37 |
| 4.8 | Fig.13 Removing Linear Conflict Heuristic……...……………….. | 38 |
| 4.9 | Fig.14 Dominating heuristics……………………...……………….. | 40 |

## List of tables

4.7 comparing the 3 heuristics…………………………………………35

# Chapter 1

## 1.Introduction

## 1.1 8 Puzzle (The problem)

The 8-puzzle is a **sliding tile puzzle** that is made up of a square structured frame area containing tiles in random/irregular order with one tile missing. It is a smaller version of the **15-puzzle** (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and numerous other names) . 8-puzzle is basically a frame area separated into 3x3 grids containing 8 tiles and one void grid. The tiles are marked in some way so as they can be identified. The tiles are mostly numbered from 1 to 8. We are given with an initial configuration of the tiles. A desired final configuration is also given. We have to reach the final state by sliding the tiles using the empty grid present.

## 1.2Objective

Solving the 8 puzzle using A*(a star) and IDA* algorithm. Various heuristics were used and the shortcomings of each were listed. An improvement in the existing Manhattan distance heuristic is implemented.

## 1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 discusses about the literature review while the various methods and proposed work is outlined in Chapter 3. Chapter 4 shows the implementation and results of the proposed work. Finally Chapter 5 concludes the entire thesis and gives a broad overview of the project.

# Chapter 2

## 2.Literature Review

### N-Puzzle(invention)

The puzzle was invented by Noyes Palmer Chapman, a postmaster in Canastota, New York, who indicated it to friends and companions, as right on time as 1874, a previous puzzle consisting of 16 grids that were to be assembled in 4 rows of equal length, each row adding to 34. Copies of the enhanced Fifteen Puzzle went to Syracuse, New York by way of Noyes' son, Frank, and from that point, by means of sundry connections, to Watch Hill, RI, and lastly to Hartford (Connecticut), when pupils in the American School for the Deaf started fabricating the puzzle and, by December 1879, offering them both provincially and in Boston, Massachusetts. Demonstrated to the puzzle, Matthias Rice, who had a fancy and extravagant carpentry business in Boston, began producing the puzzle in December 1879 and persuaded a "Yankee Notions" fancy products merchant to offer them under the name of "Gem Puzzle "so is another name of the puzzle. In January 1880, Dr. Charles Pevey, a dental specialist in Worcester, Massachusetts, earned some attention by offering a monetary reward for a solution to the 15 Puzzle.

The puzzle became popular in the U.S. in February 1880, Canada in March, Europe in April, yet that rage had about dispersed by July. It is thought the Japanese were not acquainted with the puzzle until 1889.

Noyes Chapman had applied for a patent on his "Block Solitaire Puzzle" on February 21, 1880. Nonetheless, that patent was rejected, likely in light of the fact that it was not sufficiently different from the August 20, 1878 "Puzzle-Blocks" patent (US 207124) granted to Ernest U. Kinsey.

From the beginning days of artificial intelligence the n-puzzle problem has been a standard problem as a certain amount of intelligence helps when trying to find a solution. Also the efficiency of the system can be measured using computing the time and space taken to solve the 15-puzzle problem. Various algorithms were designed and have been implemented in the computer for providing optimal solutions using lesser space possible.

# Chapter 3

# 3.Methods and Proposed Work

## 3.1 Search Space and Search Trees

### State space search:

It is a procedure used in computer science, especially in the field of artificial intelligence where progressive designs or states of a given state are considered, with the objective of discovering a *goal* and the path to goal state which has the required configuration. Problem is often modeled as a state space, a set of states that into which the problem can be configured. A graph is formed from the set of states where there is a connection between two states if there is an operation that can transform the current state to the other state. Important terms are:-

- A goal state test: Test of a state for equivalence with the goal state.
- A successor function (transition model): Given a state and action, generate successor state(child node).

**Variants:**

• Finding a path vs. an optimal path (if each step has a different cost i.e. a “step-cost”)

• Goal is already specified, we only have to find a path or optimal path – also called *Route planning*

• Path doesn’t matter, only the goal has to be found. – 8-puzzle, N-queens.

## Search trees

A search tree:

• Root which is the Start state .

• Child = successor state.

• Edges = actions and step-costs (may be specified alongside edges)

• Path from start state to a node is a “plan to get to that state” .

For most problems, the whole tree can never actually build as the tree would be very huge consuming high amount of time and memory.

Trivial search methods like depth first search(DFS), breadth first search(BFS) apply the search tree techniques and are called uninformed search methods.

## Search tree for a 8-puzzle problem

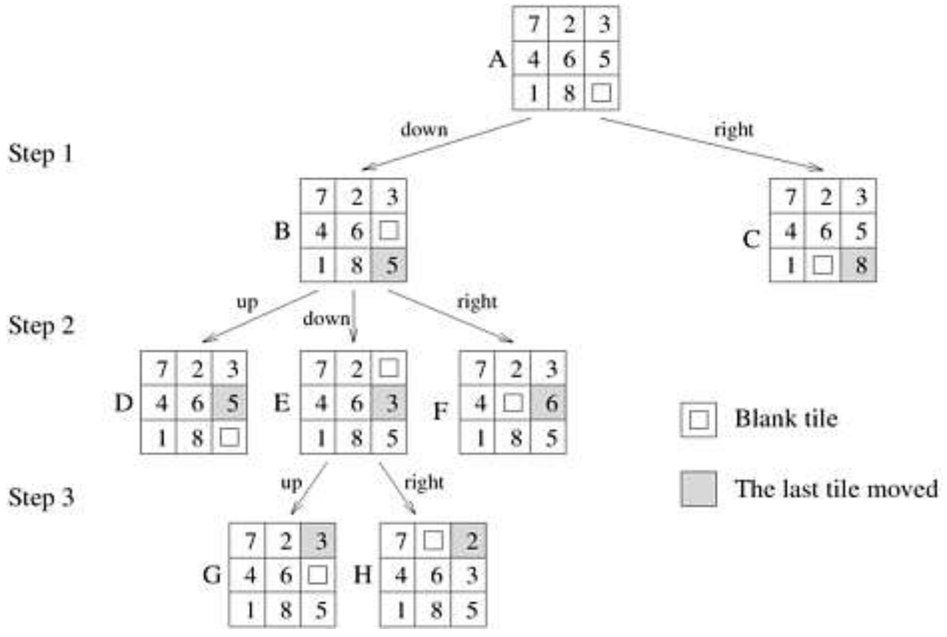


Fig 1.Search tree for 8-puzzle

## 3.2 Informed Search

All of the trivial search algorithms (BFS,DFS etc.) are examples of clueless (uninformed) search, where the procedure has no concept of the “right direction” towards goal . So it aimlessly staggers around until it happens to expand the goal node just by chance. We can improve much if there is a notion of which is the right way to go. A way to compute this kind of information and provide it to the algorithm is by using a heuristic, which determines the cost to get from any given state to a goal state. It is specified at the beginning of the informed search and the computation for the give start state is done.

### 3.2.1 Heuristic

- A **Heuristic** is a function that, when computed for a given state, returns a value that estimates the demerit of a given state, for reaching the goal state. Higher the value more is the estimated path length to the goal.
- Currently, the most used heuristic is the sum of Manhattan block distance.
- Also, it is possible to assign a weight to the heuristic, which is a factor applied to the h-value during the search.
- All types of search engines do not support this feature. The number of states to be generated can be cut-off at a given value, resulting in the search being abandoned at that point.
- In constructing heuristics, we regularly face a tradeoff between the exactness (accuracy) of a heuristic and how expensive it is to compute it.

