```python
from heapq import heappush, heappop

# Function to calculate Manhattan Distance
def manhattan(state, goal):
    distance = 0
    for i in range(1, 9):
        xi, yi = divmod(state.index(i), 3)
        xg, yg = divmod(goal.index(i), 3)
        distance += abs(xi - xg) + abs(yi - yg)
    return distance

# Function to generate possible moves
def get_neighbors(state):
    neighbors = []
    i = state.index(0)
    x, y = divmod(i, 3)
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_i = nx * 3 + ny
            new_state = state[:]
            new_state[i], new_state[new_i] = new_state[new_i], new_state[i]
            neighbors.append(new_state)
    return neighbors

# A* Algorithm
def solve_puzzle(start, goal):
    pq = []
    heappush(pq, (manhattan(start, goal), 0, start, []))
    visited = set()

    while pq:
        f, g, state, path = heappop(pq)
        if tuple(state) in visited:
            continue
        visited.add(tuple(state))

        if state == goal:
            return path + [state]

        for neighbor in get_neighbors(state):
            if tuple(neighbor) not in visited:
                heappush(pq, (g + 1 + manhattan(neighbor, goal), g + 1, neighbor, path + [state]))

    return None
```

```
# Example run
start = [1, 2, 3,
         4, 0, 6,
         7, 5, 8]  # 0 is the empty space

goal = [1, 2, 3,
        4, 5, 6,
        7, 8, 0]

solution = solve_puzzle(start, goal)

if solution:
    print("\nSteps to solve:")
    for step in solution:
        print(step[0:3])
        print(step[3:6])
        print(step[6:9])
        print()
else:
    print("No solution found.")
```

```
Steps to solve:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

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