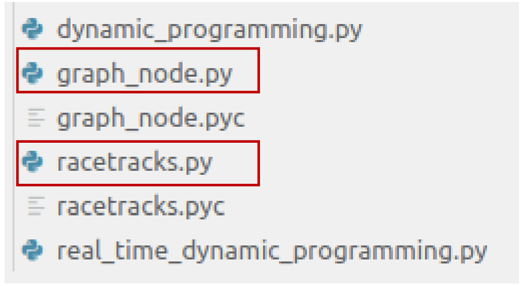
深蓝学院
shenlanxueyuan.com

第七章作业讲评

主讲人 武庆斌

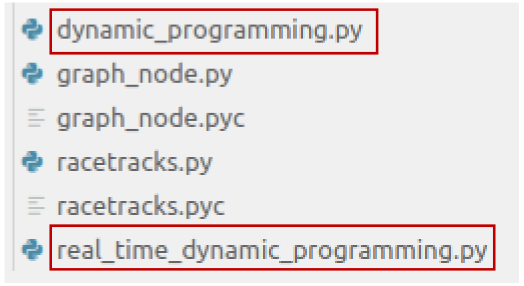
大纲

- MDP框架



```
dynamic_programming.py
graph_node.py
graph_node.pyc
racetracks.py
racetracks.pyc
real_time_dynamic_programming.py
```

- DP&RTDP流程



```
dynamic_programming.py
graph_node.py
graph_node.pyc
racetracks.py
racetracks.pyc
real_time_dynamic_programming.py
```

MDP formalize

Markov Decision Process (MDP)

A MDP is a 4-tuple (S, A, P, R) in learning field, or (X, U, P, L) in planning field:

- S or X is *sate space*,
- A or U is *(robot) action space*,
- $P\left(x_{k+1} | x_k, u_k\right)$ is *state transition function* under probabilistic model,
    - which degenerates to a set $X_{k+1}\left(x_k, u_k\right)$ under nondeterministic model,
- $R\left(x_k, x_{k+1}\right)$ is *the immediate reward*, or the negative one-step cost $-l(x_k, u_k, \theta_k)$, after transitioning from x_k to x_{k+1} due to u, θ.

MDP formalize

1.state space

① $X = \{(x, y) \mid 0 \leq x \leq 11, 0 \leq y \leq 34\}$

② $X_I = \{\text{green grids}\}$

$X_F = \{\text{yellow grids}\}$

```
START_LINE = [[0, 3], [0, 4], [0, 5], [0, 6]]
```

```
FINISH_LINE = [[34, 11], [33, 11], [32, 11]]
```

2.action space

③ $U = \{(\ddot{x}, \ddot{y}) | \ddot{x} \in \{0, \pm 1\}, \ddot{y} \in \{0, \pm 1\}\}$

```
ACTION_SPACE = [[1, 1], [0, 1],
```

3.state transition function

④ $\Theta = \{\theta_1, \theta_2\}$

- $\theta_1 : f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k \quad p_1 = 0.1$
- $\theta_2 : f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1} \quad p_1 = 0.9$

```
def connect_to_graph(self, grid):
    for u in ACTION_SPACE:
        self.next_prob_9.append(self.control(u[0], u[1], grid, success=True))
        self.next_prob_1.append(self.control(u[0], u[1], grid, success=False))
```

4.cost function

⑤ $l(\mathbf{x}_k, \mathbf{x}_k, \theta_k) = -1$

```
# TO BE IMPLEMENTED
expected_cost_uk = 0.9 * (1+child_9.g_value) + 0.1*(1+child_1.g_value)
value_uk.append(expected_cost_uk)
```