## 3.2.2 Algorithms of Informed Search

### 3.2.2.1 A star(A*) Algorithm

It is a state space search algorithm. It is an algorithm that is regularly used searching of path and traversing of a graph, to plot an optimal path traversable between states (nodes). As it is efficient and accurate, it revels in boundless utilization. The A* algorithm integrates characteristics of uniform-cost search and heuristic based search to proficiently find optimally efficient path. A* algorithm is a best-first search algorithm in which the cost linked to a state is

f(n) = g(n) + h(n),where

- g(n) is the cost of the path traversed from the initial state to node n.
- h(n) is the estimated path-cost or the heuristic function cost from node n to the goal node.
- Thus, f(n) shows the lowest total cost possible for any path leading through node n to goal state when h(n) is the estimated remaining path-cost.
- If f value of various nodes of equal, such a stalemate situation is settled by taking the node with inferior heuristic estimate h(n) values(node ordering).
- The procedure continues till the node to be expanded is a goal node.

## Efficiency of A*

**A* is the fastest search algorithm**. There is no algorithm that can find a solution expanding lesser number of nodes than a* for a given heuristic.

## How quick is the algorithm?

- relies upon the heuristic function chosen.
  - If it is a not at all useful heuristic (h(n)evaluates to zero or negligible value compared to the path length), the algorithm degenerates to a uniform cost search algorithm. (simple search like dijkstra’s algorithm).
  - If it is an ideal heuristic function, there is no real search, it is just a walk through the tree along the shortest path to the goal state.
- Even if we can find an ideal heuristic, finding it would require finding the solution first. We always use heuristics that are between the ideal and the useless case. The length of the path and the time taken to find the solution depends on the exactness of the heuristic.
- A* algorithm expands the lowest number of nodes than any other search algorithm for a given heuristic.
- Some experimental results reported in Russell & Norvig (2002): A* with heuristic of sum of Manhattan distances performs up to 36,000 times better than a classical uninformed search algorithm (such as dijkstra’s algorithm).

**ANALYSIS of A***

- A* is optimally efficient. i.e. if there exists a path from start to goal node then a* guarantees to find the optimal path.
- It is complete i.e. if a solution exists then it is found.
- Complexity-As the algorithm is optimally efficient so other algorithm can guarantee to examine fewer nodes. However

- **Time complexity** –*exponential* O(b^d)

  where b=branching factor

  d= depth of the tree.

  unless h(n) is logarithmically accurate. The condition to be satisfied is

  |h(n) – h'(n)|<=O(log(h'(n))

Here h is our estimate and h' is the optimum path.

In real world all heuristics have a proportional error.

- **space complexity-** *exponentia*l O(b^d)

  It stores all the nodes generated in the open list.

So we run out of memory even before time possesses a problem.

**SHORTCOMINGS OF A***

- The primary drawback of A* algorithm as in case of most best-first search is its space requirement.
- Since the algorithm saves an entire open list in any case, A* algorithm is severely space-restricted in practice, and so is of very less practically use than other best-first search algorithms in present machines.
- For instance, it runs effectively when implemented for the 8- puzzle, while it accessible memory is exhausted in few minutes for the 15 puzzle.

## 3.2.3 Admissibility of an algorithm

- A graph search algorithm is admissible if it always provides a least cost path, i.e. an optimal solution, if a solution exists at all.
- However, an informed or educated search algorithms is admissible only if the heuristic used h(n) never overestimates the path-length to the goal state.
- For instance if a heuristic h' is known which always evaluates to the same value as that of the distance to goal-state, then for a heuristic h to be admissible it's computed value must be less than or equal to h'.

(A star)'s admissibility …

- A* is admissible only if the heuristic we used h(n) never over-estimates the path-length to the goal.

## 3.3 Detecting Unsolvable Puzzles

- There are 9!(362880)( permutation of 9 grids and 8 tiles) total states possible.
- The 8-puzzle if solvable can be solved in less than 31 single-tile moves or 24 multi-tile moves for every configuration of puzzle. (here for 8 puzzle 2 tiles moved at once).
- The lengths of optimal solutions ranges from 0 to 80 single-tile moves or 43 multi-tile moves for the 15-puzzle,.
- For higher versions of the n-puzzle, the solution may be found easily, but the generally quest is to find the shortest solution which is NP-hard.
- The state given below represents a worst case: transforming this state into the state on right (considered goal state) requires at least 31 actions, which is the maximum path length possible.

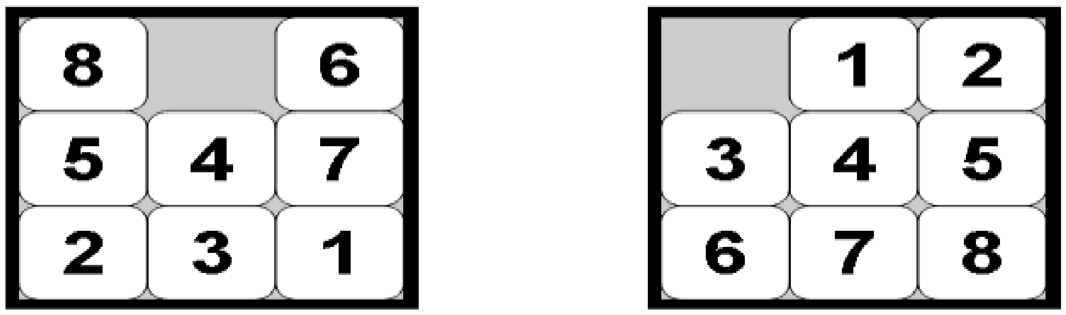


Fig.2 Problem with path length 31

- Only half of the initial positions of the 8-puzzle are solvable for a given goal state. There are 181440( 9!/2) states possible from a given state

- All the initial board configurations cannot be transformed to goal board by an order of possible moves. Example is from the states in left to states in right.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 8 | 7 | |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 15 | 14 | |

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15` | |

Fig.3 Unsolvable puzzle examples

Board configurations are classified into two parts based on their reachability to the goal state:-

- (i) Board configurations that can lead to the goal configuration.
- (ii)Board configurations that cannot lead to the goal configuration..

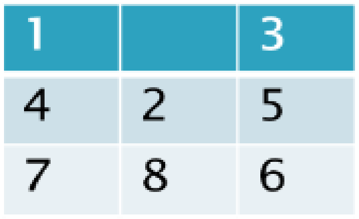
A board can be classified into one of the two parts without trying to find the solution. One important term is-

**Inversion** – It is any pair of numbered tiles *m* and *n* where *m* < *n* but *m* appears after *n* when reading the board conf. in row-major order (row 0, followed by row 1, and so forth).

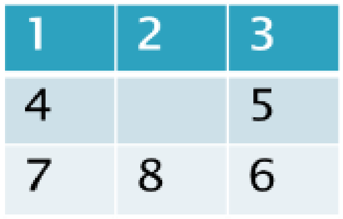
## 3.3.1Odd Board Size (when √ (n+1) is odd)

- An allowed move changes the inversions count by an even number.
- Thus, if a board has an odd number of inversions, then it cannot lead to the goal board by a sequence of legal moves because the goal board has an even number of inversions (zero).

