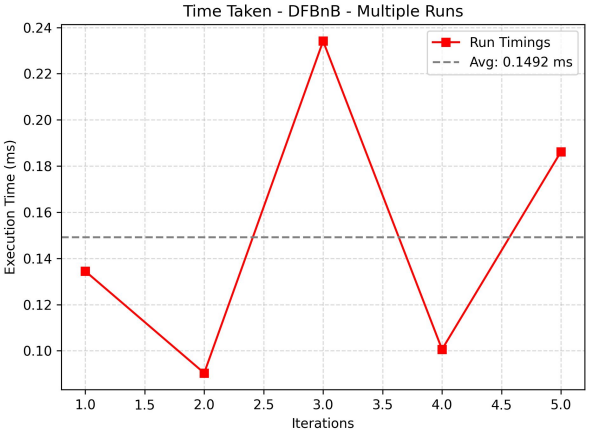
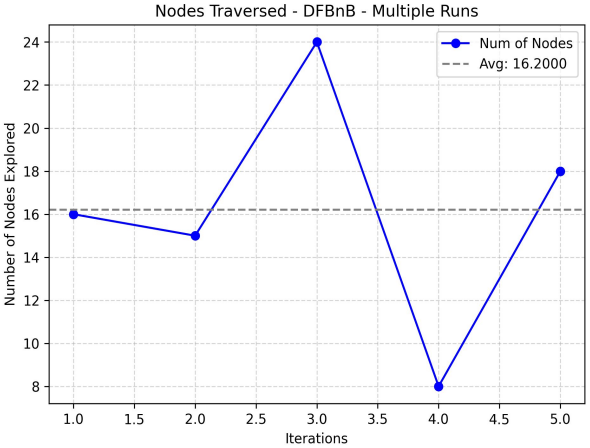
AI Assignment 2

Yashraj Motwani - CS24M104
M Yashwanth Kumar - CS24M122

DFBnB





DFBnB (Depth-First Branch and Bound) is a variant of depth-first search that prunes suboptimal paths using a bound (threshold).

We have implemented the informed version of DFBnB (Depth-First Branch and Bound) functions by using the **Manhattan distance** as a heuristic.
This heuristic estimates the remaining cost to reach the goal by calculating the number of horizontal and vertical steps needed. With this heuristic, DFBnB prunes suboptimal paths more effectively, exploring only those that are likely to lead to an optimal solution.

In each iteration, we have changed the position of the goal state.

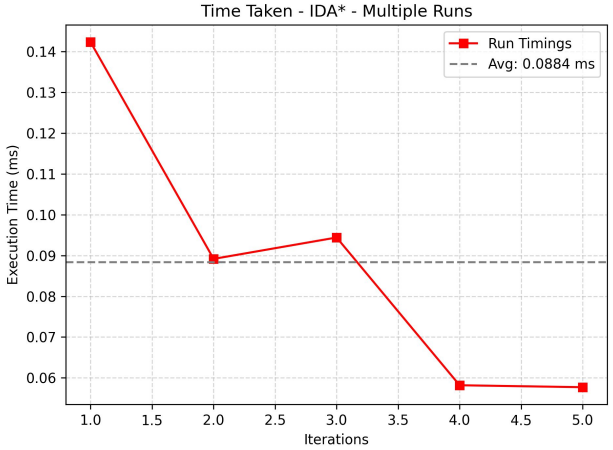
The 1st graph shows the **execution time** for each of the 5 independent runs of the DFBnB algorithm.