DP&RTDP

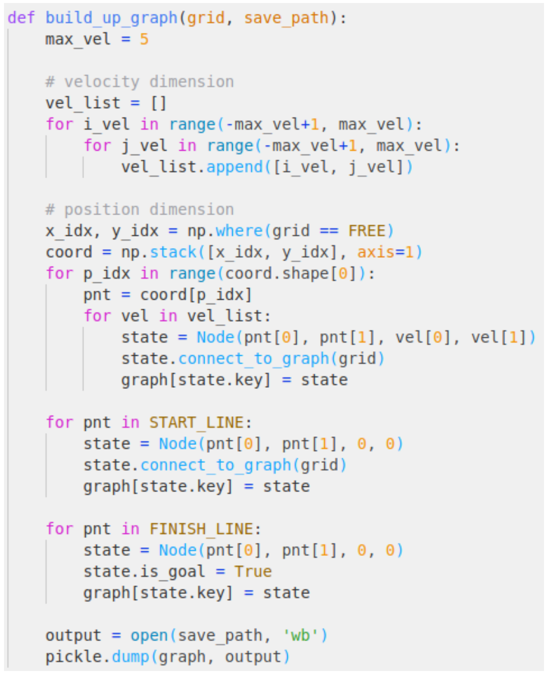
```
Initialize $G$ values of all states to finite values;
while not converge do
    for all the states $x$ do
        $G(x_F) = 0$;
        $G_k(x_k) =$
          $\min_{u_k \in U(x_k)} \{E_{\theta_k}[l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,
          if $x_k \neq x_F$;
    end
end
```

DP

1. Initialize G values of all states to admissible values;
2. Follow greedy policy picking outcomes at random until goal is reached;
3. Backup all states visited on the way;
4. Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;

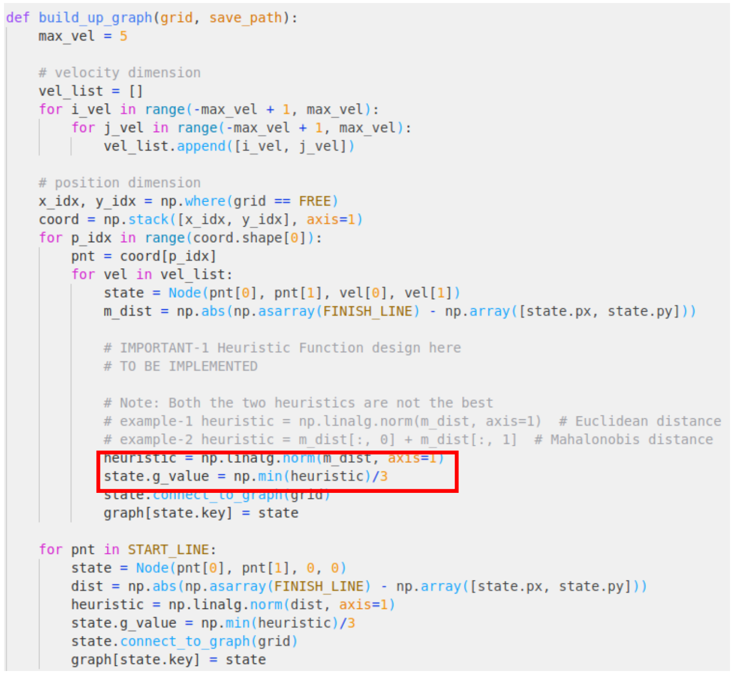
RTDP

重新构建graph



```python
def build_up_graph(grid, save_path):
    max_vel = 5

    # velocity dimension
    vel_list = []
    for i_vel in range(-max_vel+1, max_vel):
        for j_vel in range(-max_vel+1, max_vel):
            vel_list.append([i_vel, j_vel])

    # position dimension
    x_idx, y_idx = np.where(grid == FREE)
    coord = np.stack([x_idx, y_idx], axis=1)
    for p_idx in range(coord.shape[0]):
        pnt = coord[p_idx]
        for vel in vel_list:
            state = Node(pnt[0], pnt[1], vel[0], vel[1])
            state.connect_to_graph(grid)
            graph[state.key] = state

    for pnt in START_LINE:
        state = Node(pnt[0], pnt[1], 0, 0)
        state.connect_to_graph(grid)
        graph[state.key] = state

    for pnt in FINISH_LINE:
        state = Node(pnt[0], pnt[1], 0, 0)
        state.is_goal = True
        graph[state.key] = state

    output = open(save_path, 'wb')
    pickle.dump(graph, output)
```