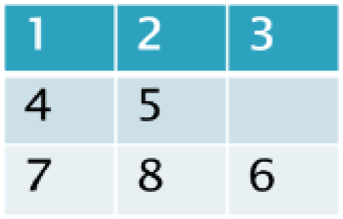
The converse is also true: If a board has an even number of inversions, then it *can* lead to the goal board by a sequence of legal moves.



| | | |
|---|---|---|
| 1 | | 3 |
| 4 | 2 | 5 |
| 7 | 8 | 6 |

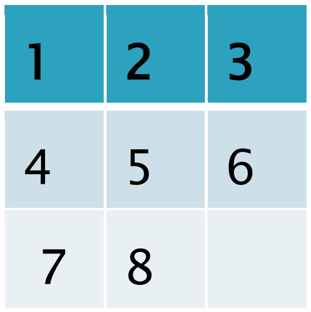


| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | | 5 |
| 7 | 8 | 6 |



| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | |
| 7 | 8 | 6 |

Inv= 4 inv=2 inv=2



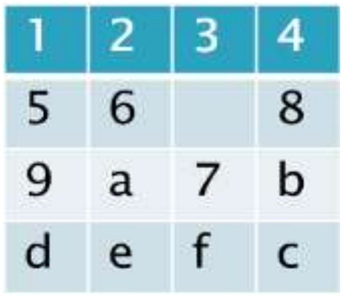
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Inv=0 goal state reached.(defined earlier)

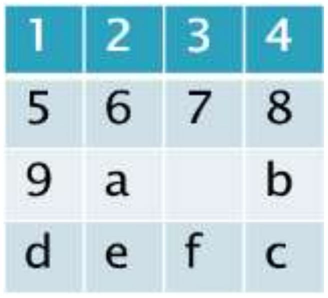
Fig.4 Counting number of inversions for odd board size

### 3.3.2 Even board size (when √ (n+1) is even)

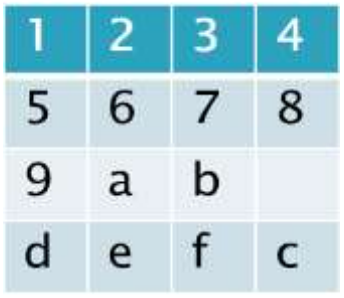
- If the board size *N* is an even integer, then the parity of the number of inversions is not invariant.
- However, the parity of the number of inversions plus the row of the blank square is invariant.
- Each legal move changes this sum by an even number.
- If the sum is even, then it cannot lead to the goal board by a sequence of legal moves; if this sum is odd, then it can lead to the goal board by a sequence of legal moves.



| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 5 | 6 |  | 8 |
| 9 | a | 7 | b |
| d | e | f | c |



| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
| 9 | a |  | b |
| d | e | f | c |



| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
| 9 | a | b |  |
| d | e | f | c |

Blank=1,inv =6 blank=2 ,inv=3 blank =2 ,inv=3 Sum=7
sum=5 sum=5

Fig.5 Even board size solvability

# Chapter 4

## 4.Implementation

The A* and the IDA* algorithm have been implemented.

Heuristics used are number of misplaced (out of order) tiles and sum of Manhattan distance.

The flaws of the heuristics are discussed and an improvement in Manhattan heuristic is done.

Performance of all the 3 heuristics used was compared.

Solvability check is implemented which prevents the program into going to run time error due to lack of memory when an unsolvable is generated as the puzzle is generated by using *rand()* function. It also prevents the time of the system in which it tried to solve the unsolvable puzzle.

All the implementations are done in c programming language. The screenshots of the output are included for all the heuristics.

## 4.1A* algorithm as implemented

There are two list maintained OPEN and CLOSED. This is to ensure a particular state does not get expanded more than once.

CLOSED: contains already expanded nodes.

OPEN: nodes on the frontier of the search tree.

Initially CLOSED is blank and OPEN has only the start node.

OPEN = <s,nil>    nodes and paths lengths are represented by lower case alphabets.

**Algorithm: a star**

```
1. while OPEN contains a node  do{
2.    delete from OPEN the node <j,p> with lowest f value.
3.     place <j,p> on CLOSED.
4.     if 'j' is  goal state ,return goal( path p).
5.    for each edge e connecting 'j' and 'k' with cost e  do{
6.      if < k,q> is in closed and {p|e } is cheaper than q
7.     delete 'k' from CLOSED.
8.     put <k,p|e > in OPEN.    End of if.
9.     else if <k,q > is in OPEN and {p|e} is cheaper than q
10.    replace <k,p|e > on OPEN.    End of else if
11.    else if k is not on OPEN
12.    put <k,p|e> on OPEN.  End of else if. End of for
13. end of while
14 .return failure
15. end of algorithm
```

## 4.2 IDA*(Iterative Deepening A*)

It is like iterative DFS (depth first search). Only the depth bound is modified to f-limit.

