CS362: Artificial Intelligence Laboratory
Report - 1

by

Group Members
Anirudh Mitra[1]
Chirag Jain[2]
Yash Sakle[3]
&
Student ID
201951024[1]
201951049[2]
201951177[3]

Code Repository: Github.

Submitted To: Prof. Pratik Shah

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY VADODARA

March 2022

Genetic Algorithm Based Approach to Square Jigsaw Puzzle Solving

Anirudh Mitra, *201951024, B.tech (CSE)*

***Abstract*—In square puzzle solving the aim is to re-construct the scrambled image in a non-overlapping way. In this report, a genetic algorithm for solving colorful square jigsaw puzzles is proposed which uses randomized search technique. The algorithm wields discontinuity in color channels along the borders as a metric to objective function.**

I. Introduction

HERE, in this experiment, we are devising an algorithm for solving colored square jigsaw puzzles.

A. Jigsaw Puzzles

The jigsaw puzzle is a popular problem that roots beyond the birth of computers and using them as problem solvers. In jigsaw puzzle a picture printed or carved on cardboard/wood that is cut into small pieces and the pieces have to be fitted together again to form back the picture that was printed. The traditional jigsaw puzzle considers pieces of different shapes and sizes which can fit together in some particular order and orientation only. However, in this experiment the pieces are square and the heights and widths of each piece is assumed to be equal.

B. Solving Puzzle

A major step towards solving picture based puzzles is to analyze the similarity or dissimilarity between the pieces in order to conceive the relative positions of those pieces.

C. Genetic Algorithm

Genetic Algorithm is a simulation of natural selection based on biological evolution. It becomes a great choice for large search space problems since examining all possible solutions to such problem can easily become non-viable. Genetic algorithm can provide with a way to search in the solution space of the problem given. The agent repeatedly modifies a population of individual solutions in search for an efficient one.

D. The Fitness Function

A Fitness Function in Genetic Algorithm is a special function used to summarise and quantify how fit a solution is. Fitness function is responsible for driving the selection simulation in right direction.
For this jigsaw solving problem, fitness function uses the discontinuity in RGB channel around borders of puzzle pieces.

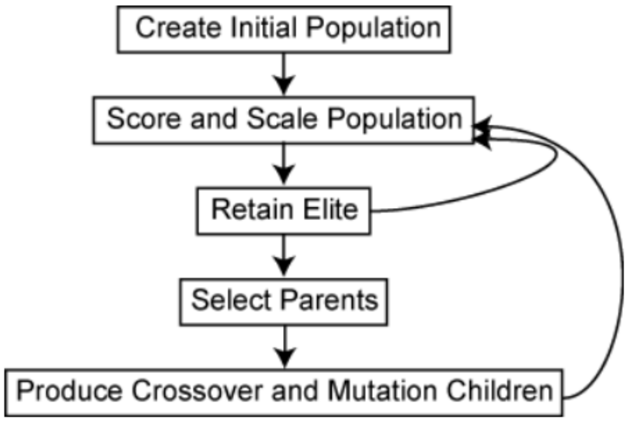

Create Initial Population
Score and Scale Population
Retain Elite
Select Parents
Produce Crossover and Mutation Children

Fig. 1: How Genetic Algorithm Works
Source: Mathworks

Algorithm 1 Genetic Algorithm

```
Initialise an initial population of candidate solutions p[1..n]
repeat
    Calculate the fitness value of each member in p[1...n]
    selected[1..n] ← the new population obtained by picking
    n members from p[1..n] into two halves to generate
    offspring[1..n] with a low probability mutate of p[1..n]
    with the k strongest members of offspring[1..n]
until some termination criteria
return the best member of p[1..n]
```

II. Approach & Algorithm

A. Approach to Jigsaw Solving using GA

For this proposed square jigsaw puzzle solving algorithm, colored images are considered. Given a scrambled image, the agent will proceed in the following direction:

1) Rows & Columns: The agent first tries to find the number of columns, subsequently, finding the number of rows. After these initial operations, the agent will have number of pieces to work with.

2) Height & Width: In the process of finding number of pieces, the agent will also try to determine the dimensions of these rows and columns. This part comes under the assumption that the heights and widths of the rows and columns, respectively, are constant(fixed).

3) Slicing the Image: The next step is to cut the image into pieces using the computations done in the earlier steps.

4) Fitness Function: Now comes the use of fitness function. The fitness metric is defined as a pair-wise sum of discontinuity in the color channels. The agent will compute cost matrices for up-down and left-right pairs and then this cost matrices will be used to determine the fitness of pieces. The pseudocode is expressed in **Algorithm 2**

Algorithm 2 Fitness Function

```
$n\_pieces = number\ of\ rows * number\ of\ columns$
Initialise left-right and up-down cost matrices as zero matrix
for p1 in n_pieces do
    for p2 in n_pieces do
        if p1 = p2 then
            left-right border cost of p1 $\leftarrow \infty$
        else
            left-right border cost of p1 & p2 $\leftarrow$ compute left-right costs of pieces p1 & p2
        end if
        if p1 = p2 then
            up-down border cost of p1 $\leftarrow \infty$
        else
            up-down border cost of p1 & p2 $\leftarrow$ compute up-down costs of rotated pieces p1 & p2
        end if
    end for
end for
return left-right & up-down border cost matrices
```

5) Cost Function: The cost function calculates sum of differences in the color channels. There are 2 cost functions: 1. left-right border cost function 2. up-down border cost function

6) Find First Piece: After the cost matrices are formulated, the agent will parse those matrices and try to guess the first piece of puzzle. The idea is to minimize the costs on the left corner. Finding pieces with minimum costs for both left-right and up-down borders and then taking the maximum out of them roughly gives the top left i.e. the first piece.

7) All Costs: Next task for the agent is to parse all the costs and maximize the fitness along all the pairs. This will give the costs for all possible solutions and the genetic logic will choose the best one in the next step.

8) Find Good Order: The agent will start to assemble the pieces while maximizing the fitness metric. Using the first piece, the agent has a good initial position. **Algorithm 3**

9) Re-construct: The final step is to make the image using the order devised in the previous step.

III. Results

Using the results that we acquired through graphs and image reconstruction, we can form our conclusions and connect them to validate each other.

1) Image Shuffling: Original image is shuffled in 3 columns and 3 rows. This information is not given to the agent as in a real world scenario.