DP

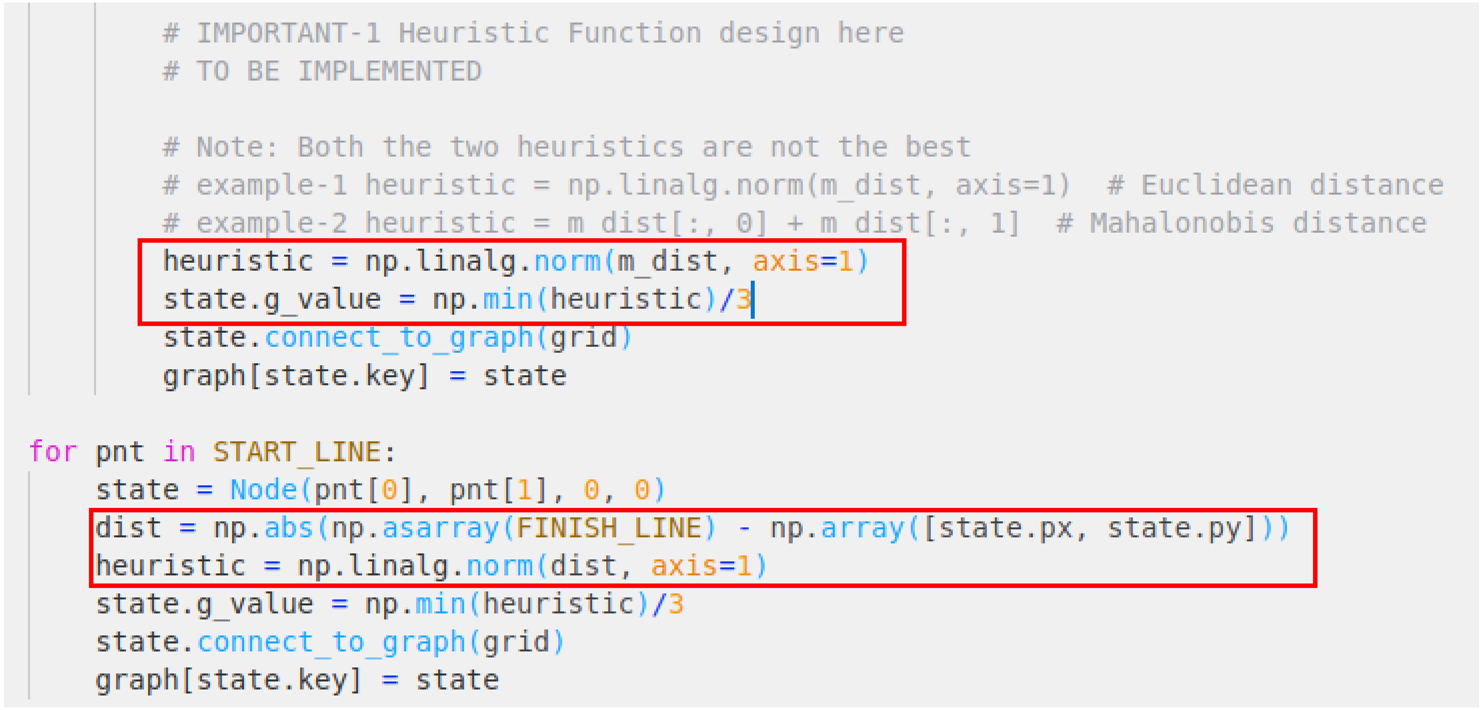


```python
def build_up_graph(grid, save_path):
    max_vel = 5

    # velocity dimension
    vel_list = []
    for i_vel in range(-max_vel + 1, max_vel):
        for j_vel in range(-max_vel + 1, max_vel):
            vel_list.append([i_vel, j_vel])

    # position dimension
    x_idx, y_idx = np.where(grid == FREE)
    coord = np.stack([x_idx, y_idx], axis=1)
    for p_idx in range(coord.shape[0]):
        pnt = coord[p_idx]
        for vel in vel_list:
            state = Node(pnt[0], pnt[1], vel[0], vel[1])
            m_dist = np.abs(np.asarray(FINISH_LINE) - np.array([state.px, state.py]))

            # IMPORTANT-1 Heuristic Function design here
            # TO BE IMPLEMENTED

            # Note: Both the two heuristics are not the best
            # example-1 heuristic = np.linalg.norm(m_dist, axis=1)  # Euclidean distance
            # example-2 heuristic = m_dist[:, 0] + m_dist[:, 1]  # Mahalonobis distance
            heuristic = np.linalg.norm(m_dist, axis=1)
            state.g_value = np.min(heuristic)/3
            state.connect_to_graph(grid)
            graph[state.key] = state

    for pnt in START_LINE:
        state = Node(pnt[0], pnt[1], 0, 0)
        dist = np.abs(np.asarray(FINISH_LINE) - np.array([state.px, state.py]))
        heuristic = np.linalg.norm(dist, axis=1)
        state.g_value = np.min(heuristic)/3
        state.connect_to_graph(grid)
        graph[state.key] = state
```