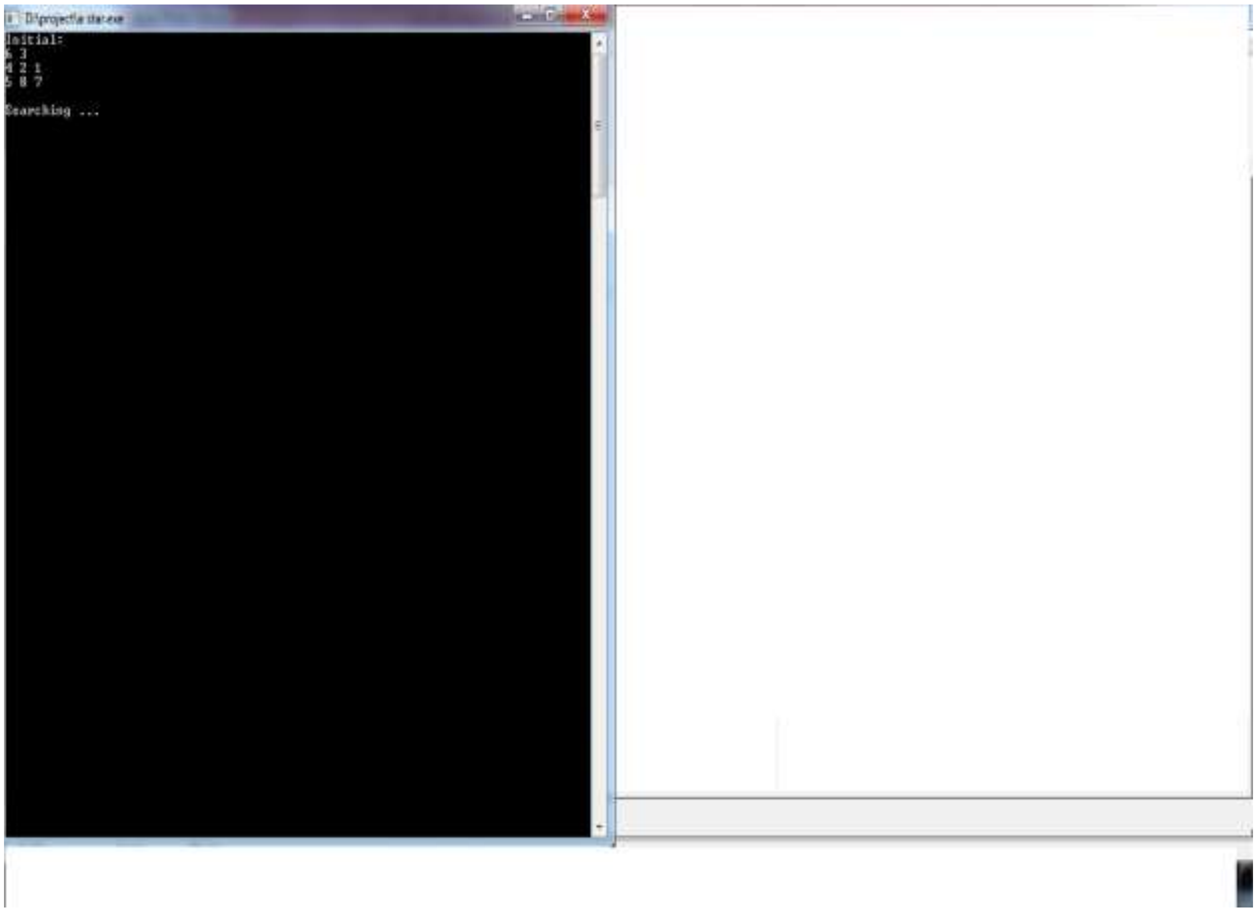
The algorithm-

- 1.start with the f(limit)=f(start)
- 2.Prune any node if f(node)>f(limit). Do a DFS.
- 3.Next f-limit=minimum of any node pruned . Go to step 1 and repeat the whole process until the goal node is found.

  It is implemented using heuristic of misplaced tiles.

## 4.3 Solvability Check

A solvability check was implemented which checks if the random puzzle generated is solvable or not. This check is done at the beginning of the execution to avoid unnecessary run time errors and crashing of the program. If this is not done then in 1 out of two cases during the start of execution there will be a run time error.  The program will crash and output screen will be as shown below.



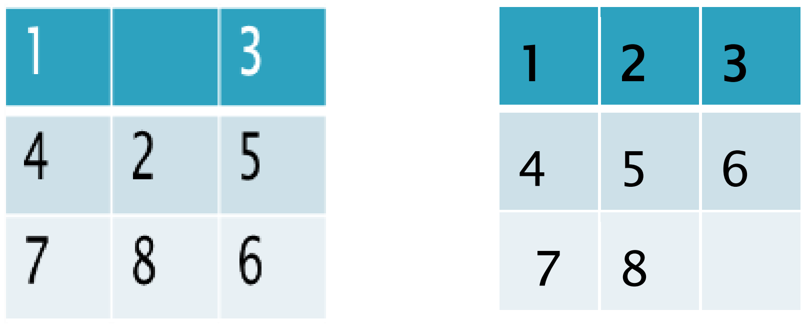
```
D:\project\a star.exe
Initial:
6 3
4 2 1
5 8 7

Searching ...
```

Fig.6 Solvability Check Screenshot

## 4.4 Heuristics

### 4.4.1 Number of Misplaced Tiles

In this heuristic, a tile from any position may be taken out and moved to any required position. The evident algorithm for finding a solution is basically moving each tile from its present spot to the spot in its goal configuration. Thus, the path-length of the least cost-path is the count of tiles that are not present in its desired positions.



| | | |
|---|---|---|
| 1 | | 3 |
| 4 | 2 | 5 |
| 7 | 8 | 6 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Fig.7 Number of misplaced tiles

Here tiles 2,5,6 are not in  correct positions . so h(n) =3

**Drawbacks:** It assumes that a misplaced tiles can simply be removed and placed in its goal positions. But we only can slide the tiles to get to correct position and cannot take the tiles out of the board. It does not consider that.

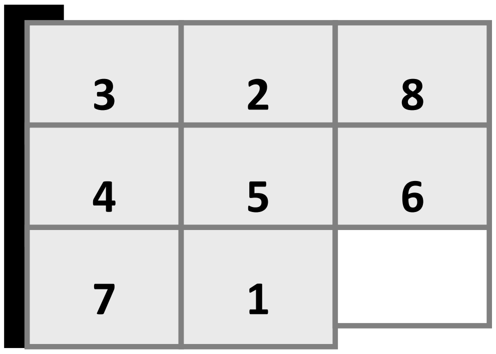
## 4.4.2 Sum of Manhattan Distances

**Manhattan Distance-** It is the linear distance the tile has to cover from initial position to reach the goal position. According to this heuristic, a tile may be moved into any horizontally or vertically adjacent position, with stacking allowed. Obviously, the optimal solution to this puzzle is found by moving each tile along a *shortest path* between its initial and goal state. For anyone tile, the length of this shortest path is the grid distance (horizontal plus vertical distance) between its current and goal positions. Therefore, the total solution length is merely the summation of these grid distances for each tile.

**Example-**

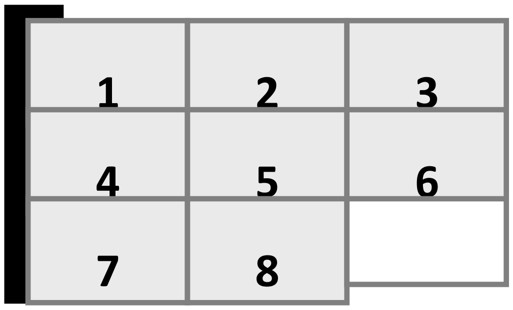
In the figure given below, only the "3"," 8" and "1" numbered tiles are, away from their goal state by 2, 3, and 3 squares respectively. So the heuristic function evaluates to 8(2 + 3 + 3). It means the heuristic signals that the goal state can be reached in just 8 moves.

fig.8 Sum of Manhattan distance



| | | |
|---|---|---|
| 3 | 2 | 8 |
| 4 | 5 | 6 |
| 7 | 1 | |

Initial state



| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Goal state

In the figure below, solution to the 8-puzzle is found using the heuristic sum of Manhattan Distance.

