```
import heapq
import copy

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

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

def heuristic_misplaced_tiles(state):
    misplaced = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0 and state[i][j] != goal_state[i][j]:
                misplaced += 1
    return misplaced

def heuristic_manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:
                goal_x, goal_y = divmod(value - 1, 3)
                distance += abs(i - goal_x) + abs(j - goal_y)
    return distance

def find_blank_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def generate_possible_moves(state):
    x, y = find_blank_position(state)
    new_states = []

    for move in moves:
        new_x, new_y = x + move[0], y + move[1]
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = copy.deepcopy(state)
            new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]
            new_states.append(new_state)

    return new_states

def is_goal(state):
    return state == goal_state

def a_star_search(initial_state, heuristic_type="misplaced_tiles"):
    def heuristic(state):
        if heuristic_type == "misplaced_tiles":
            return heuristic_misplaced_tiles(state)
        elif heuristic_type == "manhattan_distance":
            return heuristic_manhattan_distance(state)
        else:
            return 0

    priority_queue = []
    heapq.heappush(priority_queue, (0, 0, initial_state, []))
    explored = set()

    while priority_queue:
        f_n, g_n, current_state, path = heapq.heappop(priority_queue)

        if is_goal(current_state):
            return path + [current_state]

        state_tuple = tuple(tuple(row) for row in current_state)
        if state_tuple in explored:
            continue
        explored.add(state_tuple)

        for new_state in generate_possible_moves(current_state):
            new_g_n = g_n + 1
            new_f_n = new_g_n + heuristic(new_state)
            heapq.heappush(priority_queue, (new_f_n, new_g_n, new_state, path + [current_state]))
```

```
        return None

    def print_puzzle(state):
        for row in state:
            print(row)
        print()

    if __name__ == "__main__":
        initial_state = [[5, 4, 0], [6, 1, 8], [7, 3, 2]]

        print("Initial State:")
        print_puzzle(initial_state)

        print("Solving with Misplaced Tiles heuristic:")
        path = a_star_search(initial_state, heuristic_type="misplaced_tiles")
        if path:
            for step in path:
                print_puzzle(step)
        else:
            print("No solution found.")

        print("Solving with Manhattan Distance heuristic:")
        path = a_star_search(initial_state, heuristic_type="manhattan_distance")
        if path:
            for step in path:
                print_puzzle(step)
        else:
            print("No solution found.")
```

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

Solving with Misplaced Tiles heuristic:
[5, 4, 0]
[6, 1, 8]
[7, 3, 2]

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

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

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

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

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

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

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

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

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

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

[6, 3, 5]
```

```
[0, 1, 4]
[7, 2, 8]

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