Interdisciplinary Division of Aeronautical and Aviation Engineering
航空工程跨領域學部

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

# AAE2004 Introduction to Aviation Systems
# AAE
# Design of Path Planning Algorithm for Aircraft Operation

## Week 9 (Introduction to Path Planning)


Dr Li-Ta Hsu, Dr **Guohao Zhang**, and Dr Weisong Wen

Assisted by

Miss Hiu Yi HO (Queenie), Miss Yan Tung LEUNG (Nikki),

Mr Hoi Fung NG (Ivan) and Mr Feng HUANG (Darren)

# Necessary Information

- Course Repository (project download) link:
- https://github.com/IPNL-POLYU/PolyU_AAE2004_Github_Project

- TA Information & Contact:
  - Group 1-3: Queenie Ho queenie-hy.ho@connect.polyu.hk
  - Group 4-6: Nikkie Leung nikkie-yt.leung@connect.polyu.hk
  - Group 7-9: Ivan Ng hf-ivan.ng@connect.polyu.hk
  - Group 10-12: Darren Huang darren-f.huang@connect.polyu.hk

# Week 2 Content

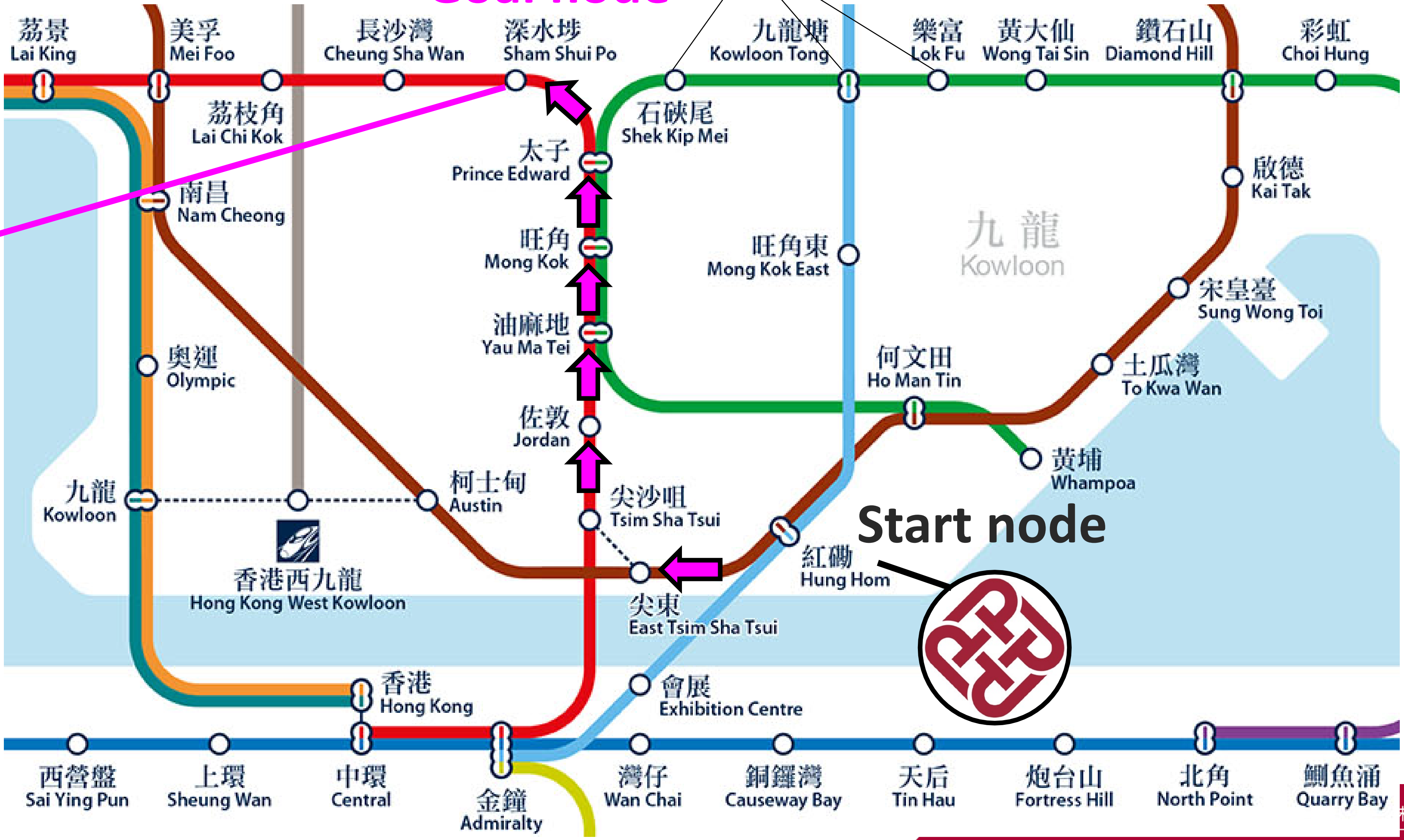
1. Introduction to A* Path Planning Algorithm
2. Cost Intensive Areas
3. Path Planning Programming Guide
4. Project Compulsory Tasks