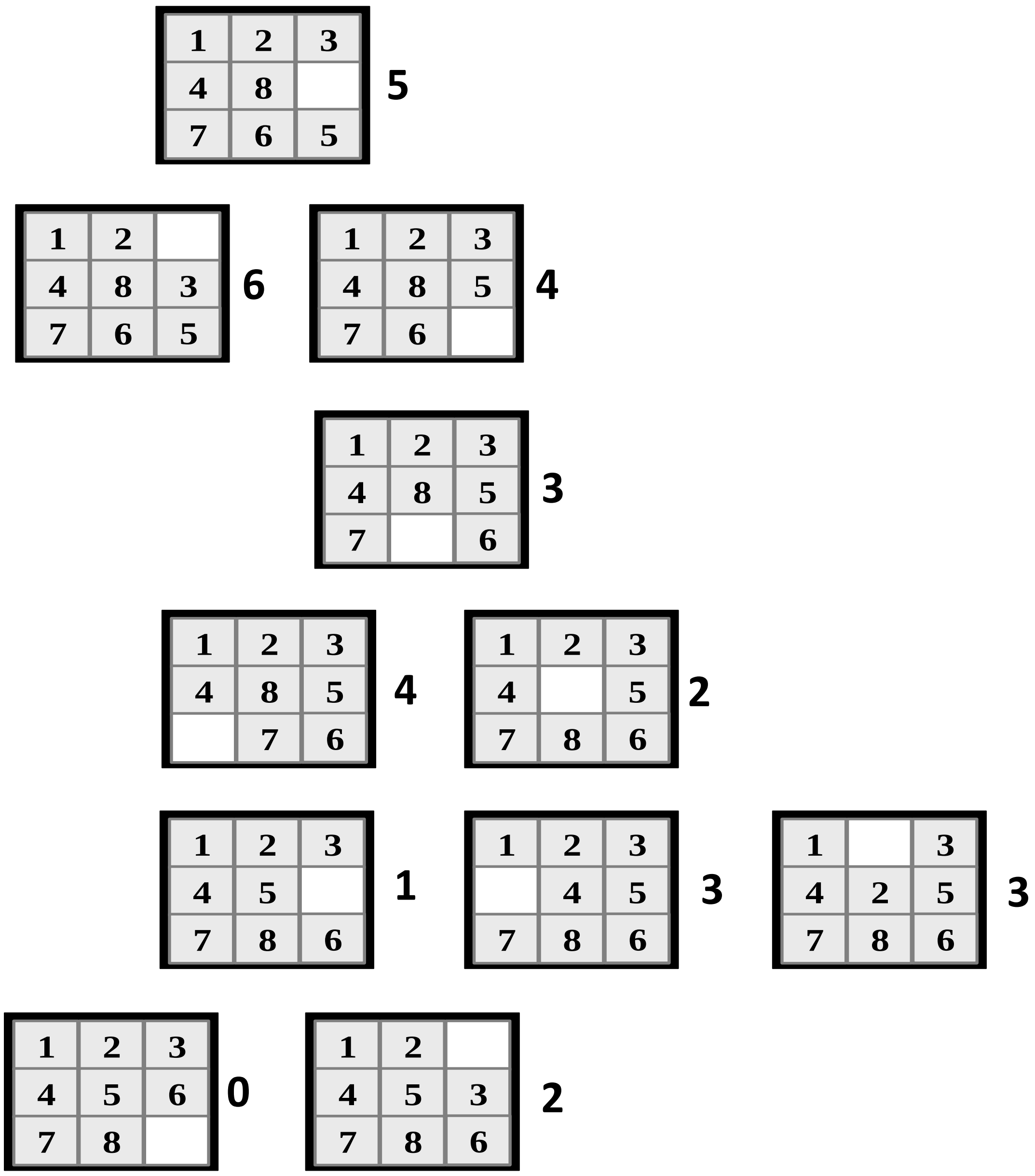


Fig.9 Solving a 8-puzzle using Manhattan distance

## Shortcomings of the Manhattan Distance Model

It states that the puzzle can be solved by displacing each tile along the taxi-cab linear path to its position in goal state. All the more particularly, the shortest path-length solution in the Manhattan Distance heuristic is a set of sub goal solution functions, one for each tile. A sub goal solution is any shortest path for a given tile from its current to its goal position. In many cases, there is a single, unique shortest path: the tile is already in its correct row (column) and need only move within that row (column).

| | | |
|---|---|---|
| | 2 | 1 |
| 3 | 7 | 5 |
| 6 | 4 | 8 |

Let this be the goal state.

| | | |
|---|---|---|
| | | |
| 5 | | 3 |
| | | |

Fig.10 Shortcomings of Manhattan distance

Let this be the initial configuration of 5 and 3.manhattan distance due to 5 and 3 will be 4 but we know that to reach their correct positions either one of the tile has to take 2 extra path lengths along the top or the bottom row.

## 4.5 Algorithm for Calculating the Linear Conflict Heuristic

Definition of **linear conflict**:

2 tiles $t_m$ and $t_n$ are said to be in linear conflict if $t_m$ and $t_n$ stand in the same line (row or column) as the goal positions of $t_m$ and $t_n$ ,$t_m$ is to the left/ right of $t_n$, and the goal position of $t_m$ is to the right/ left of that of $t_n$.

We now define some variables used by the algorithm:

s is the current state.

$Cn(t_m,r_m)$ is the number of tiles in row $r_m$ with which $t_n$ is in conflict(defined above). Similarly we get $Cn(t_m,C_n)$ for columns.

$lcn(s, r_m)$ is the number of tiles that must be removed from row $r_m$ in order to resolve the linear conflicts in that row. Similarly, $lcn(s,c_m)$ is the number of tiles that must be removed from column $C_m$ in order to resolve the linear conflicts.

$m(s, t_i)$ is the Manhattan Distance of tile $t_i$.

L is the number of tiles in a line (row or column) in the puzzle.

$L = \sqrt{(N + 1)}$.

LCN(s) is the least number of moves necessary to resolve the linear conflicts in s.

M(s) is the sum of the Manhattan Distances of all the tiles in s.

**Algorithm:linear_conflict**

For each row $r_i$ in the state s, one accounts for the conflicts local to that row *lcn*(s,ri) as follows:

- *lcn*(s,rj) = 0

For each tile *tj* in *ri* determine *Cn(tj,ri)*

While there is a non-zero *Cn(tj,ri)* value, do

- Find *tk* such that there is no *Cn(tj,ri)* greater than *Cn(tk,ri)*. (As *tk* is the tile with the most conflicts, we choose to move it out of *ri)*.
- *Cn(tk,ri)* = O.
- For every tile *tj* which had been in conflict with *tk*

  *Cn(tj,ri) = Cn(tj,ri)-1* *lcn(s,ri)=lc(s,ri)+ 1.*

Check similarly for linear conflicts in each column Cj computing *lcn*(s, cj), Then calculate the estimate of the Linear Conflict alone:

*LCN*(s) = 2[ {*lcn*(s,r1)+ ... +*lcn*(s,rL)}+{*lcn*(s,c1)+⋯+*lcn*(s,cL)} ]

Determine, for each tile tj in state s, its Manhattan Distance *m*(s,tj), and sum these to get the overall Manhattan Distance

*M*(s) =*m*(s,t1) + ... +*m*(s,tN). Calculate the total Linear Conflict heuristic

estimate. h(s) = *M*(s) + *LCN*(s).

**End {Algorithm}**

**Complexity of removing linear-conflict:**

Calculation of M(s) requires O(N) operations.

Calculation of LCN(s) requires O(N) operations in the worst case, for each line of tiles.
As there are √(N + 1) lines, it needs O (N^1.5) operations.
So computational complexity is O(N^1.5).

## 4.6 Checking for Admissibility

**1. Misplaced tiles**:

It is admissible as anyhow the misplaced tiles are to be shifted to their desired positions.so it can never overestimate the path length.

**2. Sum of Manhattan distances:**

It is admissible as anyhow the tile has to take the shortest taxicab distance (no diagonal move allowed) to reach to its goal position.

**3. Linear Conflict:**

It is also admissible as we have implemented it from the shortcomings of the Manhattan distance and extra path length of 2 has to be covered when linear conflict is there.