2) Finding Rows and Columns: The first step in the process of puzzle solving is to find out the number of rows and columns along with their height and width. The agent measures discontinuity between two consecutive lines of pixels are decomposes the image into n pieces. Fig.4 & Fig.5 show the plots.

Algorithm 3 Good Order Function

```
right border cost of first piece $\leftarrow \infty$
down border cost of first piece $\leftarrow \infty$
// first piece cannot go in right-most or bottom-most position
for level in rows + cols do
    for row in minimum of rows and levels do
        if level = row + cols then
            all_levels $\leftarrow$ NONE
        else
            get parent out index
            if parent out index $\neq$ NONE then
                compute parent in index
            end if
            find best parent
            all_levels $\leftarrow$ best parent
        end if
        in_levels $\leftarrow$ all_levels
    end for
    result_in_levels $\leftarrow$ in_levels
end for
return result_in_levels
```

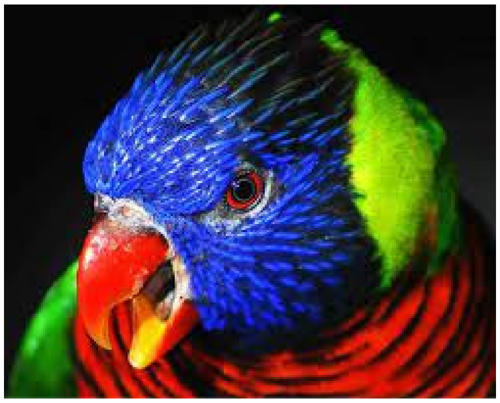


Fig. 2: Original Image
Source: Pinterest

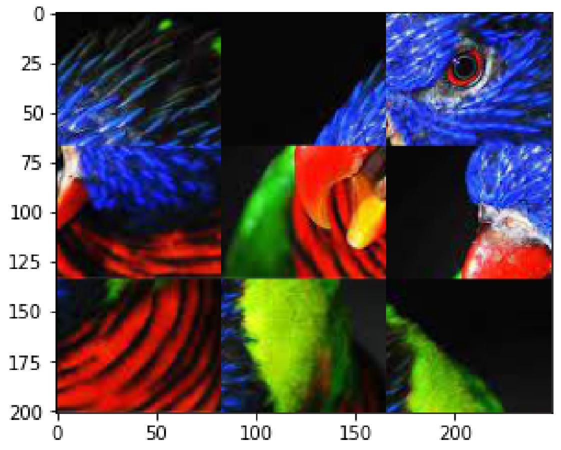
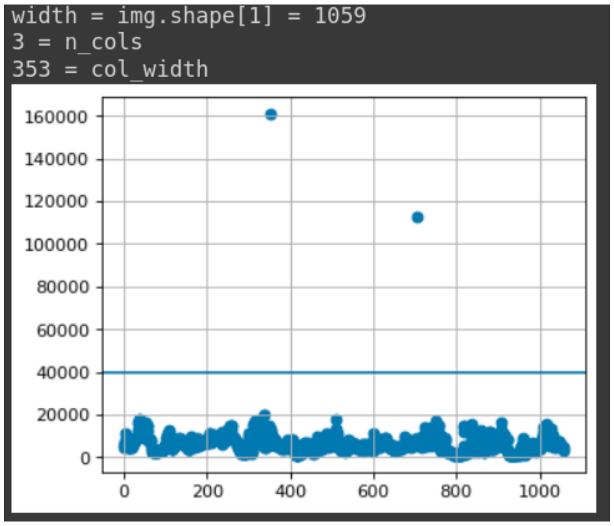
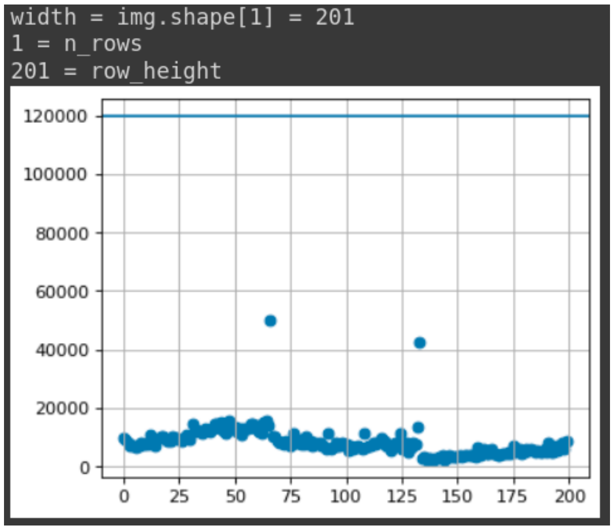


Fig. 3: Shuffled Image



```
width = img.shape[1] = 1059
3 = n_cols
353 = col_width
```

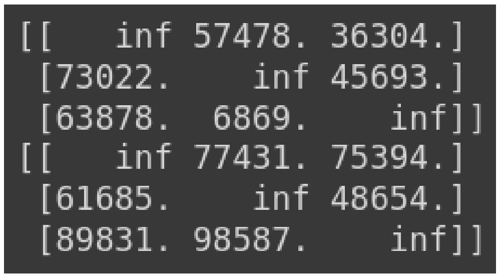
Fig. 4: Discontinuity in columns
Blue line represents the threshold



```
width = img.shape[1] = 201
1 = n_rows
201 = row_height
```

Fig. 5: Discontinuity in rows
Blue line represents the threshold

3) Cost Matrix: Fig.6 shows the cost matrix for given image.



```
[[   inf 57478. 36304.]
 [73022.    inf 45693.]
 [63878.  6869.    inf]]
[[   inf 77431. 75394.]
 [61685.    inf 48654.]
 [89831. 98587.    inf]]
```

Fig. 6: Cost Matrices

4) Order: The agent will find the best order using in and out levels of the parent elements. Fig.7 shows the first piece figured out by the agent.

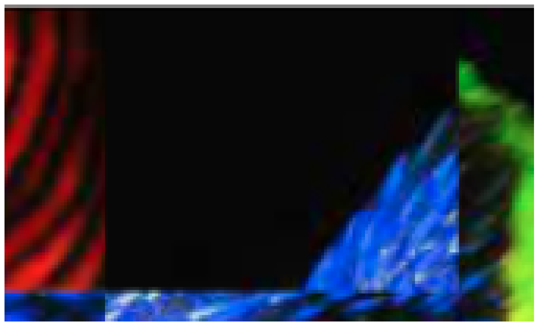
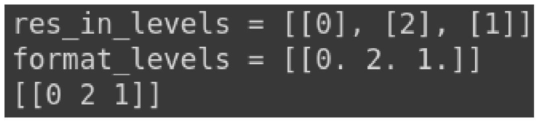


Fig. 7: First Piece



```
res_in_levels = [[0], [2], [1]]
format_levels = [[0. 2. 1.]]
[[0 2 1]]
```

Fig. 8: Order of Pieces Selected by the Agent

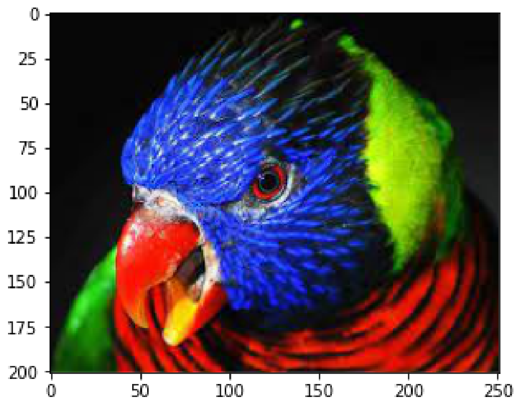


Fig. 9: Reconstructed Image

IV. Conclusion

This algorithm provides a way to solve colorful square jigsaw puzzles. It requires a bit a tuning for different images but can be accurate to a decent level. Future work may include using gray-scale gradients as the fitness metric so that B&W puzzles could also be solved using this algorithm. Feasibility of the code can also be improved by adding learning to it. This will minimize the overall work required. In conclusion, Genetic Algorithms are a good non-deterministic approach to such problems with large search space.

References

[1] “Artificial Intelligence: A Modern Approach, 4th US ed.,” Berkeley.edu, 2021. http://aima.cs.berkeley.edu/
[2] Khemani, Deepak. A First Course in Artificial Intelligence. India: Mc-Graw Hill Education (India), 2013.
[3] N. Alajlan, “AN IMPROVED HEURISTIC-BASED APPROACH FOR SOLVING SQUARE JIGSAW PUZZLES.” https://www.iiis.org/CDs2008/CD2008SCI/SCI2008/PapersPdf/S357XQ.pdf
[4] D. Delahaye, S. Chaimatanan, and M. Mongeau, “Simulated Annealing: From Basics to Applications,” Handbook of Metaheuristics, pp. 1–35, Sep. 2018, doi: 10.1007/978-3-319-91086-4_1.
[5] F. Toyama, Y. Fujiki, K. Shoji and J. Miyamichi, ”Assembly of puzzles using a genetic algorithm,” 2002 International Conference on Pattern Recognition, 2002, pp. 389-392 vol.4, doi: 10.1109/ICPR.2002.1047477.

Game Playing Agent, Minimax, Alpha-Beta Pruning

Chirag Jain, *201951049, B.tech(CSE),*

***Abstract*—The min max algorithm in AI, popularly known as the MINIMAX, is a backtracking algorithm used in decision making, game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, tic-tac-toe, go, and various tow-players game. The MINIMAX algorithm performs a depth-first search algorithm for the exploration of the complete game tree. In the later part we see the Alpha-Beta Purning, it is the optimised technique for MINIMAX algorithm. We will implement MINIMAX and alpha-beta pruning agents for Noughts and Crosses game tree and see how we can optimise the complexity by bounding the range using Alpha-Beta Purning.**

I. INTRODUCTION

WE are given game of Noughts and Crosses and game of Nim. We have to observe the size of the game tree for Noughts and Crosses. Then in the game of Nim we are given with the three pile, we have to build the game playing agent in such a way that regardless of the strategy of player-1 the player-2 must always win the game. We will see the technique used in MINIMAX for the game of Noughts and Crosses and see how it can be optimised using the Alpha-Beta Pruning.

A MINIMAX algorithm is a recursive program written to find the best game-play that minimizes any tendency to lose a game while maximizing any opportunity to win the game. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit as MAX will select the maximized value and MIN will select the minimized value. It use depth first search for exploration of game tree. Since the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half by using alpha-beta pruning. In this we use range bounds(alpha and beta) to optimise the search in trees.

II. THEORY

Minimax is a backtracking algorithm used in decision-making and game theory to determine the best move for a player, provided that their opponent is likewise playing optimally.The two participants in Minimax are known as the maximizer and minimizer. The maximizer looks for the best possible score, whereas the minimizer looks for the lowest possible score. Every state has value attached to it. If the maximizer has the upper hand in a particular state, the score will tend to be positive. In that board state, if the minimizer has the upper hand, it will tend to be a negative value. The values are determined by a set of heuristics that are unique to each game.

A modified version of the minimax method is alpha-beta pruning. It's a way for improving the minimax algorithm. Without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning.The Alpha-beta pruning to a standard minimax algorithm produces the same result as the standard algorithm, but it removes all nodes that aren't really affecting the final decision but are slowing down the algorithm. As a result, pruning these nodes speeds up the algorithm.

III. PROBLEM STATEMENTS

A. What is the size of the game tree for Noughts and Crosses? Sketch the game tree.

For the first one we have nine different states. For second level we have 8 states are available after first level occupied a state out of nine so total of 9*8. For the third one we got 7 states available after 9 and 8 in first and second state respectively, so total 9*8*7. further we can go till last level by following the same pattern.

So the total no of states possible is summation of states of all the levels..

$9 + 9 * 8 + 9 * 8 * 7 + 9 * 8 * 7 * 6 ... 9! \approx 10^6$

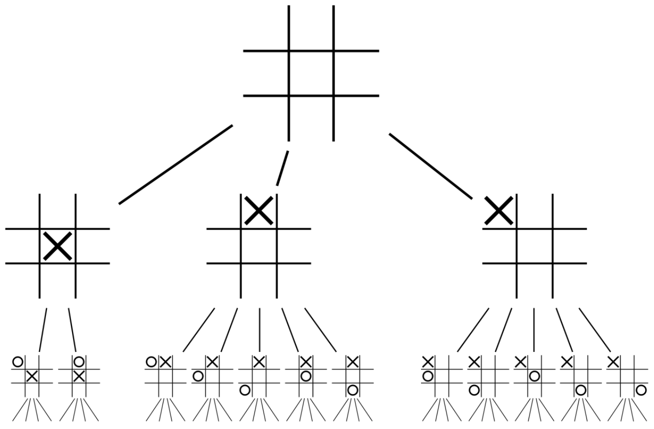


Fig. 1: Tic-Tac-Toe game tree

B. Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.

Nim is a mathematical game of strategy in which two players take turns removing (or "nimming") objects from distinct heaps or piles. On each turn, a player must remove at least one object, and may remove any number of objects

Fig. 2: Nim Piles

provided they all come from the same heap or pile. The goal of the game is either to avoid taking the last object or to take the last object.

Both the players player-1 and player-2 will make a optimal move to win the game.

If player 1 is able to make a move in such a way that XOR of all the three piles is equal to zero then player 1 will always win regardless of moves of player 2.

C. Implement MINIMAX and alpha-beta pruning agents. Report on number of evaluated nodes for Noughts and Crosses game tree.

After running the code we can observe that the evaluated states in the minimax and alpha-beta pruning are 549946 and 18297 respectively.

D. Use recurrence to show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is O($b^{m/2}$), where b is the effective branching factor and m is the depth of the tree..

In the best case the best move is left most alternative as we don't explore further nodes in most optimal solution.

We are given b as branching factor and m as depth.

T(m) be the minimum number of states to be traversed to find the exact value of the current state, and

K(m) be the minimum number of states to be traversed to find the bound on the current state. we determine the recursive relation for the alpha- beta pruning as :

$$T(m) = T(m-1) + (b-1)K(m-1)$$

here T(m-1) is to traverse to child node and find the exact value, and (b-1)K(m-1) is to find min/max bound for the current depth of the tree.

In best case, we know that

$$T(0) = K(0) = 1$$

we can also say that to the exact value of one child is

$$K(m) = T(m-1)$$

Now we expand the recursive relation and solve it further we get :

$$T(m) = T(m-1) + (b-1)K(m-1)$$

$$T(m-1) = T(m-2) + (b-1)K(m-2)$$

$$T(m-2) = T(m-3) + (b-1)K(m-3)$$

and so on till

$$T(1) = T(0) + (b-1)K(0) = 1 + b - 1 = b$$

after solving we can get

$$T(m) = T(m-2) + (b-1)K(m-2) + (b-1)K(m-1)$$

$T(m) = T(m-3) + (b-1)K(m-3) + (b-1)K(m-2) +$

(b-1)K(m-1) using above equation,

$T(m) = T(m-2) + (b-1)T(m-3) + (b-1)T(m-2) =$

b(T(m-2)) + (b-1)T(m-3)

we can clearly say that

$$T(m-2) > T(m-3)$$

therefore,

$$T(m) < (2b-1)T(m-2)$$

since b can be really large we can say b-1 $\approx b, so$

$$T(m) < 2bT(m-2)$$

That is, the branching factor every two levels is less than 2b, which means the effective branching factor is less than $\sqrt{2b}$.

$$T(m) <= \sqrt{b}^{m}$$

which is same as b$^{m/2}$.

IV. Conclusion

In this paper we can see that the minimax is backtracking algorithm used in optimal decision making for the different games. But in minimax the number of states it examine are exponential in depth of tree, so it can't be used in case of a large tree. For such cases we use alpha-beta pruning which is optimal version of minimax where we range bound the result and cut off the states which are out of the range bounded by pruning. In this experiment we can see that the number of states evaluated in minimax is 549946 while in pruning it's just 18297 which is very less than minimax, so alpha-beta pruning is very optimised version on minimax.

References

[1] "Artificial Intelligence — Mini-Max Algorithm - Javatpoint," www.javatpoint.com, 2021. https://www.javatpoint.com/mini-max-algorithm-in-ai (accessed Mar. 18, 2022).

[2] "Best-Case Analysis of Alpha-Beta Pruning." Accessed: Mar. 18, 2022. [Online]. Available: http://www.cs.utsa.edu/ bylander/cs5233/a-b-analysis.pdf.

[3] "Minimax Algorithm in Game Theory — Set 3 (Tic-Tac-Toe AI - Finding optimal move) - GeeksforGeeks," GeeksforGeeks, Jun. 30, 2016. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/ (accessed Mar. 18, 2022).

Graph Search Agent for 8-Puzzle

Yash Sakle, *201951177, B.tech(CSE)*

***Abstract*—The goal of this lab is to perform to design a graph search agent and understand the use of a hash table, queue in state space search. In this lab, we need prior knowledge of the types of agents involved and use this knowledge to solve a puzzle called 8- puzzle. .**

Code Link,

I. Introduction

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square.

The objective is to move the tiles around into different positions to reach the configuration shown in ”Goal State” in Fig 1.

A stack is used to provide temporary storage space for values. It is defined as a data structure which operates on a first in, last out basis. Its uses a single pointer (index) to keep track of the information in the stack[3].

How algorithm works:
Considering that, after we visit the start node, we will continue to visit all of its neighbors and then we will proceed to the next “layer” and will need a queue. When we visit a node, we check its neighbors, and if the neighbors are not visited already, we add them to the queue. Then, we practically just pick the next node to visit from the queue. Another thing that we need to do is mark the visited nodes for not visiting them again. This way, we will visit all of the graph nodes only once.

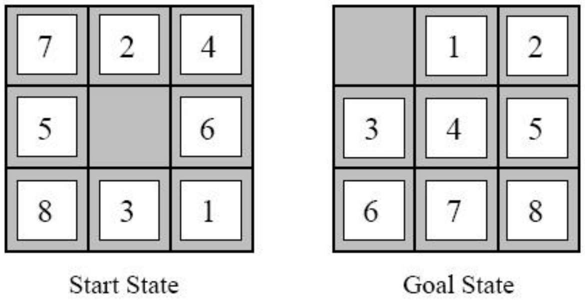


Fig. 1: Start State & Goal State

A. Iterative deepening depth first search

Iterative deepening depth first search (IDDFS) is a hybrid of BFS and DFS. In IDDFS, we perform DFS up to a certain “limited depth,” and keep increasing this “limited depth” after every iteration. Basically it performs DFS at every depth thus reducing the space complexity compared to BFS.

Let’s suppose b is the branching factor and depth is d then
Time Complexity:

$$O(b^d) \tag{1}$$

Space Complexity:

$$O(bd) \tag{2}$$

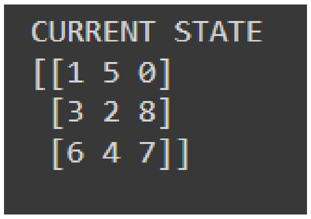
II. Approach

Algorithm 1 8 Puzzle

```
if $pq.empty()$ then return false
    $puz \leftarrow pq.extract()$    ▷ all possible successors to puz
end if
if $search(pq)$ then return true
end if
for $each suc in successors$ do
    if $suc not in visited$ then
        $pq.empty(suc)$
        $visited.insert(suc)$
    end if
end for
if $search(pq.visited)$ then return true
end if
return false
```

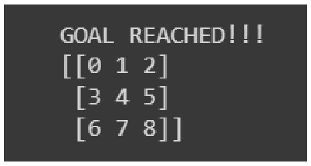
III. Experiment Result

There are some outputs :-



```
CURRENT STATE
[[1 5 0]
 [3 2 8]
 [6 4 7]]
```

Fig. 2: Current State



```
GOAL REACHED!!!
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Fig. 3: Goal State

IV. Conclusion

Solves the puzzle by finding a path from current state to the goal state, using the heuristic provided. If no heuristic is provided, solves using normal BFS. Prints ”GOAL REACHED!!!” if goal is reached and prints ”STRUCK!!!” if no possible move is left.

References

[1] “Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS) - GeeksforGeeks,” GeeksforGeeks, May 19, 2016. https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/ (accessed Mar. 18, 2022).

[2] dpthegrey, “8 puzzle problem - dpthegrey - Medium,” Medium, Jun. 30, 2020. https://medium.com/@dpthegrey/8-puzzle-problem-2ec7d832b6db (accessed Mar. 18, 2022).

[3] “Stacks, Queues, Lists and Hash Tables,” Uniovi.es, 2022. https://www6.uniovi.es/datas/data5.htm (accessed Mar. 18, 2022).

An Algorithm Based on Simulated Annealing for Solving Travelling Salesman Problem

Anirudh Mitra, *201951024, B.tech(CSE),* Chirag Jain, *201951049, B.tech(CSE),*
Yash Sakle, *201951177, B.tech(CSE),*

***Abstract*—For problems with large search spaces, randomized search becomes a meaningful option given partial/ full-information about the domain. The information about the distances between the cities could be used to find an optimal path. In this paper, we're proposing one such algorithm for solving Travelling Salesman Problem using an approach revolving around Simulated Annealing and Non-deterministic state search.**

I. Introduction

Given a graph in which the nodes are locations of cities, and edges are labelled with the cost of travelling between cities. The optimal path to cover all the nodes and return to the starting node using simulated annealing to reduce the cost.

A. Non-Deterministic Algorithms

A non-deterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. They are used to find the approximate solutions where the actual solution is difficult using a deterministic function. Like in travelling salesman problem it will take different paths with same input.There are usually two phases and output steps in a non-deterministic algorithm. The first phase is the guessing phase, in which the problem is solved using random characters. The verifying phase, which returns true or false for the chosen string, is the second phase.

B. Simulated Annealing

Simulated Annealing is a probabilistic global search optimization algorithm. The algorithm is inspired by annealing in metallurgy where metal is heated to a high temperature quickly, then cooled slowly, which increases its strength and makes it easier to work with.Simulated annealing executes in the way that the neighboring point generates better result than the current one, then it is saved as base solution for the next iteration.

C. Travelling Salesman Problem (TSP)

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. TSP is a popular NP-hard problem. using brute-force approach and evaluating every node(city) i.e. $(n-1)!$ nodes need to be evaluated. This number gets pretty huge even with a relatively mid-sized input. For n cites problem, the distance matrix will store pairwise distances between the cities $D = (d_{i,j})_{n\times n}$. For set of permutations π the goal is to find $(\pi(1), \pi(2), .., \pi(n))$ that will minimize,

$$f(x) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

II. Results

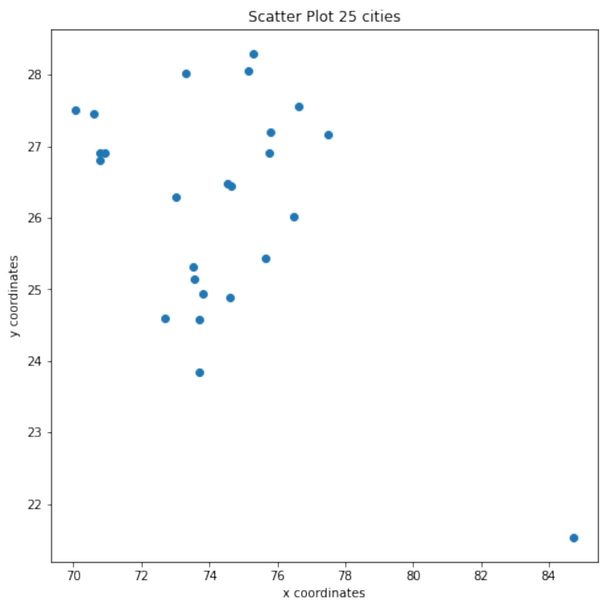


Fig. 1: Random 25 nodes

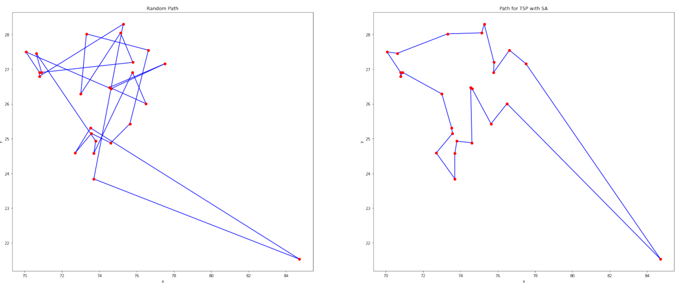


Fig. 2: Random and Simulated path of 25 nodes

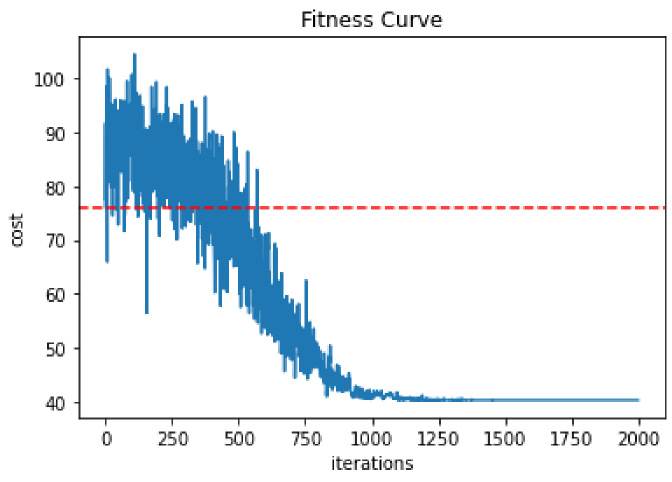


Fig. 3: Fitness Curve of 25 nodes

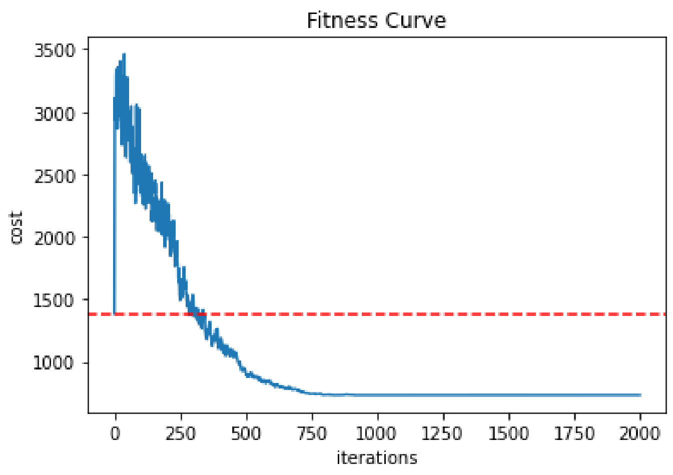


Fig. 6: Fitness Curve of 131 nodes

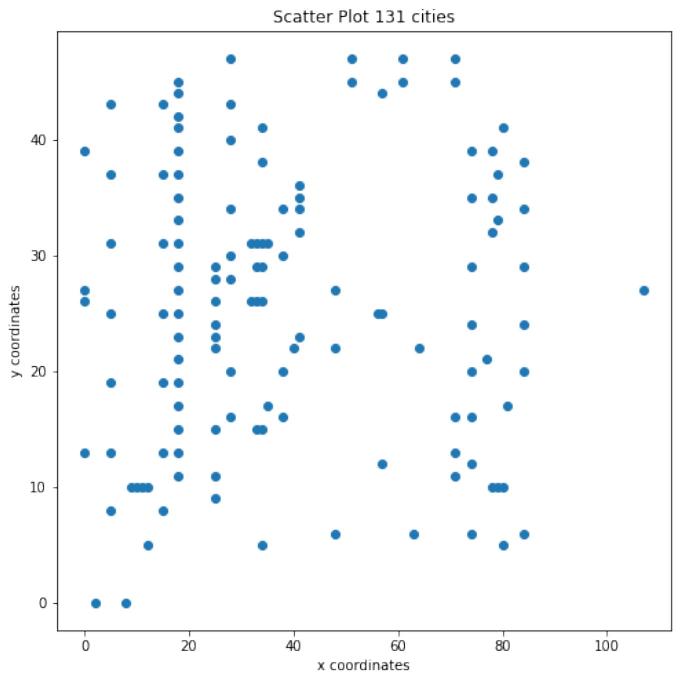


Fig. 4: Random 131 nodes

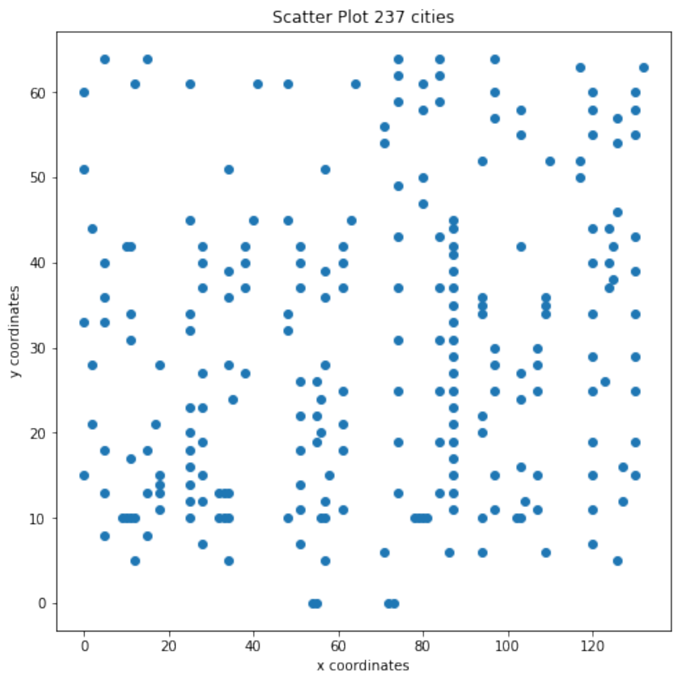


Fig. 7: Random 237 nodes

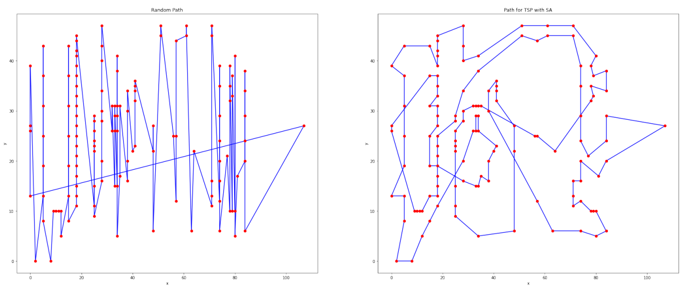


Fig. 5: Random and Simulated path of 131 nodes

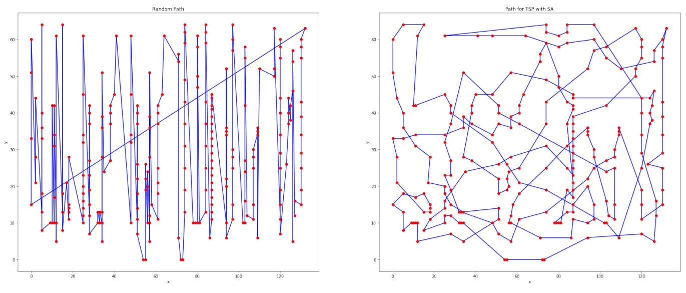


Fig. 8: Random and Simulated path of 237 nodes

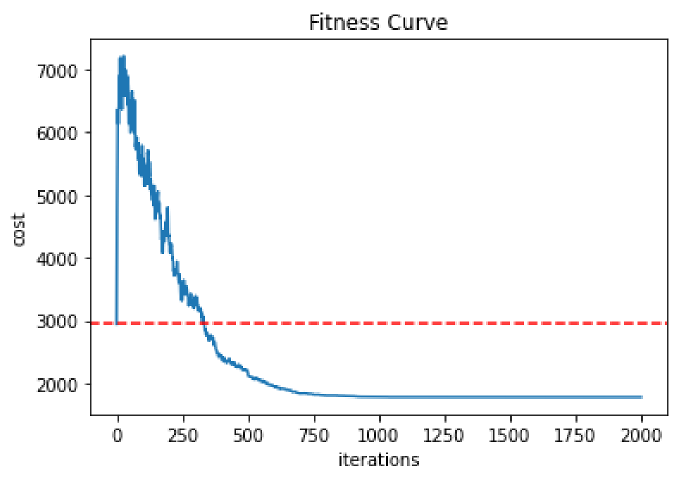


Fig. 9: Fitness Curve of 237 nodes

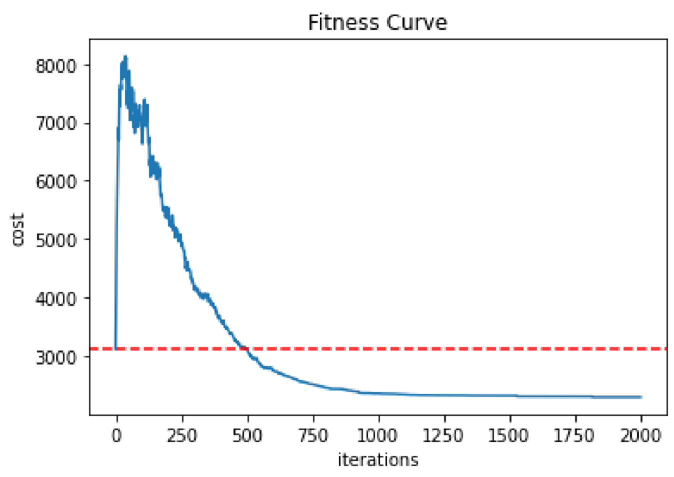


Fig. 12: Fitness Curve of 343 nodes

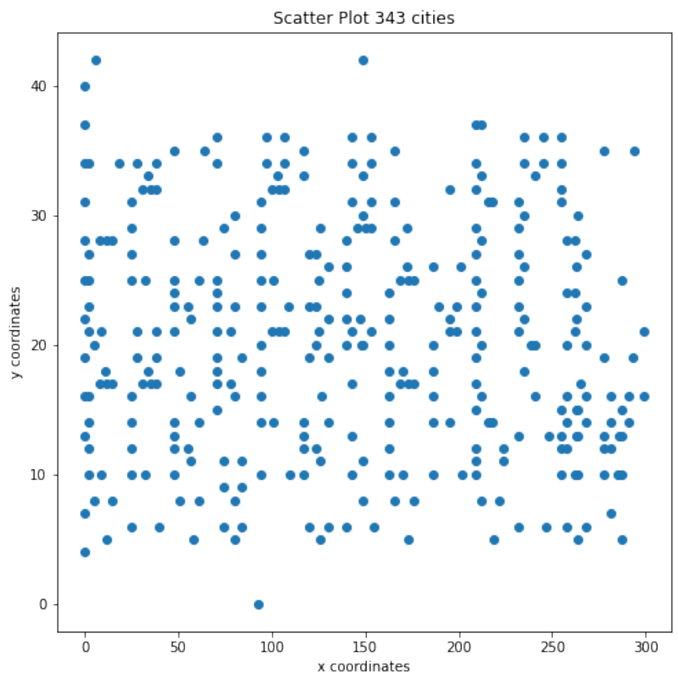


Fig. 10: Random 343 nodes

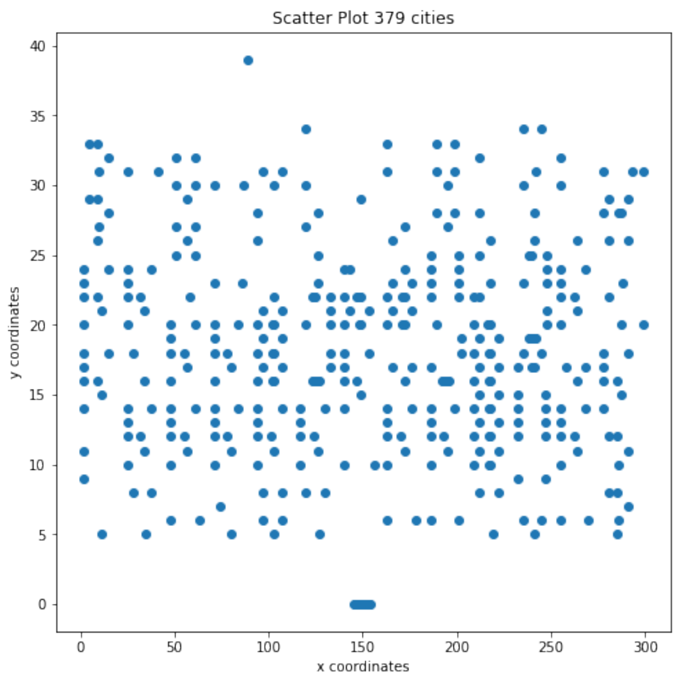


Fig. 13: Random 379 nodes

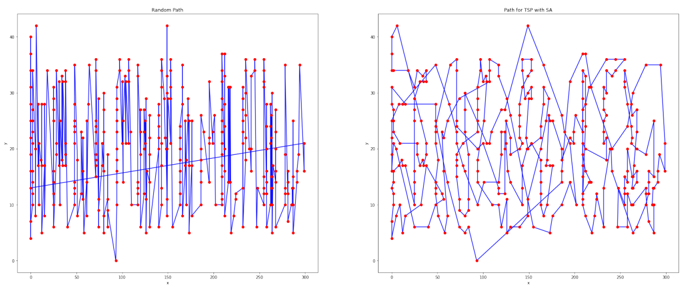


Fig. 11: Random and Simulated path of 343 nodes

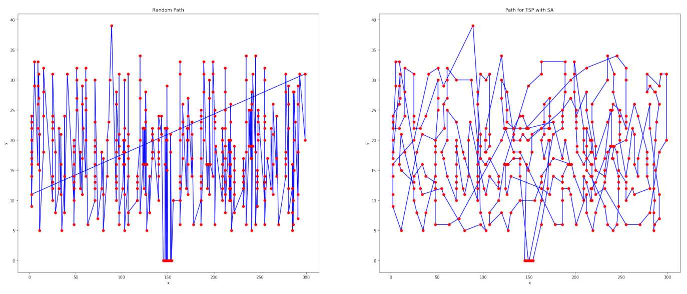


Fig. 14: Random and Simulated path of 379 nodes

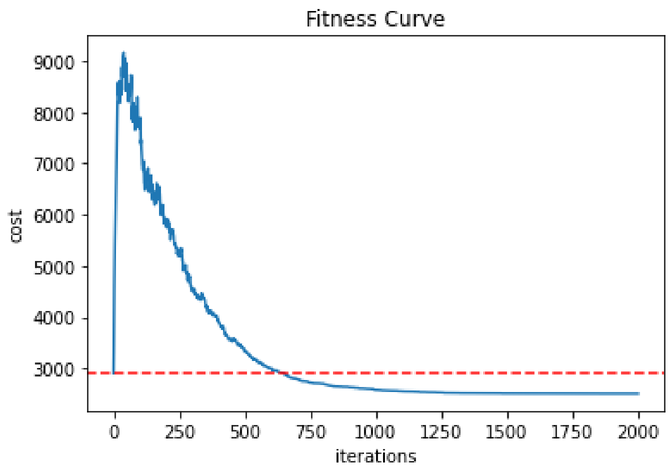


Fig. 15: Fitness Curve of 379 nodes

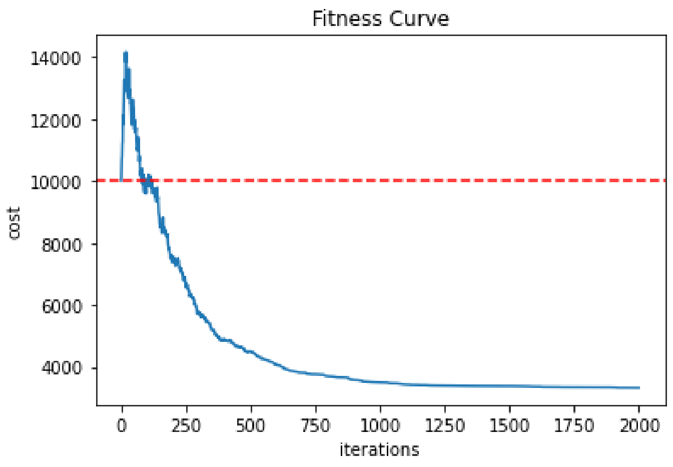


Fig. 18: Fitness curve of 380 nodes

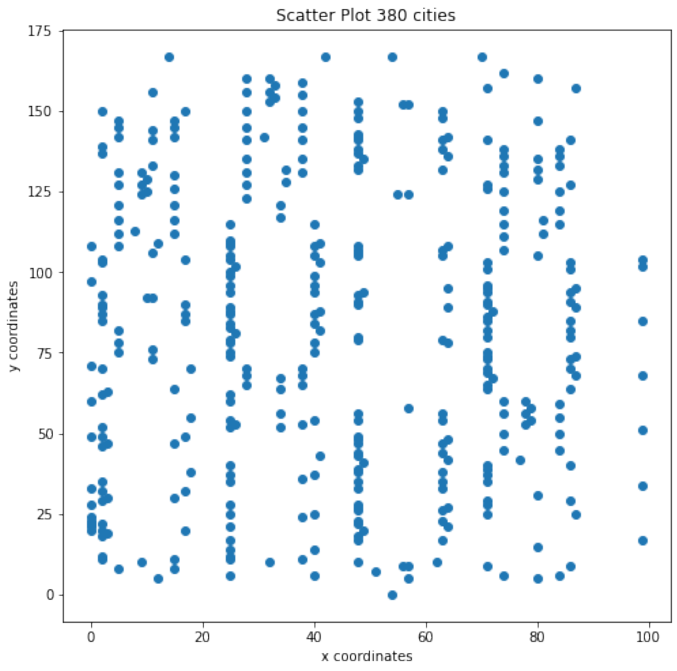


Fig. 16: Random 380 nodes

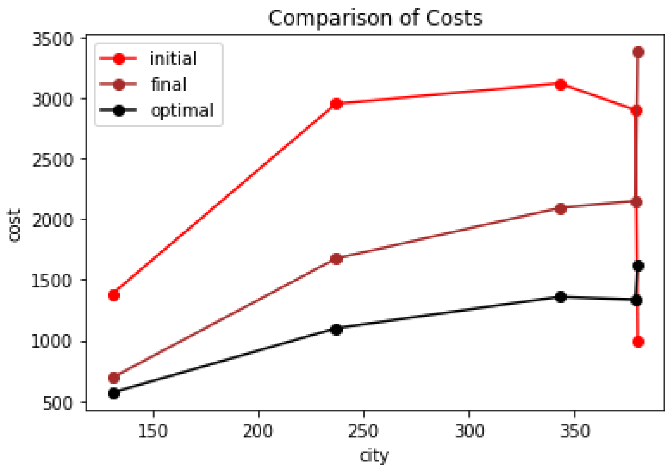


Fig. 19: Comparison of initial, final and optimal cost

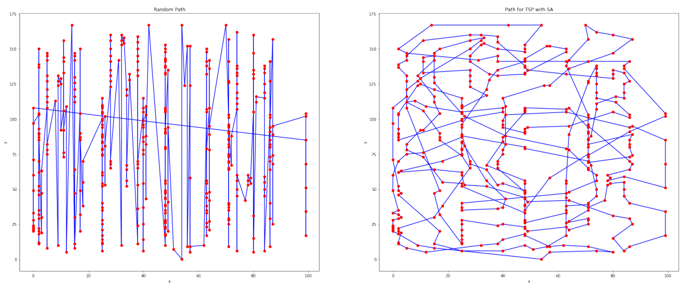


Fig. 17: Random and Simulated path of 380 nodes

III. Conclusion

We can see that the final cost or the cost for route calculated using simulated annealing is smaller than the cost for random route. When the comparing the cost with the optimal cost from VLSI logs of computation for each data set, we can see that our calculated final cost is comparable with the optimal cost for data points like 131 and 237. We can further decrease the cost by increasing the number of iterations and using heuristic functions to solve the traveling salesman problem

References

[1] "Artificial Intelligence: A Modern Approach, 4th US ed.," Berkeley.edu, 2021. http://aima.cs.berkeley.edu/
[2] Khemani, Deepak. A First Course in Artificial Intelligence. India: McGraw Hill Education (India), 2013.