# Introduction to A* Path Planning Algorithm

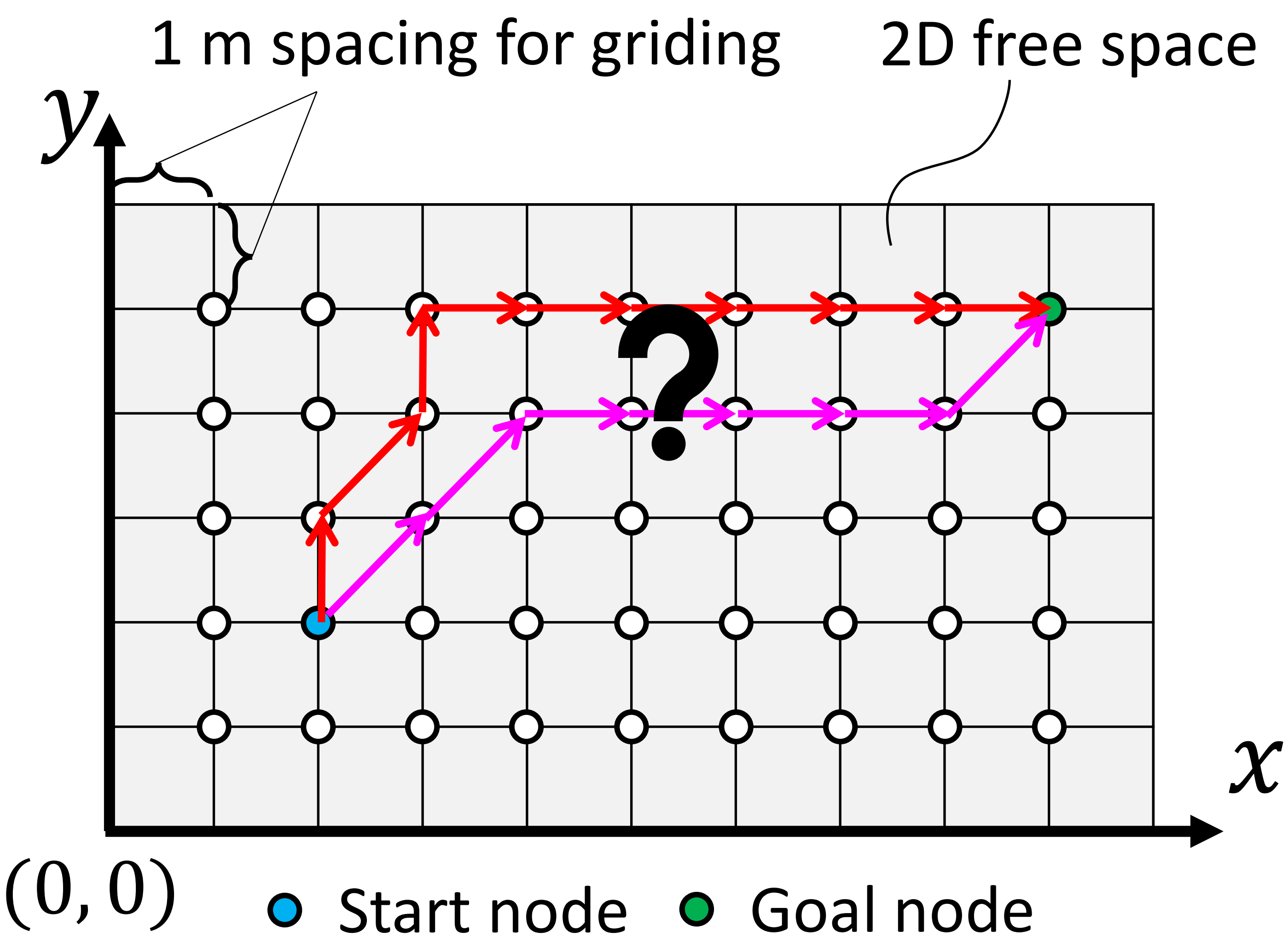
# Daily Path Planning

**MTR station** – available **node** to go

**Goal node**

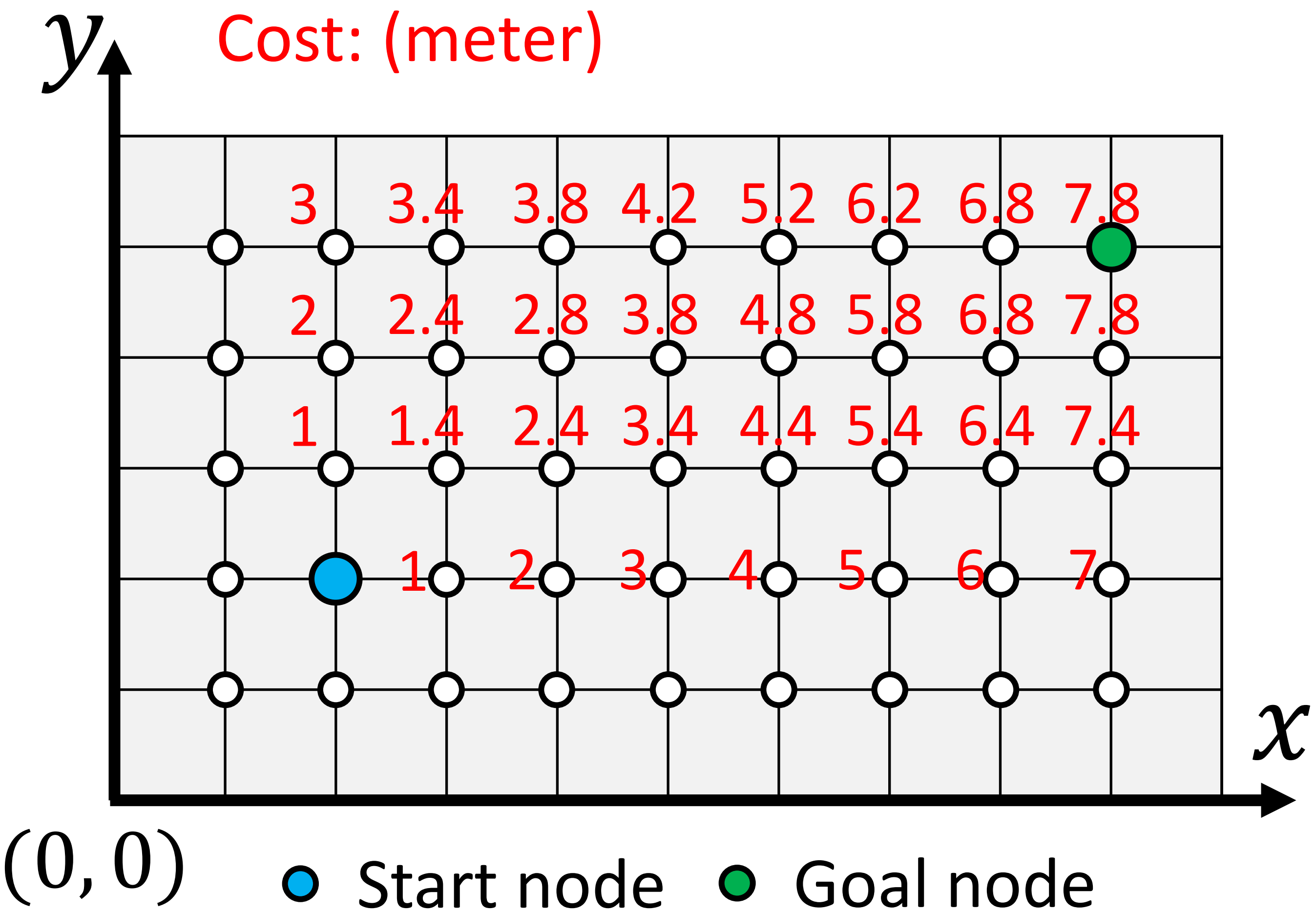
# Definition of Path Planning




- **Node** — All potential position you can go across with a unique position $(x, y)$
- **Search space** — A collection of nodes, like all board positions of a board game.
- **Objective of path planning**— Find the shortest routes with smallest cost from start node to goal node.