## 4.7 Comparing the Performance of the Three Heuristics

Comparison of the memory used by the three heuristics is done as per results of the implemented program:

| Misplaced tiles | | | Manhattan distance | | | Linear conflict | | |
|---|---|---|---|---|---|---|---|---|
| Path length | Open list | Closed list | Path length | Open list | Closed list | Path length | Open list | Closed list |
| 20 | 1719 | 1230 | 20 | 268 | 313 | 20 | 55 | 74 |
| 20 | 1485 | 2515 | 24 | 614 | 968 | 22 | 114 | 184 |
| 28 | 2871 | 4112 | 18 | 411 | 645 | 18 | 49 | 82 |
| 24 | 4521 | 7534 | 22 | 856 | 907 | 14 | 17 | 21 |
| 18 | 1317 | 2259 | 16 | 683 | 774 | 24 | 136 | 186 |
| 16 | 1221 | 1345 | 21 | 765 | 876 | 22 | 77 | 116 |
| 18 | 1986 | 2118 | 29 | 564 | 878 | 19 | 43 | 89 |
| 19 | 4117 | 5609 | 13 | 455 | 643 | 22 | 34 | 47 |
| 21 | 2334 | 2674 | 19 | 761 | 963 | 26 | 55 | 78 |
| 20 | 2856 | 3177 | 19 | 438 | 567 | 17 | 98 | 148 |
| 15 | 1104 | 1407 | 21 | 384 | 529 | 19 | 34 | 69 |
| 23 | 4703 | 4854 | 23 | 738 | 879 | 22 | 54 | 94 |
| 22 | 4194 | 4462 | 19 | 414 | 564 | 20 | 137 | 165 |
| | | | | | | | | |

All of the three heuristics provide optimal solutions as they are admissible. In our observation Average path length of:

1. Misplaced tiles = 264/13 = 20.307

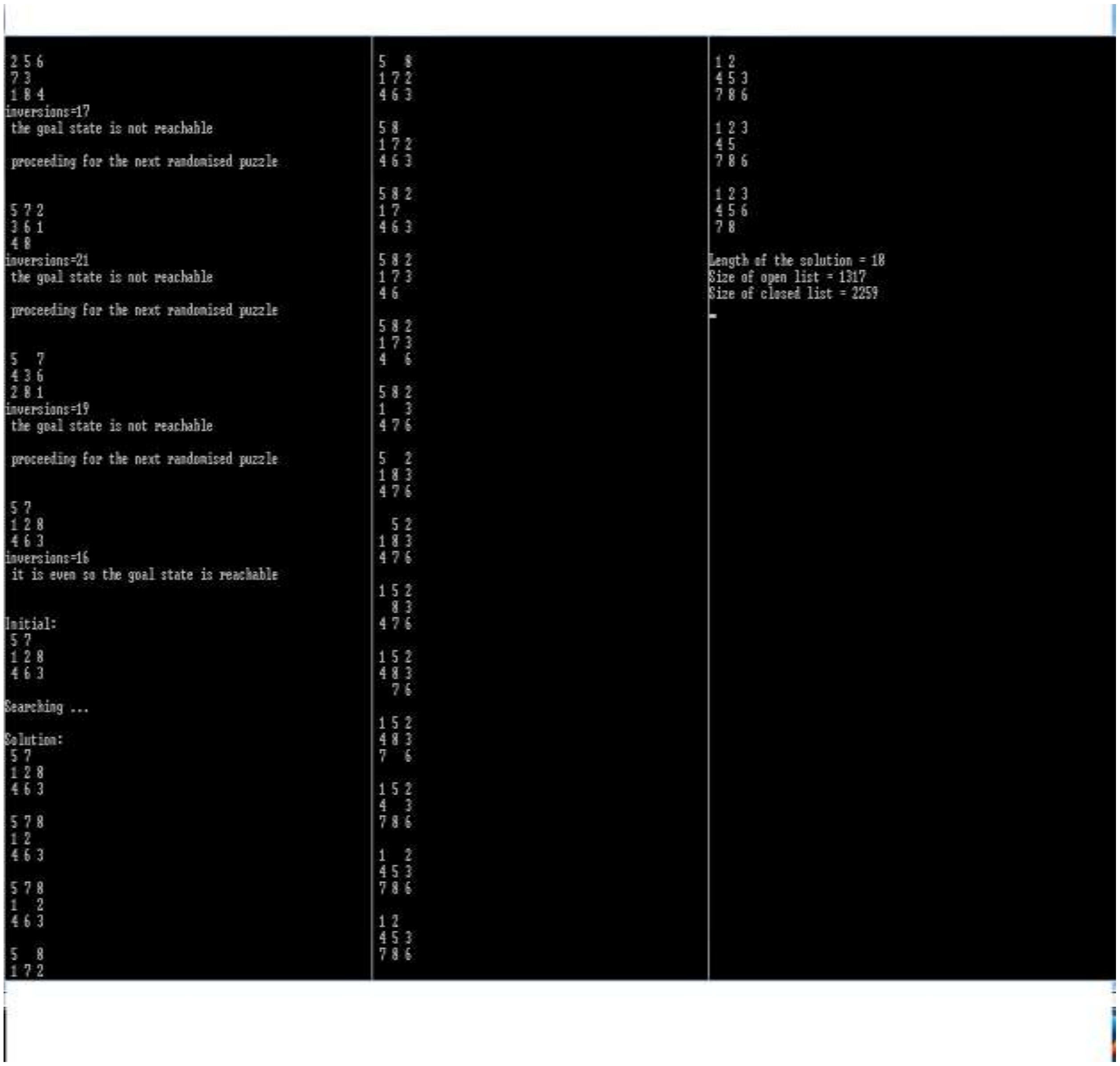
2. Sum of Manhattan distance = 263/13 = 20.23

3. Linear conflict = 285 /13 = 21.9

## 4.8 Screenshots

Screenshots of the output using the various heuristics are shown.