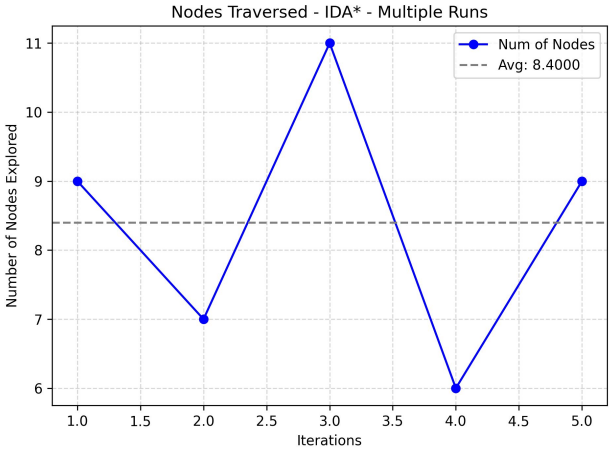
The 2nd graph shows the total number of nodes covered by the algorithm in each iteration to go to the goal state.
These give an insight into how much of the state space DFBnB explores before finding the optimal path. The DFBnB algo is Optimal and Complete when the heuristic is Admissible.

Algorithm 1 DFBnB Algorithm Informed Search Version

```
1:  Input: start state s, goal state g, U = c (greater than δ*(s, g))
2:  Output: Goal State (if present)
3:  function DFBnB(x)
4:      if IsGoal(x) == True then
5:          U = (d[x] < U)?d[x] : U
6:          Return
7:      end if
8:      W ← Expand(x)
9:      for each y ∈ W do
10:         if d[x] + w(x, y) + h(y) ≤ U then
11:             if d[x] + w(x, y) < d[y] then
12:                 d[y] ← d[x] + w(x, y)
13:                 Call DFBnB(y)
14:             end if
15:         end if
16:     end for
17: end function
```

IDA*





Iterative Deepening A* (IDA*) is a graph search algorithm that combines the space-efficiency of depth-first search with the optimality and heuristic guidance of the A* algorithm.

Instead of maintaining a full open list like A*, IDA* performs repeated depth-limited searches, where the limit is determined by the minimum f(n) value that exceeds the current threshold.
f(n) = d(n) + h(n)
Here, d(n) = cost to reach node n, and, h(n) is the estimate of the cost from n to the goal state.
In our implementation, we use the **Manhattan distance** as the heuristic function. This enables IDA* to prune large portions of the search space while guaranteeing optimality. As a result, the algorithm explores significantly fewer nodes and consumes less memory compared to traditional A*, making it suitable for memory-constrained or real-time applications.

The 1st graph shows the **execution time** for each of the 5 independent runs of the IDA* algorithm.

The 2nd graph shows the total number of nodes covered by the algorithm in each iteration to go to the goal state. In each iteration, we have changed the position of the goal state but they match the respective iteration in DFBnB algorithm.