RTDP

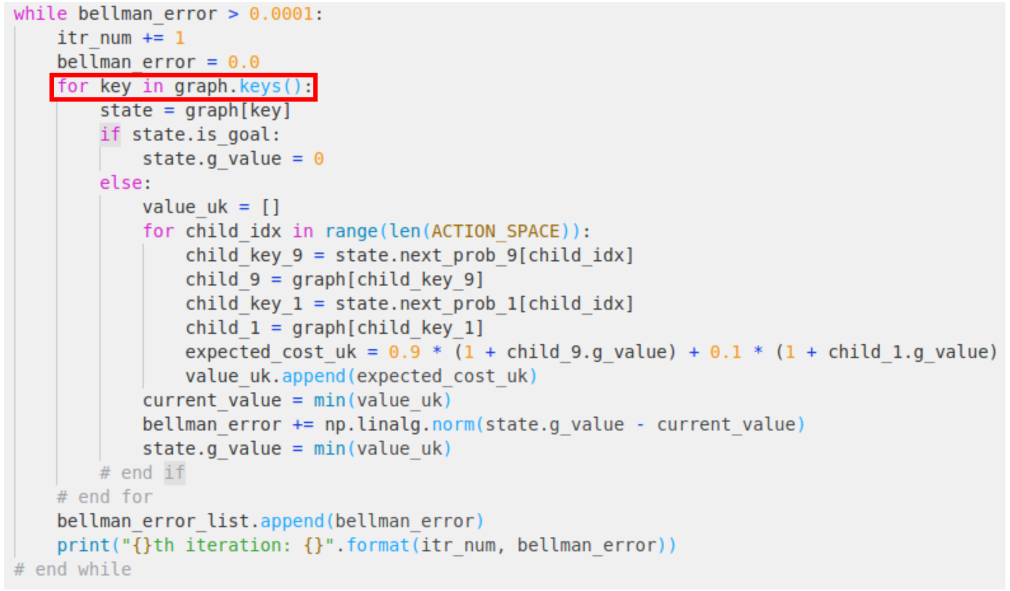
TIPS1：DP的g_value为默认0，RTDP的g_value需要初始化
TIPS2：程序要跑两次，第一次在主函数运行build_up_graph，第二次屏蔽掉主函数的build_up_graph函数。

❶ Initialize G values of all states to admissible values;

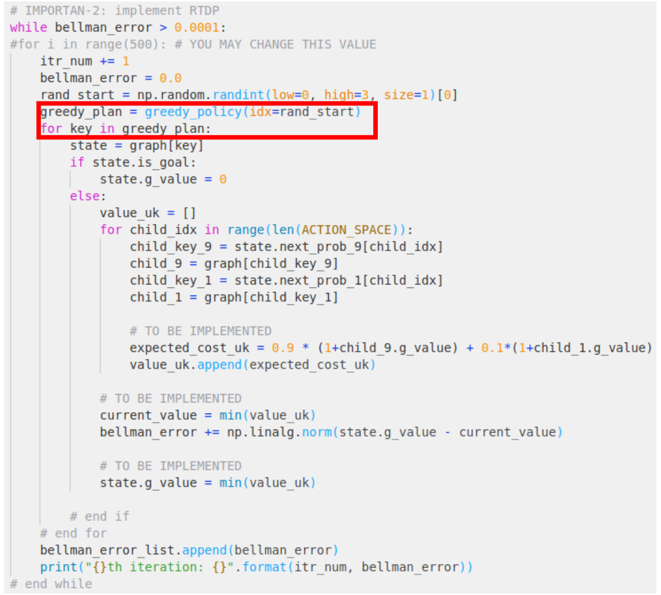


```
        # IMPORTANT-1 Heuristic Function design here
        # TO BE IMPLEMENTED

        # Note: Both the two heuristics are not the best
        # example-1 heuristic = np.linalg.norm(m_dist, axis=1)  # Euclidean distance
        # example-2 heuristic = m_dist[:, 0] + m_dist[:, 1]  # Mahalonobis distance
        heuristic = np.linalg.norm(m_dist, axis=1)
        state.g_value = np.min(heuristic)/3
        state.connect_to_graph(grid)
        graph[state.key] = state

for pnt in START_LINE:
    state = Node(pnt[0], pnt[1], 0, 0)
    dist = np.abs(np.asarray(FINISH_LINE) - np.array([state.px, state.py]))
    heuristic = np.linalg.norm(dist, axis=1)
    state.g_value = np.min(heuristic)/3
    state.connect_to_graph(grid)
    graph[state.key] = state
```

