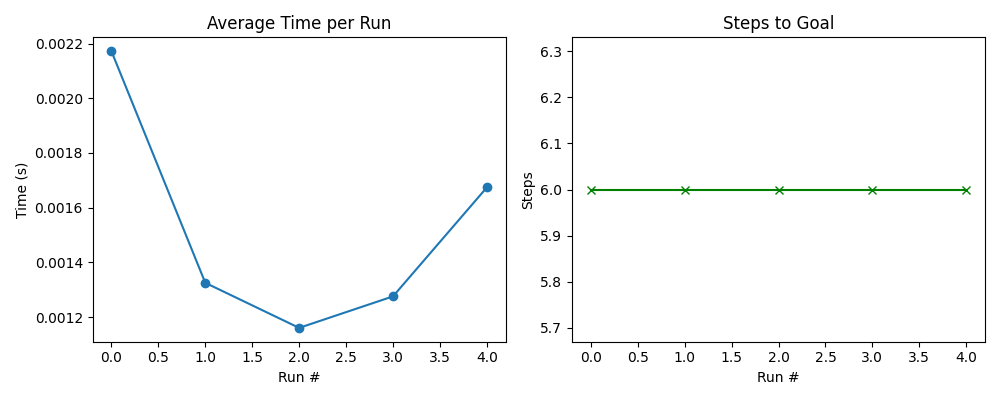
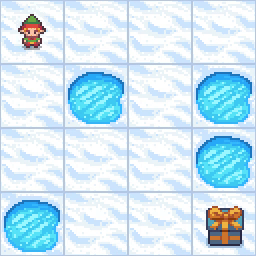
**Assignment 2: Search and Optimization**

1. **Branch & Bound (Frozen Lake Env)**

* **Heuristic Function used**
* **def manhattan\_heuristic(state, goal=15):**
* **row1, col1 = divmod(state, 4)**
* **row2, col2 = divmod(goal, 4)**
* **return abs(row1 - row2) + abs(col1 - col2)**
* **Results**

****

* **GIF**

****

* **Observations (BnB on Frozen Lake)**

#### **Average Time per Run:**

* The time taken per run ranges from approximately **0.0011s to 0.0022s**.
* Run 0 shows the **highest time** (~0.0022s), while Run 2 shows the **lowest** (~0.0011s).
* The variation in time is minor but consistent, suggesting **non-deterministic behavior** due to factors like random seed initialization or environmental variance.

#### **Steps to Goal:**

* The number of steps to reach the goal is **consistently 6** across all 5 runs.
* This consistency indicates a **stable path-finding capability** of the BnB algorithm in this deterministic environment.

**Analysis**

* BnB reliably finds the shortest path in Frozen Lake due to the deterministic nature of the environment.
* Time variation is minimal and doesn’t affect performance.
* Algorithm is both **efficient and consistent** for small, grid-based problems.

1. **Iterative Deepening A\* (Frozen Lake Env)**

* **Heuristic Function used**

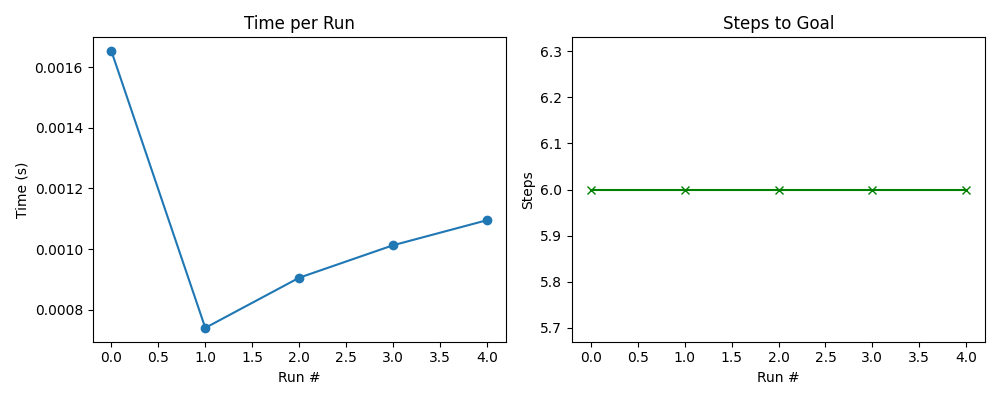
**def manhattan\_heuristic(state, goal=15):**

**row1, col1 = divmod(state, 4)**

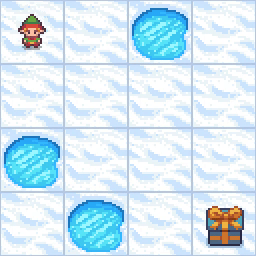
**row2, col2 = divmod(goal, 4)**

**return abs(row1 - row2) + abs(col1 - col2)**

* **Results**

****

* **GIF**

****

* **Observations**
* **Time per Run:** Time ranges from **0.00073s to 0.00166s** across 5 runs.  
   The first run is the slowest, with subsequent runs gradually increasing after a quick drop in Run 1.
* **Steps to Goal:** All runs took **exactly 6 steps** to reach the goal — showing consistent and optimal solutions.
* **Convergence:** IDA\* reached the goal in every run well within the **10-minute (τ)** limit.

### **Analysis**

* **Consistency:** Like BnB, IDA\* consistently finds the shortest path (6 steps) due to the fixed, deterministic structure of Frozen Lake.
* **Performance:** Slight time differences are likely due to **depth-limit iterations and heuristic evaluations** inherent in IDA\*.
* **Efficiency:** Overall time remains low (within milliseconds), showing that IDA\* is **efficient for small search spaces**.

1. **Hill Climbing (Travelling Salesman Problem Env)**

* **Heuristic Function used**

**def nearest\_neighbor\_heuristic(cities):**

**unvisited = cities[:]**

**route = [unvisited.pop(0)]**

**while unvisited:**

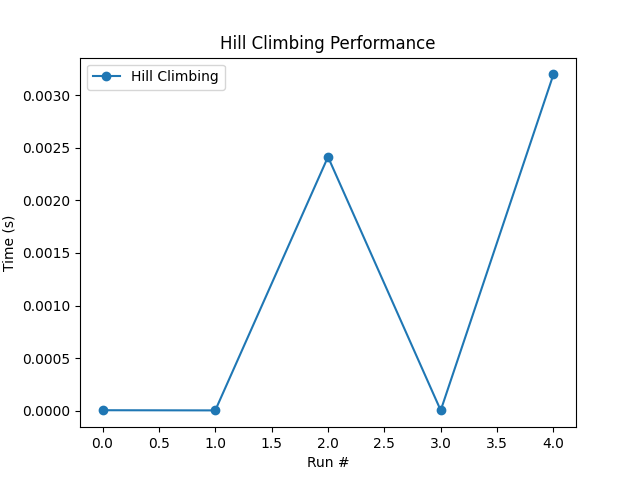
**next\_city = min(unvisited, key=lambda city: math.dist(route[-1], city))**

**route.append(next\_city)**

**unvisited.remove(next\_city)**

**return route**

* **Result**

****

* **GIF**

****

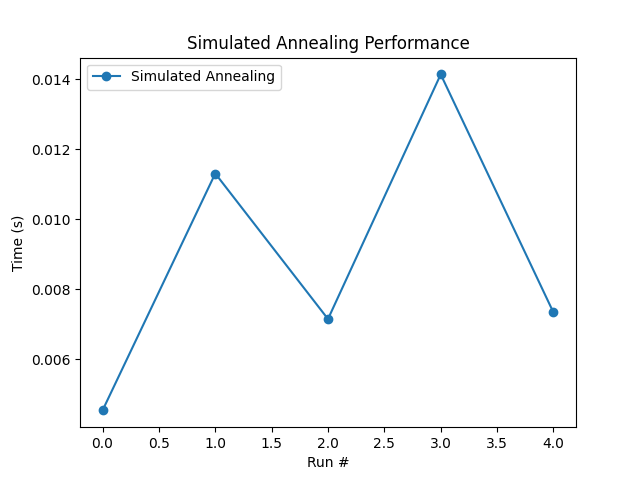
* **Observations**
  + Time taken per run shows high variability across the five runs.
  + Some runs took almost 0 time, while others peaked at ~0.0032 seconds.
  + This inconsistency suggests unpredictable convergence behavior.

### **Explanation:**

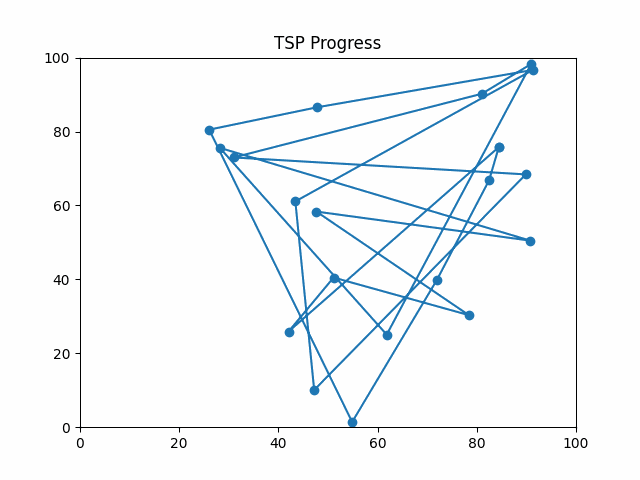
* **Hill Climbing** is **greedy** and only moves towards better solutions.
* Due to this, it often gets **stuck in local minima**, especially in complex problems like TSP.
* The runs with **0 time** likely ended quickly by hitting a local minimum very early, with **no better neighbor found**.
* The runs that took longer might have found **a few improving neighbors** before getting stuck, resulting in higher execution time.
* Since TSP has a **large search space**, and Hill Climbing doesn't backtrack, results are **not consistently optimal**.

1. **Simulated\_Annealing (Travelling Salesman Problem Env)**

* **Heuristic Function used**
* **def nearest\_neighbor\_heuristic(cities):**
* **unvisited = cities[:]**
* **route = [unvisited.pop(0)]**
* **while unvisited:**
* **next\_city = min(unvisited, key=lambda city: math.dist(route[-1], city))**
* **route.append(next\_city)**
* **unvisited.remove(next\_city)**
* **return route**
* **Result**

****

* **GIF**

****

* **Observations**
  + **Execution Time (per run):** Ranges between **0.004s to 0.014s**.
  + **Performance:** Stable and efficient compared to Hill Climbing.
  + **Path quality:** Generally finds better solutions than Hill Climbing due to randomness.
* **Analysis** 
  + **Escapes Local Optima:** Accepts worse solutions at high temperature → helps avoid getting stuck.
  + **Gradual Cooling (alpha=0.995):** Allows broad exploration early on, fine-tuning later.
  + **Stochastic Nature:** Each run varies slightly due to randomness, but consistently avoids poor-quality traps.
  + **Better for TSP:** TSP has a huge search space → SA handles it better than greedy methods like Hill Climbing.