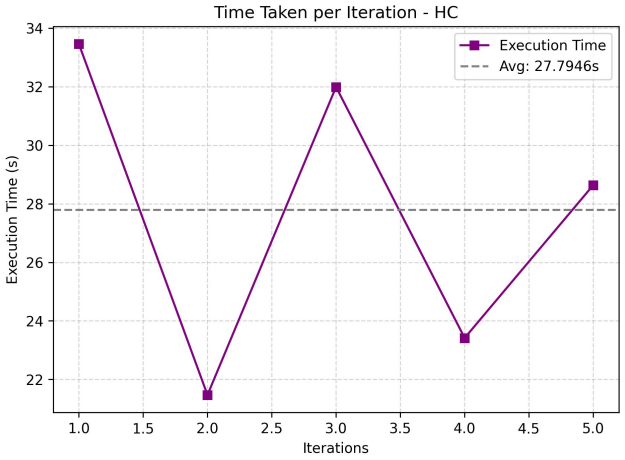
Algorithm 2 IDA* Algorithm

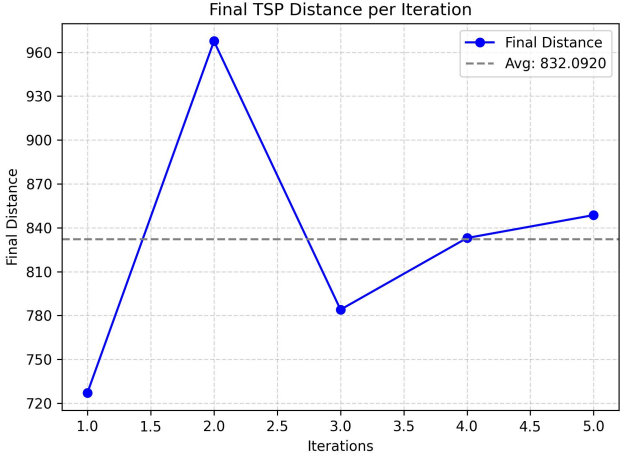
```
function IDA*(x, q)
    if IsGoal(x) = True then
        return Time
    end if
    w ← Expand(x)
    for y ∈ w do
        if q + w(x, y) + h(y) ≤ v then
            r ← IDA*(y, q + w(x, y))
            if r = Time then
                return r
            else if customer > finding min f_new then
                if f(q + w(x, y) + h(y)) < v' then
                    v' ← q + w(x, y) + h(y)
                end if
            end if
        end if
    end for
    return False
end function
function DriverIDA*(s)
    r ← False
    u' ← h(s)
    while r = False do
        f : u ← u'
        u' ← ∞
        v ← IDA*(s, 0)
    end while
end function
```

Observations from Experimental Comparison of IDA* and DFBnB on FrozenLake

Each iteration of both IDA* and DFBnB was executed on the same environment setup: identical start state, goal state, and lake (obstacle) placements. However, this environment setup was varied across different iterations to ensure fairness and generality. Based on this, following observations can be made:

1. Despite using the same admissible heuristic, IDA* consistently expanded fewer nodes than DFBnB. This indicates that IDA*'s iterative deepening enables more effective pruning and limits unnecessary exploration compared to DFBnB’s bound-based approach.
2. IDA* was faster in all runs, primarily due to its lower node expansion.
3. Both algorithms are complete and optimal—they reliably find the shortest path when one exists, given the admissible heuristic.
4. IDA* adapts its search space based on incremental f-value thresholds, resulting in balanced and focused exploration. In contrast, DFBnB can prematurely explore deeper branches, especially when the initial bound is loose, leading to sub-optimal paths in early phases of the search.
5. IDA* is more memory efficient than DFBnB as DFBnB has additional overhead from maintaining best-cost bounds and visited paths.





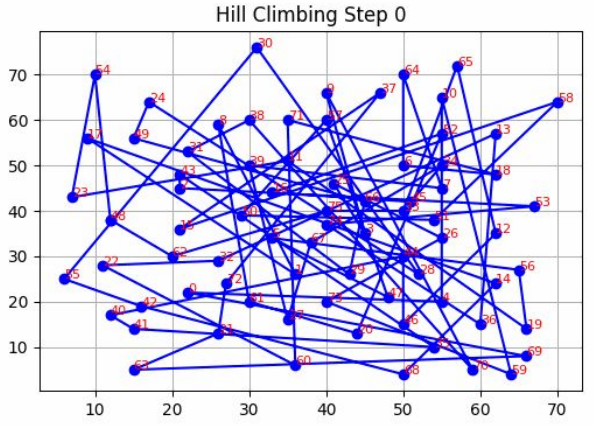
Hill Climbing

We address the TSP using **Steepest Ascent Hill Climbing**. The algorithm initializes with a **random permutation** of cities (which satisfies the TSP constraints—each city is visited exactly once, and the tour returns to the starting point). We iteratively improve this solution by **exploring its neighborhood**, i.e. all possible pairwise city swaps within the tour. For each iteration, the algorithm evaluates the total tour length (cost) of all neighbors and selects the **best node** with a lower cost than the current tour where only the best improving move is accepted.

This process continues **greedily** and terminates when there is no neighboring configuration yielding a lower cost than the current one. Importantly, throughout the search, all intermediate configurations remain valid solutions to the TSP.

This form of Hill Climbing is susceptible to getting trapped in **local minima/maxima**, it is not complete and does not guarantee the optimal solution .

For each iteration, a random permutation is taken and we attempt to reach the optimal solution possible. However, using variations of HC could result better results, eg, Stochastic HC.