### Missing Tiles Heuristic



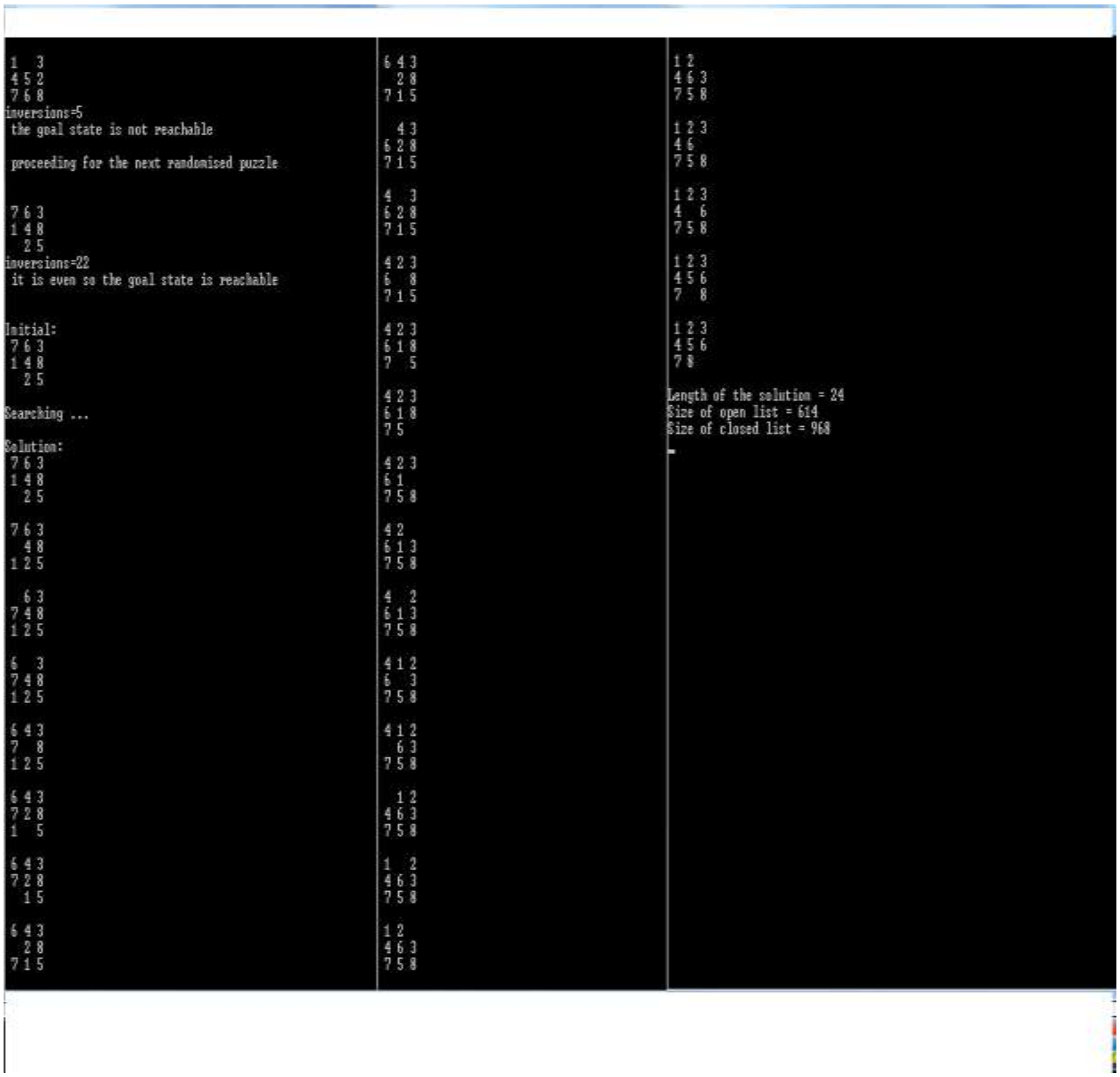
```
 2 5 6
 7 3
 1 8 4
inversions=17
 the goal state is not reachable

 proceeding for the next randomised puzzle


 5 7 2
 3 6 1
 4 8
inversions=21
 the goal state is not reachable

 proceeding for the next randomised puzzle


 5   7
 4 3 6
 2 8 1
inversions=19
 the goal state is not reachable

 proceeding for the next randomised puzzle


 5 7
 1 2 8
 4 6 3
inversions=16
 it is even so the goal state is reachable


Initial:
 5 7
 1 2 8
 4 6 3

Searching ...

Solution:
 5 7
 1 2 8
 4 6 3

 5 7 8
 1 2
 4 6 3

 5 7 8
 1   2
 4 6 3

 5   8
 1 7 2
```

```
 5   8
 1 7 2
 4 6 3

 5 8
 1 7 2
 4 6 3

 5 8 2
 1 7
 4 6 3

 5 8 2
 1 7 3
 4 6

 5 8 2
 1 7 3
 4   6

 5 8 2
 1   3
 4 7 6

 5   2
 1 8 3
 4 7 6

   5 2
 1 8 3
 4 7 6

 1 5 2
   8 3
 4 7 6

 1 5 2
 4 8 3
   7 6

 1 5 2
 4 8 3
 7   6

 1 5 2
 4   3
 7 8 6

 1   2
 4 5 3
 7 8 6

 1 2
 4 5 3
 7 8 6
```

```
 1 2
 4 5 3
 7 8 6

 1 2 3
 4 5
 7 8 6

 1 2 3
 4 5 6
 7 8

Length of the solution = 18
Size of open list = 1317
Size of closed list = 2259

_
```

Fig.11 Implementation of missing tile heuristic

## Sum of Manhattan Distance Heuristic



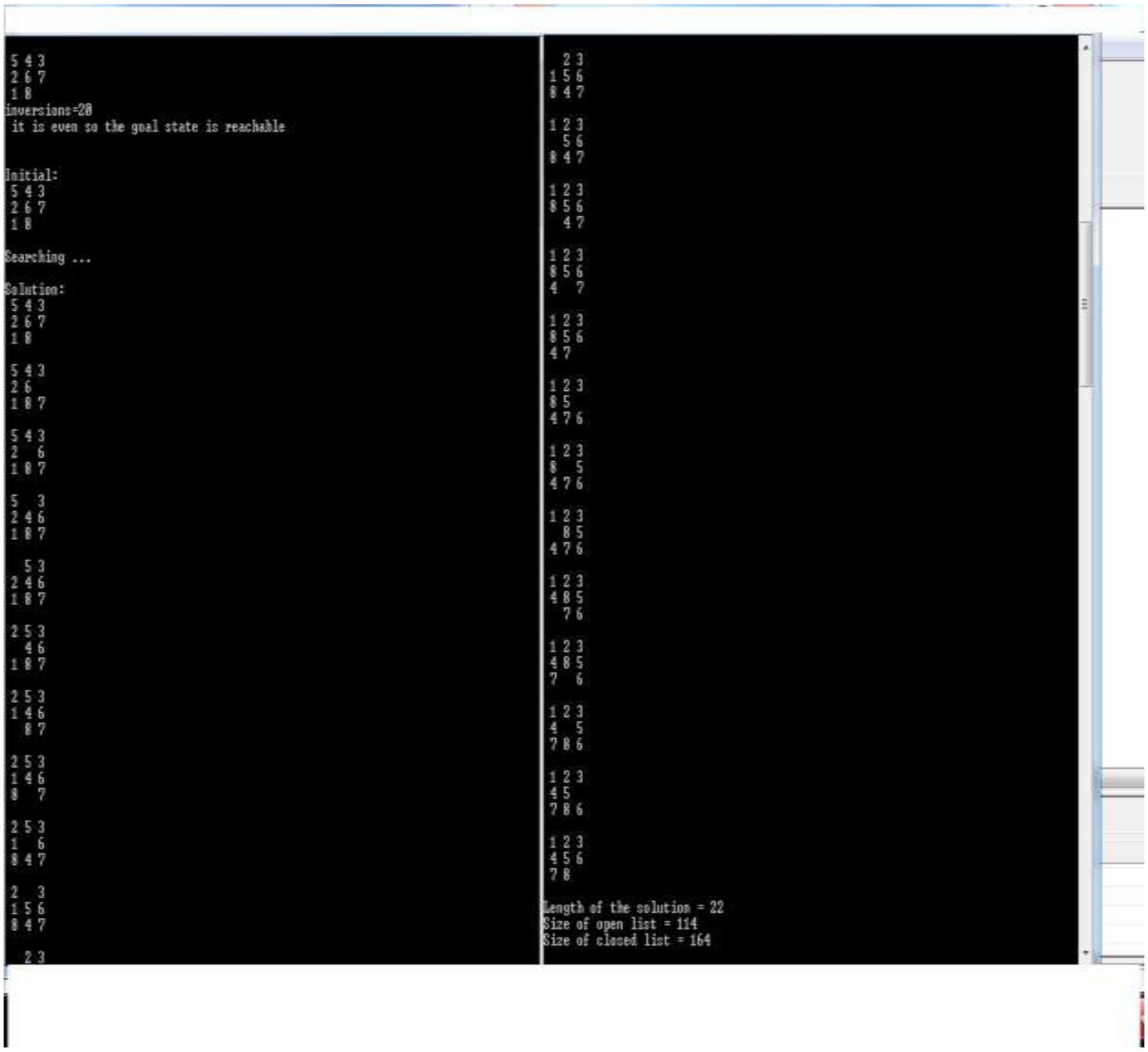
```
 1   3
 4 5 2
 7 6 8
inversions=5
 the goal state is not reachable

 proceeding for the next randomised puzzle


 7 6 3
 1 4 8
   2 5
inversions=22
 it is even so the goal state is reachable


Initial:
 7 6 3
 1 4 8
   2 5

Searching ...

Solution:
 7 6 3
 1 4 8
   2 5

 7 6 3
   4 8
 1 2 5

   6 3
 7 4 8
 1 2 5

 6   3
 7 4 8
 1 2 5

 6 4 3
 7   8
 1 2 5

 6 4 3
 7 2 8
 1   5

 6 4 3
 7 2 8
   1 5

 6 4 3
   2 8
 7 1 5
```

```
6 4 3
  2 8
7 1 5

  4 3
6 2 8
7 1 5

4   3
6 2 8
7 1 5

4 2 3
6   8
7 1 5

4 2 3
6 1 8
7   5

4 2 3
6 1 8
7 5

4 2 3
6 1
7 5 8

4 2
6 1 3
7 5 8

4   2
6 1 3
7 5 8

4 1 2
6   3
7 5 8

4 1 2
  6 3
7 5 8

  1 2
4 6 3
7 5 8

1   2
4 6 3
7 5 8

1 2
4 6 3
7 5 8
```

```
 1 2
 4 6 3
 7 5 8

 1 2 3
 4 6
 7 5 8

 1 2 3
 4   6
 7 5 8

 1 2 3
 4 5 6
 7   8

 1 2 3
 4 5 6
 7 8

Length of the solution = 24
Size of open list = 614
Size of closed list = 968
```