**Which one is better?**

# Path Planning by Checking All Available Nodes




1 m spacing for griding

Test each possible nodes one-by-one from Start node

Record its shortest path between Start node

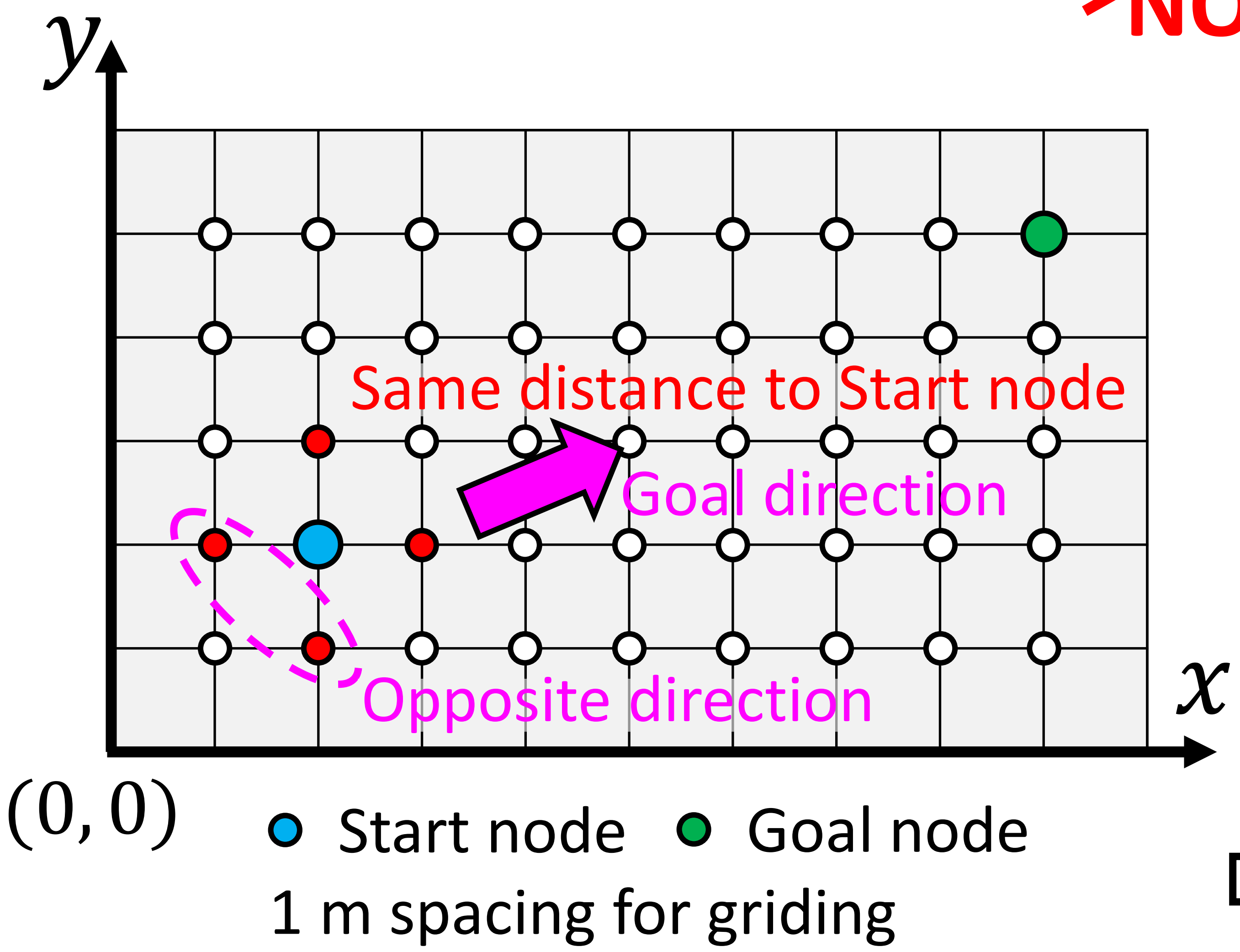
Reach Goal

Retrieve the shortest path (Dijkstra's algorithm)