value update



```
while bellman_error > 0.0001:
    itr_num += 1
    bellman_error = 0.0
    for key in graph.keys():
        state = graph[key]
        if state.is_goal:
            state.g_value = 0
        else:
            value_uk = []
            for child_idx in range(len(ACTION_SPACE)):
                child_key_9 = state.next_prob_9[child_idx]
                child_9 = graph[child_key_9]
                child_key_1 = state.next_prob_1[child_idx]
                child_1 = graph[child_key_1]
                expected_cost_uk = 0.9 * (1 + child_9.g_value) + 0.1 * (1 + child_1.g_value)
                value_uk.append(expected_cost_uk)
            current_value = min(value_uk)
            bellman_error += np.linalg.norm(state.g_value - current_value)
            state.g_value = min(value_uk)
        # end if
    # end for
    bellman_error_list.append(bellman_error)
    print("{}th iteration: {}".format(itr_num, bellman_error))
# end while
```



```
# IMPORTAN-2: implement RTDP
while bellman_error > 0.0001:
#for i in range(500): # YOU MAY CHANGE THIS VALUE
    itr_num += 1
    bellman_error = 0.0
    rand_start = np.random.randint(low=0, high=3, size=1)[0]
    greedy_plan = greedy_policy(idx=rand_start)
    for key in greedy_plan:
        state = graph[key]
        if state.is_goal:
            state.g_value = 0
        else:
            value_uk = []
            for child_idx in range(len(ACTION_SPACE)):
                child_key_9 = state.next_prob_9[child_idx]
                child_9 = graph[child_key_9]
                child_key_1 = state.next_prob_1[child_idx]
                child_1 = graph[child_key_1]

                # TO BE IMPLEMENTED
                expected_cost_uk = 0.9 * (1+child_9.g_value) + 0.1*(1+child_1.g_value)
                value_uk.append(expected_cost_uk)

            # TO BE IMPLEMENTED
            current_value = min(value_uk)
            bellman_error += np.linalg.norm(state.g_value - current_value)

            # TO BE IMPLEMENTED
            state.g_value = min(value_uk)

        # end if
    # end for
    bellman_error_list.append(bellman_error)
    print("{}th iteration: {}".format(itr_num, bellman_error))
# end while
```