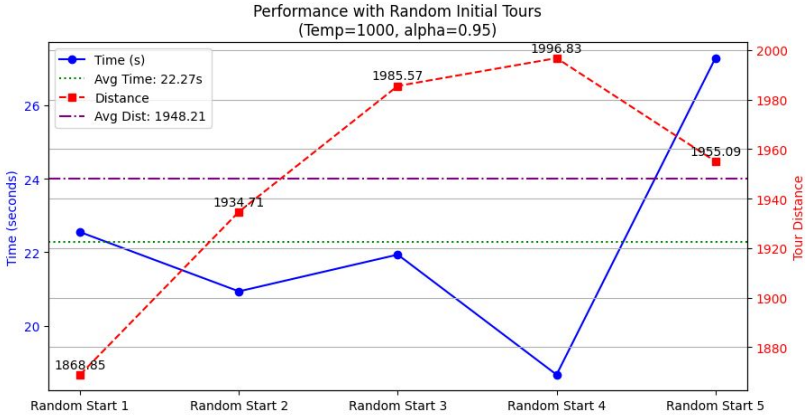


Algorithm 3 Hill-Climbing with Heuristic Guidance

Input: Initial state s, heuristic $h(\cdot)$
Output: Improved state or local optimum

```
current ← s
improved ← True
while improved do
    improved ← False
    neighbors ← Expand(current)
    best_neighbor ← SelectBest(neighbors, h)
    if h(best_neighbor) > h(current) then
        current ← best_neighbor
        improved ← True
    end if
end while
return current
```

The total distance serves as the heuristic function:
h(tour)=total length of the tour



Simulated Annealing

Simulated Annealing (SA) is a metaheuristic optimization technique inspired by the physical process of annealing in metallurgy — where a material is heated to a high temperature and then cooled slowly to reach a low-energy crystalline state. In optimization, SA is used to escape local minima by allowing probabilistic acceptance of worse solutions early on, with decreasing probability as the "temperature" reduces.

The acceptance criteria in Simulated Annealing (SA) determine whether a newly generated solution should replace the current one — a decision that's crucial for balancing exploration (trying diverse solutions) and exploitation (refining good solutions).

1. If the new solution is better (i.e., it has a lower cost or shorter distance in TSP), it is always accepted. This ensures that the algorithm continues improving the solution whenever possible.
2. If the new solution is worse, it may still be accepted with a certain probability, given by the Boltzmann-like formula:

