```
import numpy as np
import heapq

class Node:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent
        self.zero_position = self.find_zero(state)
        self.g = 0  # Cost from start to current node
        self.h = self.heuristic()  # Estimated cost from current to goal
        self.f = self.g + self.h  # Total cost

    def find_zero(self, state):
        return tuple(np.argwhere(state == 0)[0])

    def heuristic(self):
        # Using Misplaced Tiles heuristic
        goal_state = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])
        misplaced_count = 0
        for i in range(3):
            for j in range(3):
                if self.state[i, j] != goal_state[i, j] and self.state[i, j] != 0:
                    misplaced_count += 1
        return misplaced_count

    def generate_children(self):
        children = []
        x, y = self.zero_position
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_state = self.state.copy()
                new_state[x, y], new_state[new_x, new_y] = new_state[new_x, new_y], new_state[x, y]
                children.append(Node(new_state, self))
        return children

    def __lt__(self, other):
        return self.f < other.f

def a_star(start_state):
    start_node = Node(start_state)
    goal_state = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])

    open_set = []
    closed_set = set()

    heapq.heappush(open_set, start_node)

    while open_set:
        current_node = heapq.heappop(open_set)

        if np.array_equal(current_node.state, goal_state):
            return current_node

        closed_set.add(tuple(map(tuple, current_node.state)))

        for child in current_node.generate_children():
            if tuple(map(tuple, child.state)) in closed_set:
                continue

            child.g = current_node.g + 1
            child.h = child.heuristic()
            child.f = child.g + child.h

            if not any(child.f < node.f and np.array_equal(child.state, node.state) for node in open_set):
                heapq.heappush(open_set, child)

    return None

def print_solution(node):
    path = []
    while node:
        path.append(node.state)
        node = node.parent
    for state in reversed(path):
        print(state)

# Example usage
if __name__ == "__main__":
```

```
start_state = np.array([[2, 8, 3], [1, 6, 4], [7, 0, 5]])
solution_node = a_star(start_state)

if solution_node:
    print("Solution found:")
    print_solution(solution_node)
else:
    print("No solution exists.")
```

```
Solution found:
 [[2 8 3]
  [1 6 4]
  [7 0 5]]
 [[2 8 3]
  [1 0 4]
  [7 6 5]]
 [[2 0 3]
  [1 8 4]
  [7 6 5]]
 [[0 2 3]
  [1 8 4]
  [7 6 5]]
 [[1 2 3]
  [0 8 4]
  [7 6 5]]
 [[1 2 3]
  [8 0 4]
  [7 6 5]]
```