Fig.12 Sum of Manhattan distance heuristic

## Removing Linear Conflict Heuristic



```
 5 4 3
 2 6 7
 1 8
inversions=20
 it is even so the goal state is reachable


Initial:
 5 4 3
 2 6 7
 1 8

Searching ...

Solution:
 5 4 3
 2 6 7
 1 8

 5 4 3
 2 6
 1 8 7

 5 4 3
 2   6
 1 8 7

 5   3
 2 4 6
 1 8 7

   5 3
 2 4 6
 1 8 7

 2 5 3
   4 6
 1 8 7

 2 5 3
 1 4 6
   8 7

 2 5 3
 1 4 6
 8   7

 2 5 3
 1   6
 8 4 7

 2   3
 1 5 6
 8 4 7

   2 3
```

```
   2 3
 1 5 6
 8 4 7

 1 2 3
   5 6
 8 4 7

 1 2 3
 8 5 6
   4 7

 1 2 3
 8 5 6
 4   7

 1 2 3
 8 5 6
 4 7

 1 2 3
 8 5
 4 7 6

 1 2 3
 8   5
 4 7 6

 1 2 3
   8 5
 4 7 6

 1 2 3
 4 8 5
   7 6

 1 2 3
 4 8 5
 7   6

 1 2 3
 4   5
 7 8 6

 1 2 3
 4 5
 7 8 6

 1 2 3
 4 5 6
 7 8

Length of the solution = 22
Size of open list = 114
Size of closed list = 164
```

Fig.13 Removing Linear Conflict Heuristic

## 4.9 Dominating Heuristic

Between two heuristics h1 and h2, h2 dominates h1 if the value of

h2(n) >= h1(n) for any node n.

a* algorithm will expand less number of nodes if implemented using h2 than h1.

A node where f(n) < f*(n) when f* is the optimal path will be expanded. Thus n is expanded whenever

h(n)<f*(n) - g(n)

given h2(n) value greater than h1(n) if a node is expanded by h2 then it will surely be expanded using h1.

Let this be the goal state of the Eight Puzzle((0 1 2 3 45 6 7 8))

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

:

If initial state is

| | | |
|---|---|---|
| | 2 | 1 |
| 7 | 4 | 5 |
| 6 | 3 | 8 |

Misplaced Tiles = 4

Manhattan Distance = 6

Linear Conflict =8

Optimal Solution = 22

(0 2 1 7 4 5 6 3 8)

Another example

| | | |
|---|---|---|
| | 2 | 1 |
| 5 | 4 | 3 |
| 6 | 7 | 8 |

Misplaced Tiles = 4

Relaxed Adjacency = 6

Manhattan Distance = 6

Linear Conflict = 12

Optimal Solution = 20

(0 2 1 5 4 3 6 7 8)

Fig.14 Dominating Heuristics

So we conclude that in dominance

Linear Conflict >Manhattan Distance >Misplaced Tiles

# Chapter 5

## 5.Conclusion

An approach for solving the 8-puzzle problem has been proposed which minimizes the memory required while achieving optimum results. All the heuristics have been applied to a* algorithm to bring uniformity and the results were shown. Comparison of memory used makes sense only in a star algorithm. At last memory use of all the heuristics are compared. Ida* algorithm was implemented but only with one heuristic i.e. number of misplaced tiles.

A solvability check was implemented at the beginning of every program to avoid unnecessary run time errors and crashing of the program.

# Bibliography

[1] Othar Hansson Andrew E. Mayer Mordechai M. Yung "**Generating Admissible Heuristics by Criticizing Solutions to Relaxed Models"** *Information Sciences, 19920915.*

[2] peter e. hart,Nils j. nillson, member IEEE, "**A formal basis for the heuristic determination of Minimum cost paths".**

[3]Alexander reinfeld ."**Complete Solution of Eight-Puzzle and the Benefit of Node Ordering in IDA***"*Paperborn Centre for Parallel Computing Warburger Str. 100. D-33095 Paperborn,Germany*.

[4] A. Reyes[a], H. Yu[b,], G. Kelleher[c], S. Lloyd[d] " **Integrating Petri Nets and hybrid heuristic search for the scheduling of FMS**" *Computers in Industry Volume 47, Issue 1, January 2002* .

[5] Michael Katz and Carmel Domshlak_ Faculty of Industrial Engineering & Management Technion, Israel "**Optimal Additive Composition of Abstraction-based Admissible Heuristics**", *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008).*

[6] Bellmore, M. and Nemhauser, G.L. "**The Traveling Salesman Problem: A Survey**". *Operations Research* 16 (1968), 538-558.

[7] Doran, 1. and Michie, D. "**Experiments with the Graph-Traverser Algorithm**" *Proceedings of the Royal Society, 294 (A), 1966, pp. 235-259.*

[8] Sudip Roy. "**Artificial intelligence approach to test vector reordering for dynamic power reduction during VLSI testing**", *TENCON 2008 IEEE Region 10 Conference, 11/2008*

[9]https://www.cs.princeton.edu/courses/archive/fall12/cos226/assignment/.

[10]https://courses.cs.washington.edu/courses/cse473/12au/slides/lect3.pdf.

[11] http://en.wikipedia.org/wiki/15_puzzle**.**