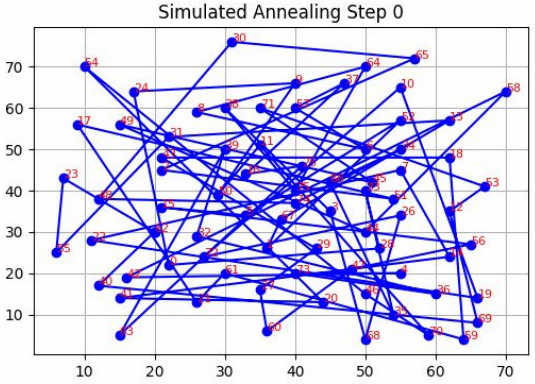
$$P = e^{-\frac{E(w') - E(w)}{t}}$$

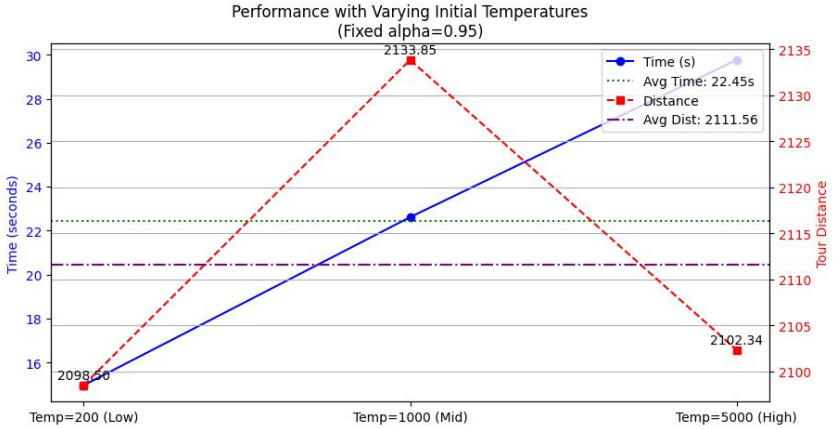
Algorithm 4 Simulated Annealing

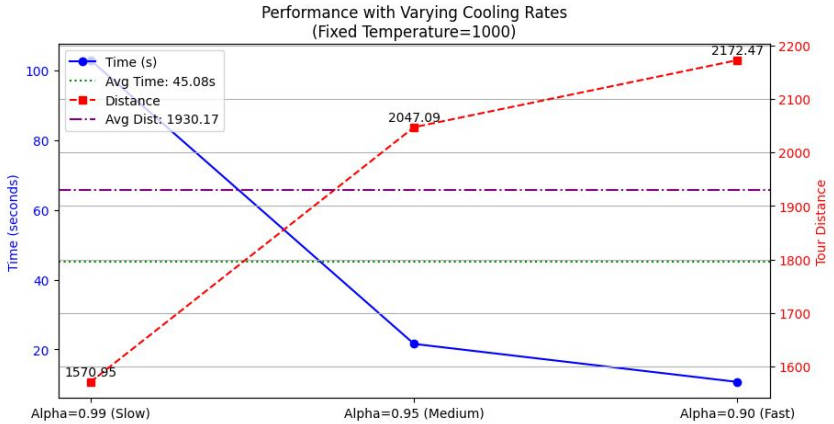
Algorithm Parameters: Objective function $E : \Omega \to R$, Cooling Schedule $\mathcal{T}$, Neighboring Candidate Generator $\mathcal{N}$, ϵ
Input: Parameter Space Ω, Starting Temperature t, Stopping Temperature t_0

$w \leftarrow \text{randomSample}(\Omega)$
$E(w) \leftarrow \infty$
while $t \geq t_0$ **do**
  $w' \leftarrow \mathcal{N}(w, \text{rand}(0, 1))$
  if $\|E(w) - E(w')\| < \epsilon$ **then**
    $w \leftarrow w'$
  else
    $w \leftarrow w'$ **with probability** $e^{-\frac{E(w') - E(w)}{t}}$
  end if
  $t \leftarrow \mathcal{T}(t)$
end while

Heuristic function or Objective function - E(w):
Computes the total distance of the tour w, which is the sum of Euclidean distances (or graph weights) between consecutive cities in the tour, including returning to the starting city.







The performance of Simulated Annealing (SA) in solving the TSP is significantly influenced by the choice of initial temperature and cooling rate (alpha).

The initial temperature governs the extent of early exploration — higher values (e.g., 5000) permit more frequent acceptance of worse solutions, enabling the algorithm to escape local minima, but at the cost of longer convergence time. However, lower temperatures (e.g., 200) reduce exploration, making the algorithm faster but prone to getting stuck in suboptimal regions of the solution space.

This is evident in the first graph, where a mid-range temperature of 1000 balances exploration and convergence, achieving the best average tour distance, while very high or very low temperatures degrade performance either due to stagnation or excessive time.

The second graph explores the effect of varying the cooling rate alpha, which dictates how quickly the temperature decreases. A slower cooling rate (alpha = 0.99) extends the exploration phase, improving solution quality but significantly increasing computation time, as seen from the long execution time and the best tour distance achieved. On the other hand, a faster cooling rate (alpha = 0.90) results in quicker convergence, but at the expense of suboptimal tour distances, indicating premature convergence.

Thus, the graphs collectively highlight a trade-off in SA between solution quality and execution time: effective tuning of temperature and alpha is crucial to balance exploration and exploitation. An ideal setup for TSP requires a moderately high initial temperature (to explore the space freely in early stages) and a gradual cooling schedule (to allow thorough refinement of the solution in later stages).