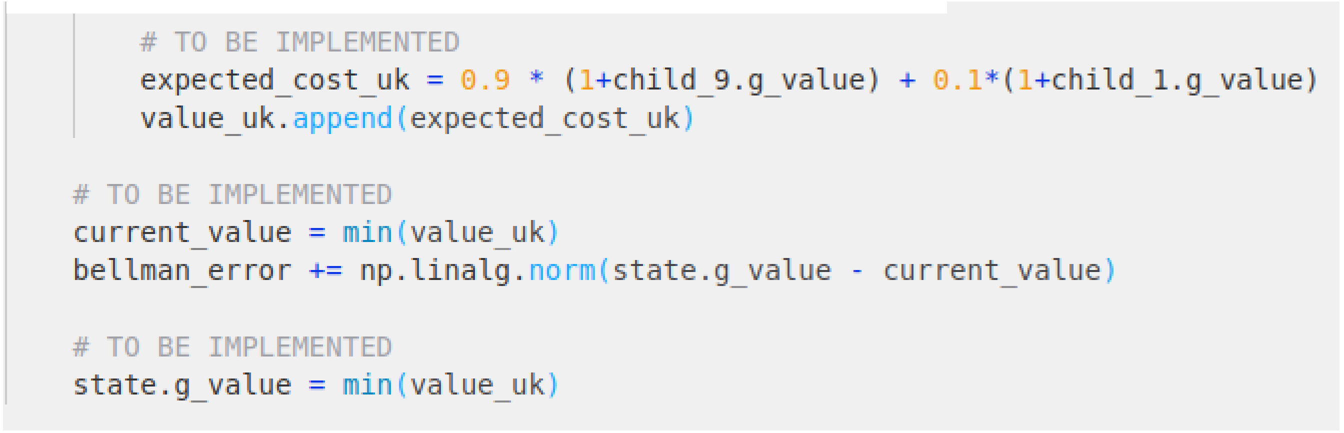
DP　　　　　　　　　　　　RTDP

TIPS1：DP更新graph上的所有节点，RTDP只更新已搜索到路径上的点。

value update

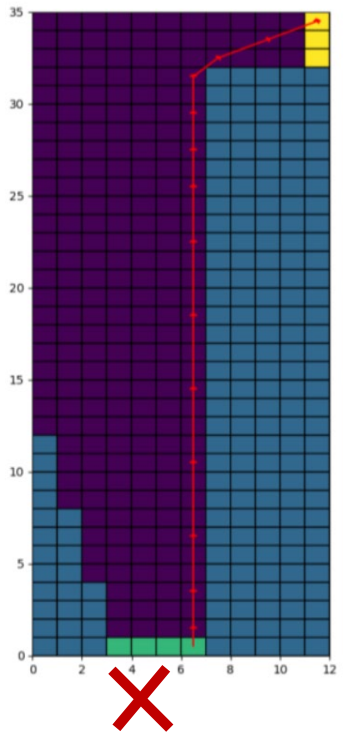
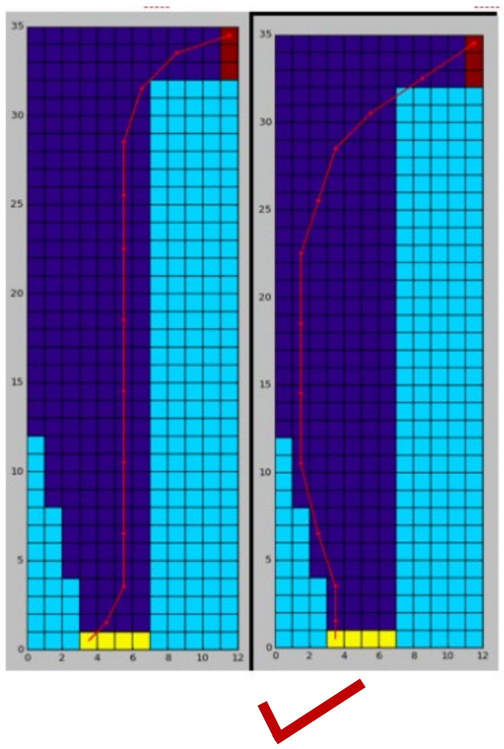
深蓝学院
shenlanxueyuan.com

1. Initialize G values of all states to admissible values;
2. Follow greedy policy picking outcomes at random until goal is reached;
3. Backup all states visited on the way;
4. Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



```
    # TO BE IMPLEMENTED
    expected_cost_uk = 0.9 * (1+child_9.g_value) + 0.1*(1+child_1.g_value)
    value_uk.append(expected_cost_uk)

# TO BE IMPLEMENTED
current_value = min(value_uk)
bellman_error += np.linalg.norm(state.g_value - current_value)

# TO BE IMPLEMENTED
state.g_value = min(value_uk)
```

最优路径判断

TIPS1：由于cost function是执行一个动作加1，所以用动作个数来判断是否达到最优，即路径段数
TIPS2：最优路径的判断是路径的段数，基本是12段，和形状没有关系

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening