Experiment 7

Aim: To implement Travelling salesman problem using Hill climbing algorithm

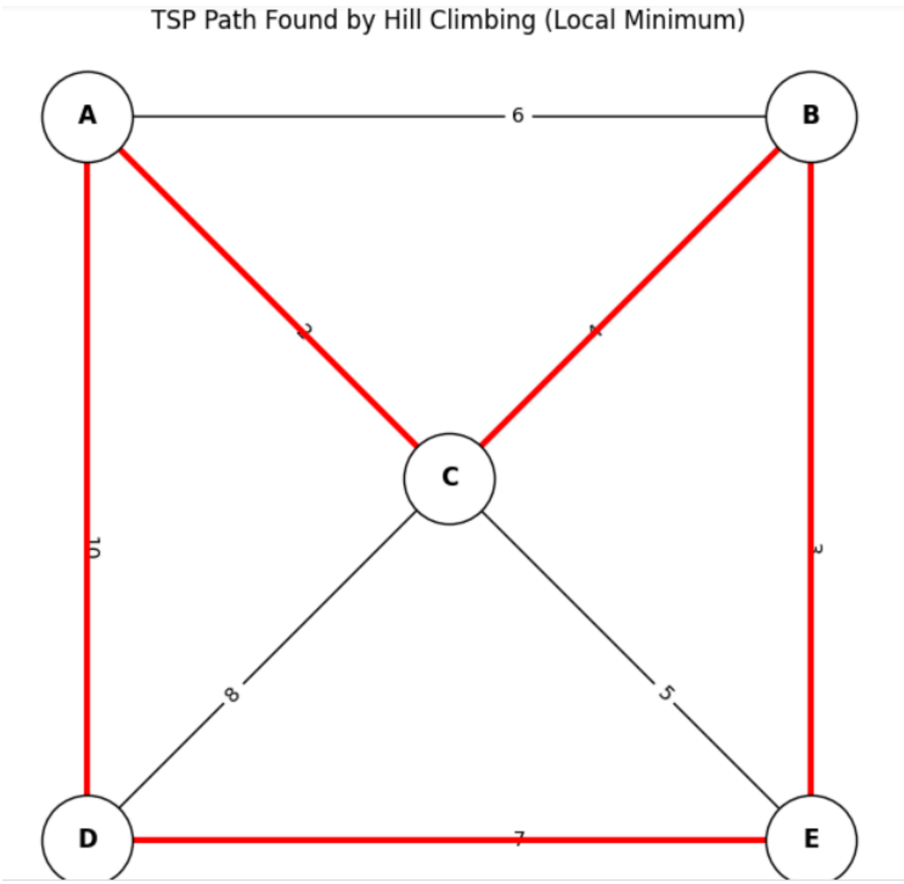
Code:

```
import heapq
import networkx as nx
import matplotlib.pyplot as plt
# Function to calculate total distance of a path
def total_distance(path, graph):
    distance = 0
    for i in range(len(path) - 1):
        if graph.has_edge(path[i], path[i+1]):  # Ensure valid edge exists
            distance += graph[path[i]][path[i+1]]['weight']
        else:
            return float('inf')  # Penalize invalid paths

    # Ensure returning to the start node
    if graph.has_edge(path[-1], path[0]):
        distance += graph[path[-1]][path[0]]['weight']
    else:
        return float('inf')  # Penalize invalid paths

    return distance

# Hill Climbing algorithm for TSP
def hill_climb(graph, max_restarts=10):
    best_path = None
    best_distance = float('inf')

    for _ in range(max_restarts):  # Restart search if stuck in a local
minimum
        path = list(graph.nodes)
        random.shuffle(path)
        current_path = path[:]
        current_distance = total_distance(current_path, graph)

        while True:
            neighbors = []
            for i in range(len(current_path) - 1):
```

```
            for j in range(i + 1, len(current_path)):
                new_path = current_path[:]
                new_path[i], new_path[j] = new_path[j], new_path[i]
                neighbors.append(new_path)

        next_path = min(neighbors, key=lambda p: total_distance(p,
graph))
        next_distance = total_distance(next_path, graph)

        if next_distance < current_distance:
            current_path = next_path
            current_distance = next_distance
        else:
            break  # Local minimum reached

    if current_distance < best_distance:
        best_path = current_path
        best_distance = current_distance

    return best_path + [best_path[0]], best_distance

best_path, best_distance = hill_climb(G)

print(f"Best path found: {best_path}")
print(f"Total distance: {best_distance}")

plt.figure(figsize=(6, 6))
nx.draw(G, pos, with_labels=True, node_size=2000, node_color="white",
edge_color="black",
        font_size=12, font_weight="bold", edgecolors="black", arrows=True)

# Draw edge weights above edges
edge_labels = {(u, v): w for u, v, w in edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=10, label_pos=0.6)

# Highlight the best path found by Hill Climbing
path_edges = list(zip(best_path, best_path[1:]))  # Get path edges
nx.draw_networkx_edges(G, pos, edgelist=path_edges, width=3,
edge_color="red", arrows=True, arrowstyle="->")  # Highlight path
```

```
# Show plot
plt.title("TSP Path Found by Hill Climbing")
plt.show()
```

Output




```
Best path found: ['D', 'A', 'C', 'B', 'E', 'D']
Total distance: 26
```