**Need higher efficiency!**

# Path Planning by Checking All Available Nodes

NOT



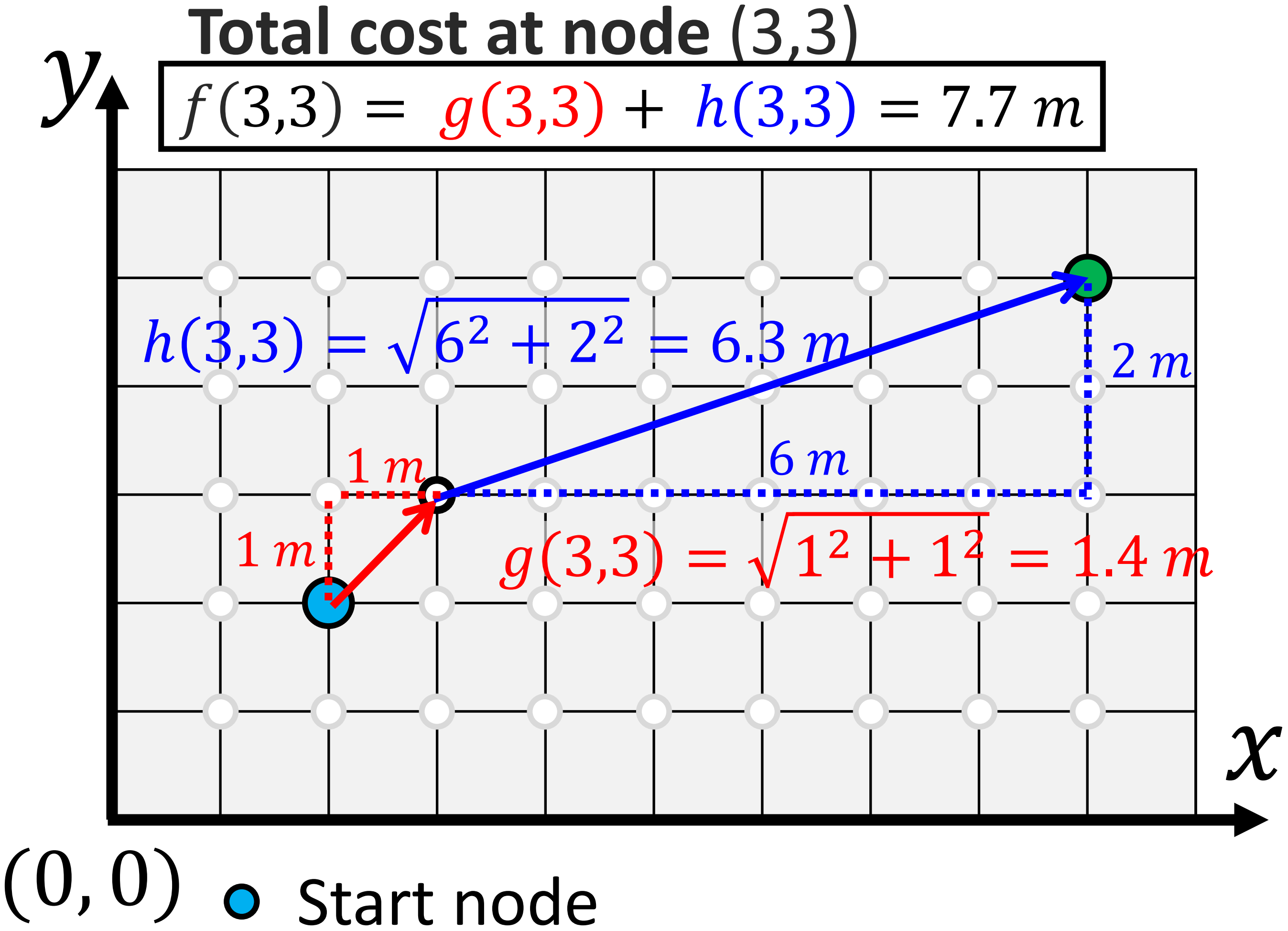

Test each possible nodes one-by-one from Start node

Record its shortest path between Start node

We also know a rough direction to Goal!

**A-star Search Algorithm**

# Path Planning by A-star (A*) Search Algorithm




Start node

Goal node

1 m spacing for griding

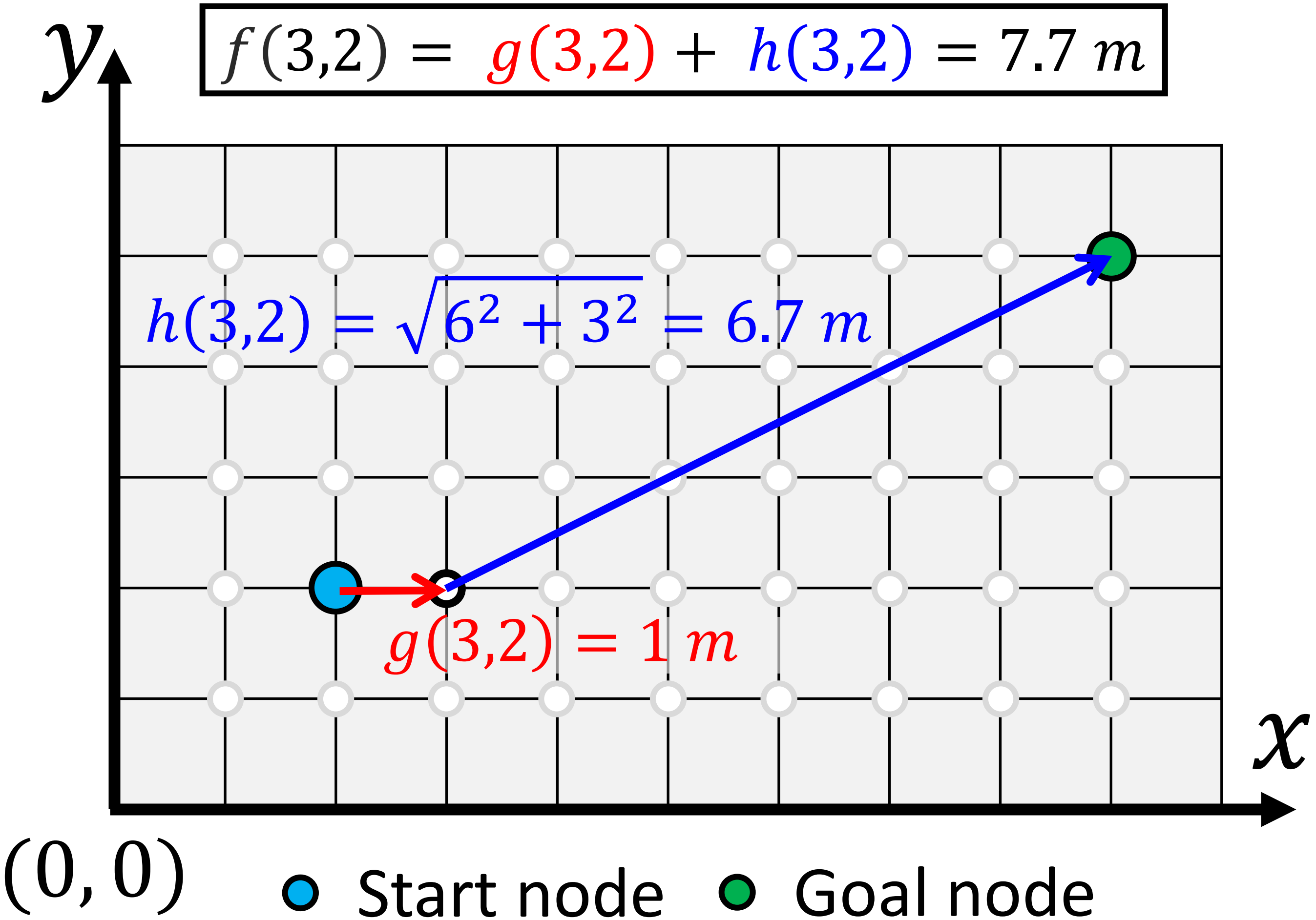
**Definition of cost:**

$g(x, y)$ — this represents the ***exact cost*** of the path **from** the **Start** node to node $(x, y)$

$h(x, y)$ — this represents the **heuristic** ***estimated cost*** from node $(x, y)$ **to** the **Goal** node

$f(x, y) = g(x, y) + h(x, y)$ — **total** cost of a neighboring node $(x, y)$

# A-star (A*) Path Planning – Cost at Node (3,2)




Start node   Goal node

1 m spacing for griding

**Definition of cost:**

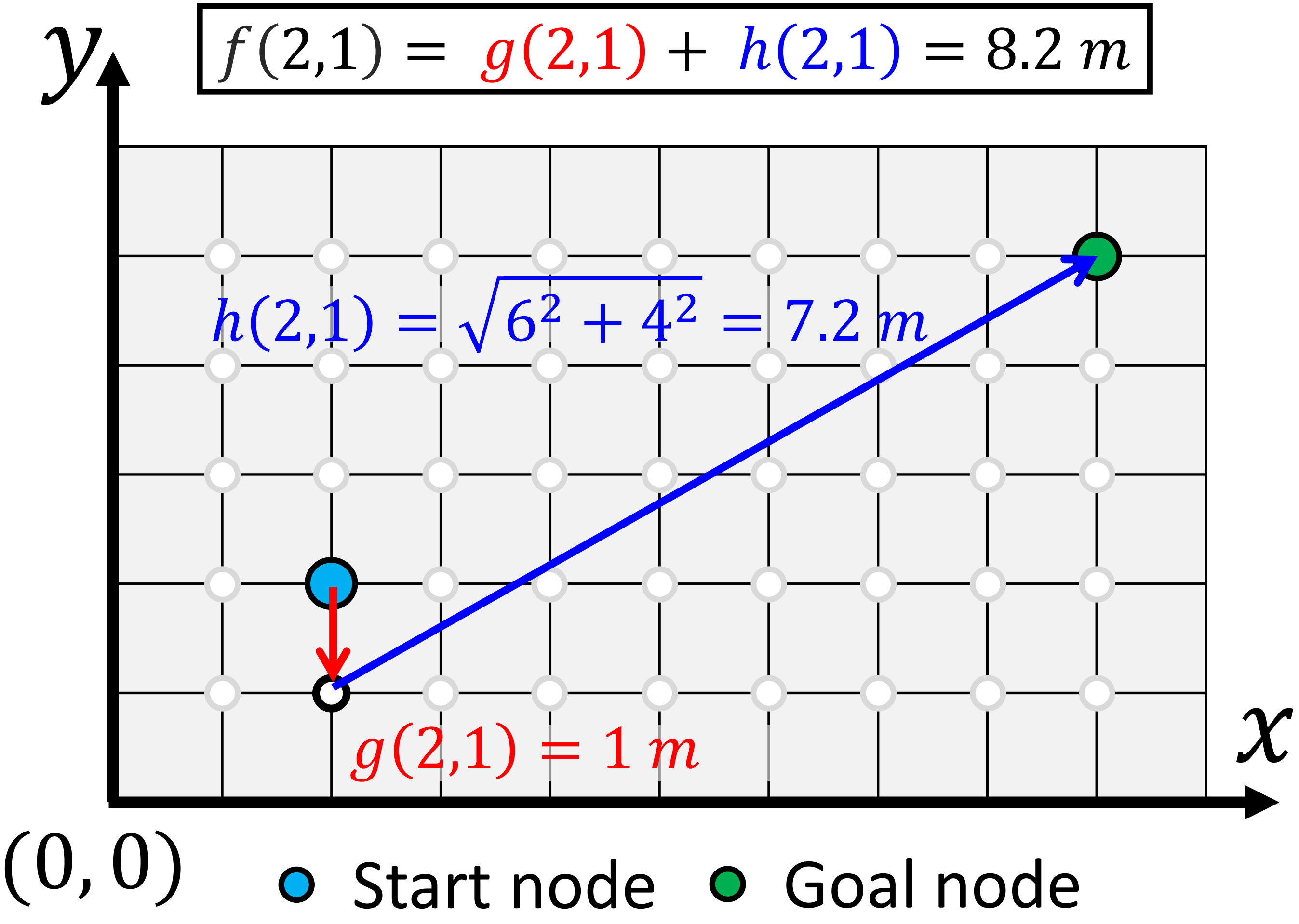
$g(x, y)$ — this represents the ***exact cost*** of the path **from** the **Start** node to node $(x, y)$

$h(x, y)$ — this represents the heuristic ***estimated cost*** from node $(x, y)$ **to** the **Goal** node

$f(x, y) = g(x, y) + h(x, y)$ — **total** cost of a neighboring node $(x, y)$

# A-star (A*) Path Planning – Cost at Node (2,1)

$$f(2,1) = g(2,1) + h(2,1) = 8.2\ m$$




1 m spacing for griding

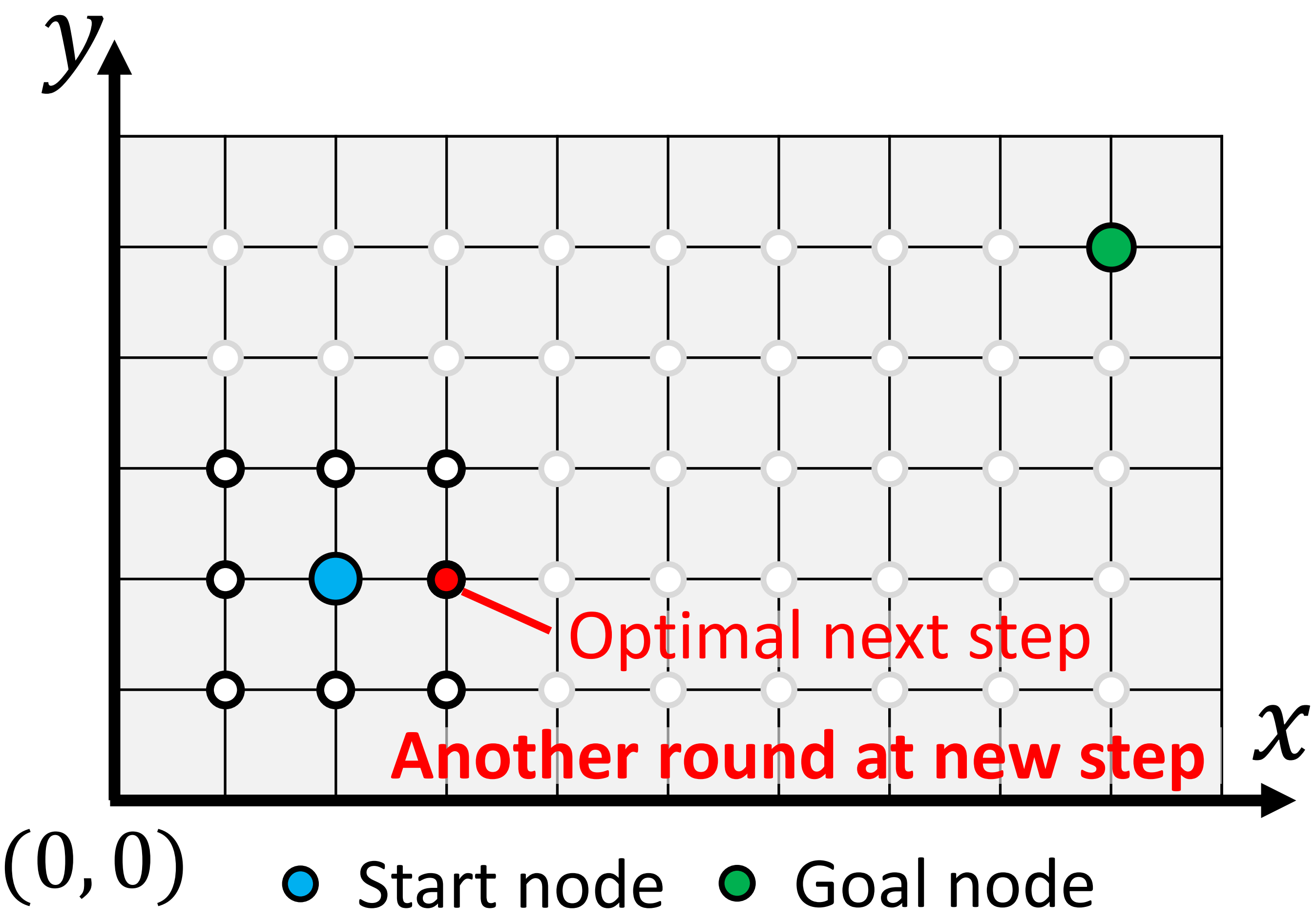
**Definition of cost:**

$g(x, y)$ — this represents the ***exact cost*** of the path **from** the **Start** node to node $(x, y)$

$h(x, y)$ — this represents the heuristic ***estimated cost*** from node $(x, y)$ **to** the **Goal** node

$f(x, y) = g(x, y) + h(x, y)$ — **total** cost of a neighboring node $(x, y)$

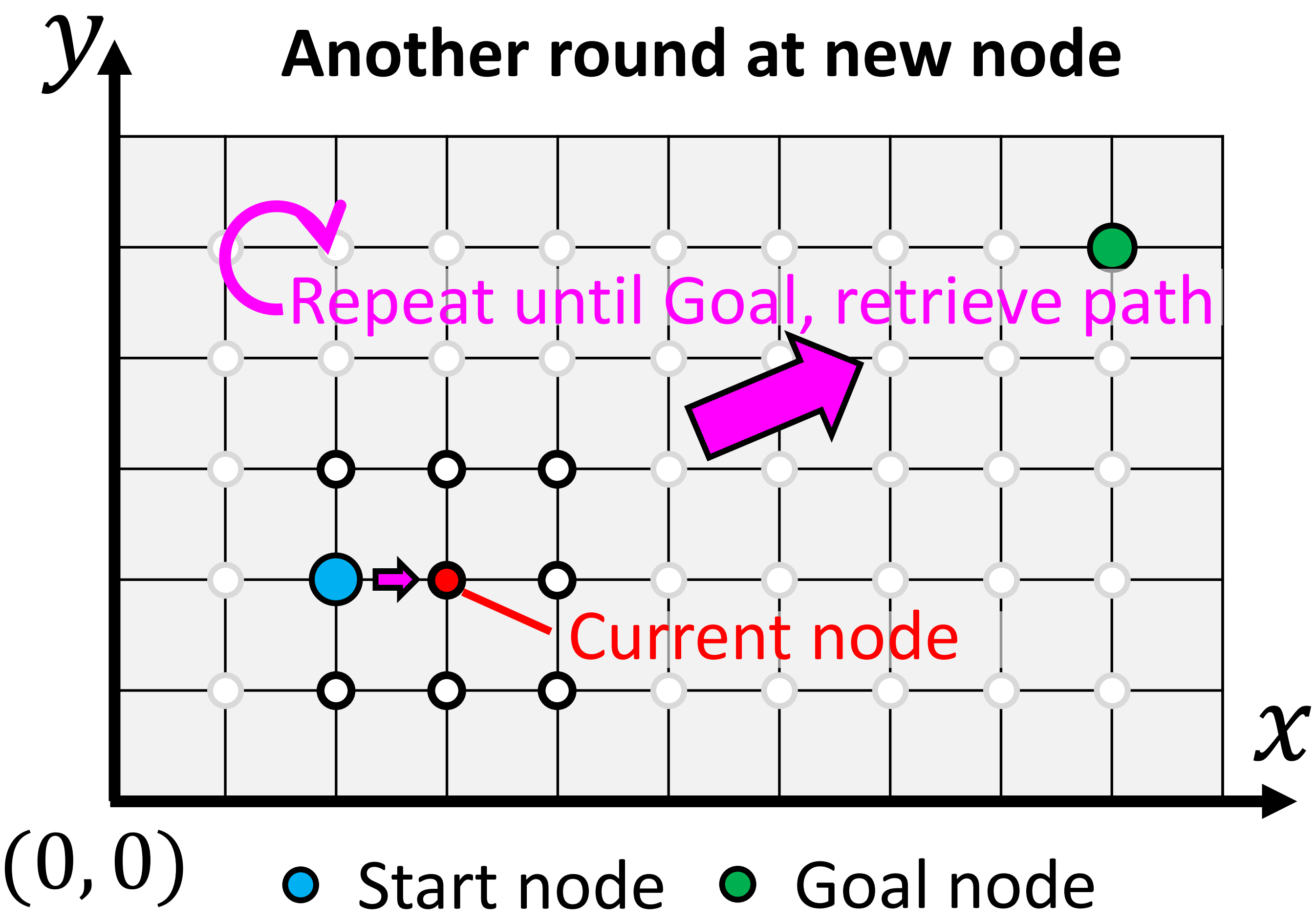
# Total Costs at Neighbouring Nodes




| Node ID | $g(x,y)$ Exact cost from Start | $h(x,y)$ Estimated cost to Goal | $f(x,y)$ Total cost |
|---|---|---|---|
| (1,1) | 1.4 | 8.9 | 10.3 |
| (1,2) | 1 | 8.5 | 9.5 |
| (1,3) | 1.4 | 8.2 | 9.6 |
| (2,1) | 1 | 8.1 | 9.1 |
| (2,3) | 1 | 7.3 | 8.3 |
| (3,1) | 1.4 | 7.2 | 8.6 |
| (3,2) | 1 | 6.7 | 7.7 |
| (3,3) | 1.4 | 6.3 | 7.7 |

# Total Costs at Neighbouring Nodes – Round 2

**Another round at new node**

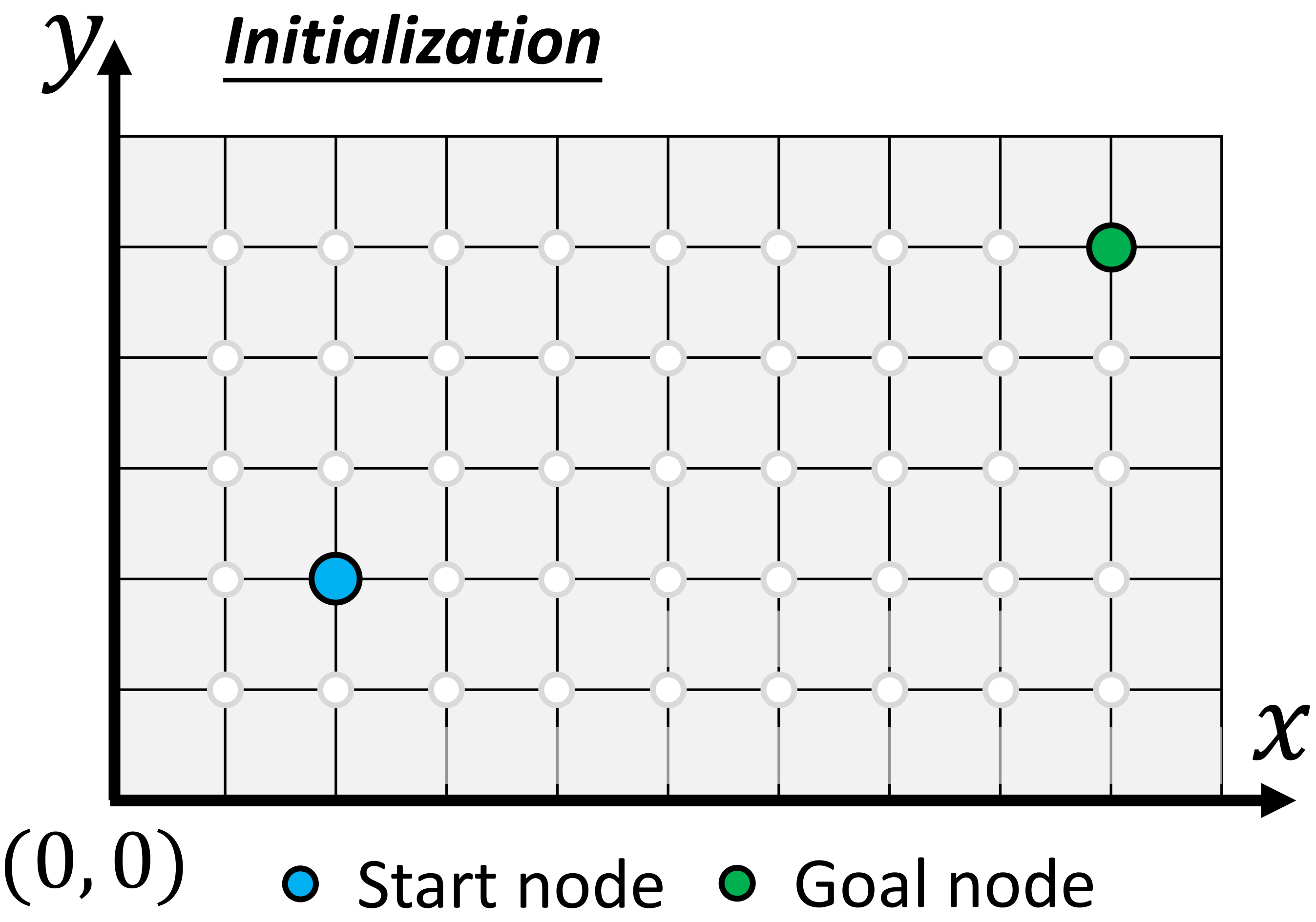



*Exact cost from Start* | *Estimated cost to Goal* | *Total cost*

| Node ID | $g(x,y)$ | $h(x,y)$ | $f(x,y)$ |
|---|---|---|---|
| (2,1) | | | |
| (2,2) | | | |
| (2,3) | | ? | |
| (3,1) | | | |
| (3,3) | | | |
| (4,1) | | | |
| (4,2) | | | |
| (4,3) | | | |

**How does computer search step by step and got path?**

# A-star (A*) Path Planning – Record Path



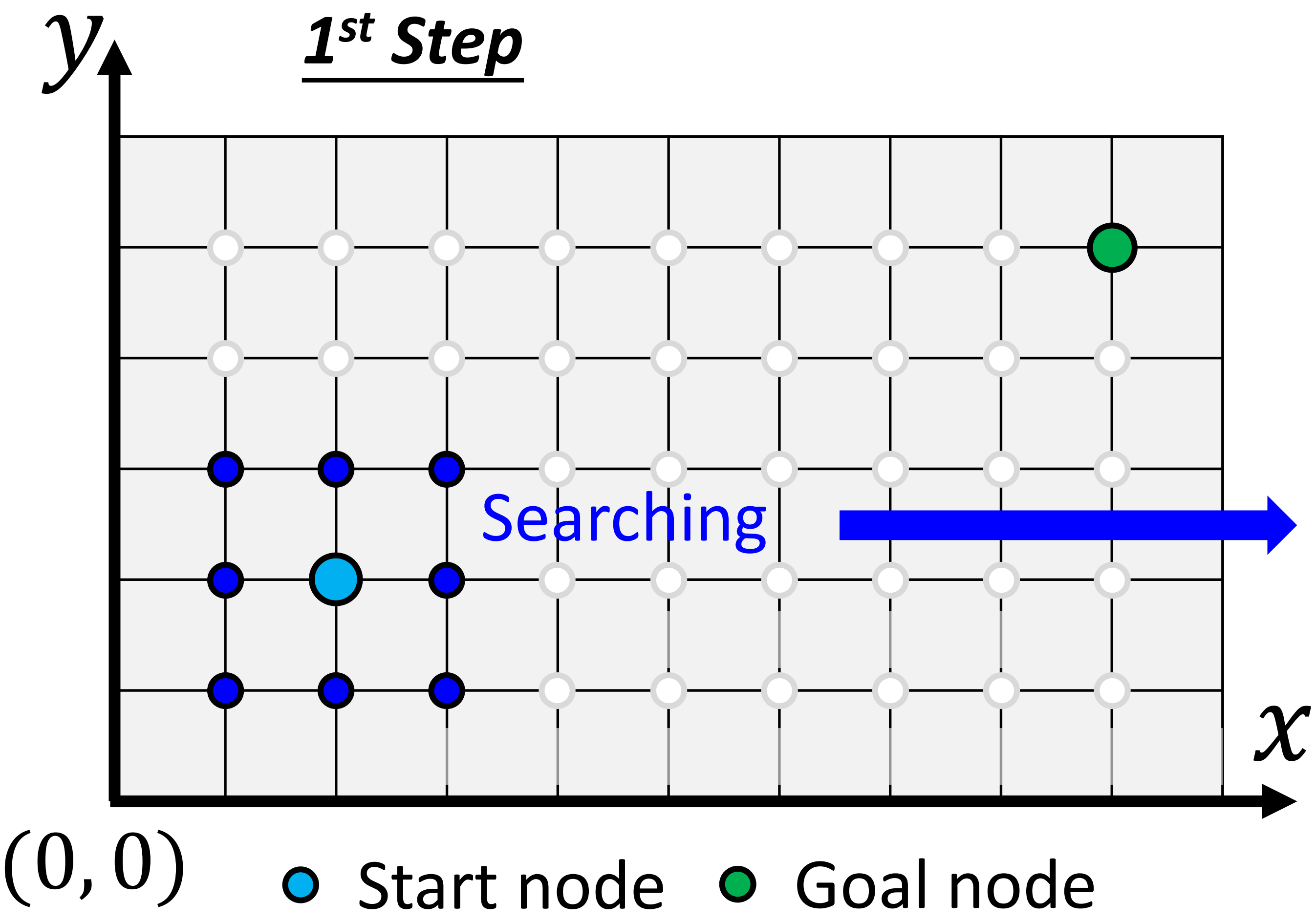

**Open List**
*(searched nodes)*

| Node | $f$ | Source |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**Close List**
*(arrived nodes)*

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# A-star (A*) Path Planning – Record Path



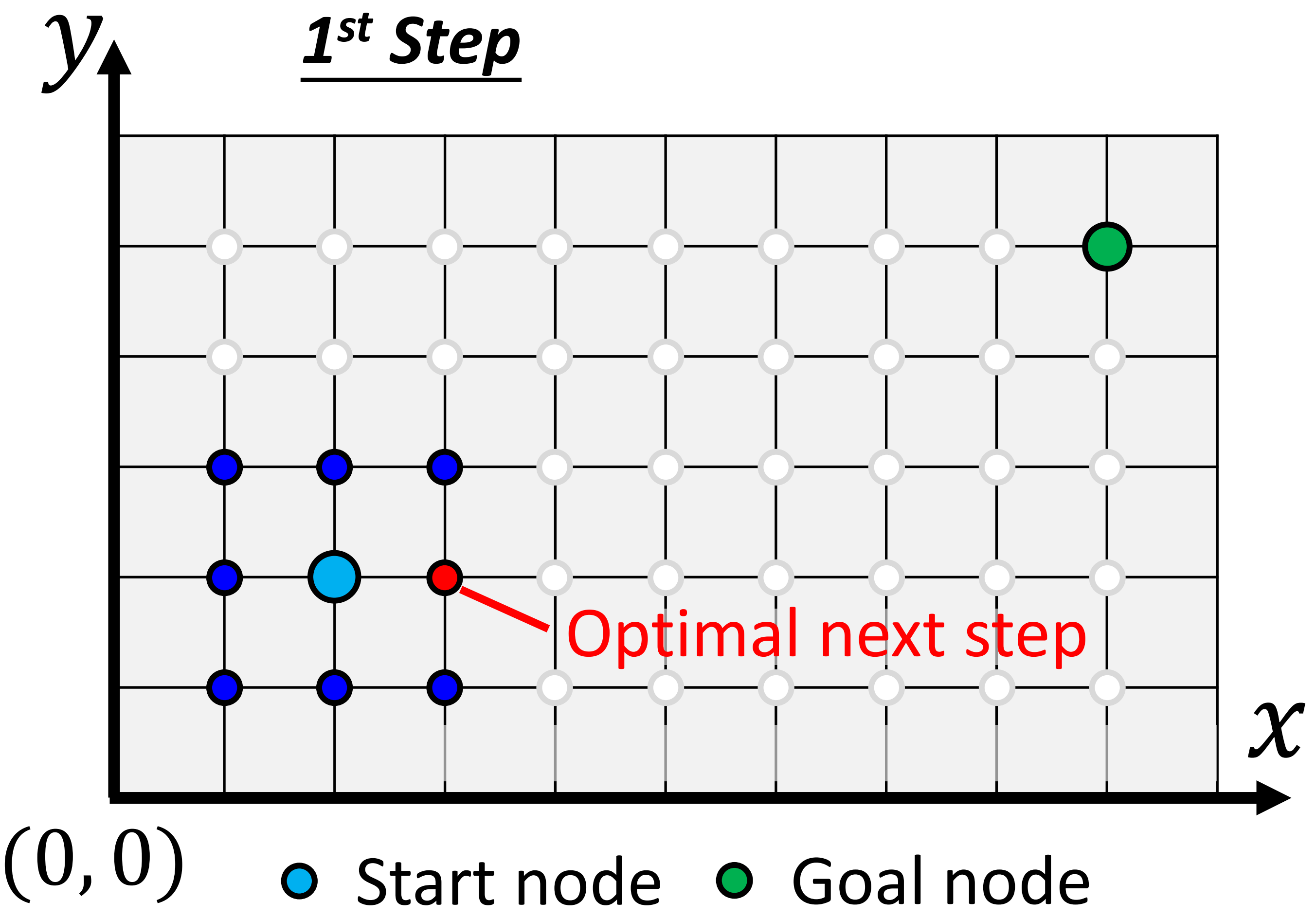

**Open List**
*(searched nodes)*

| Node | $f$ | Source |
|---|---|---|
| (1,1) | 8.9 | (2,2) |
| (1,2) | 8.5 | (2,2) |
| (1,3) | 8.2 | (2,2) |
| (2,1) | 8.1 | (2,2) |
| (2,3) | 7.3 | (2,2) |
| (3,1) | 7.2 | (2,2) |
| (3,2) | 6.7 | (2,2) |
| (3,3) | 6.3 | (2,2) |

**Close List**
*(arrived nodes)*

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# A-star (A*) Path Planning – Record Path



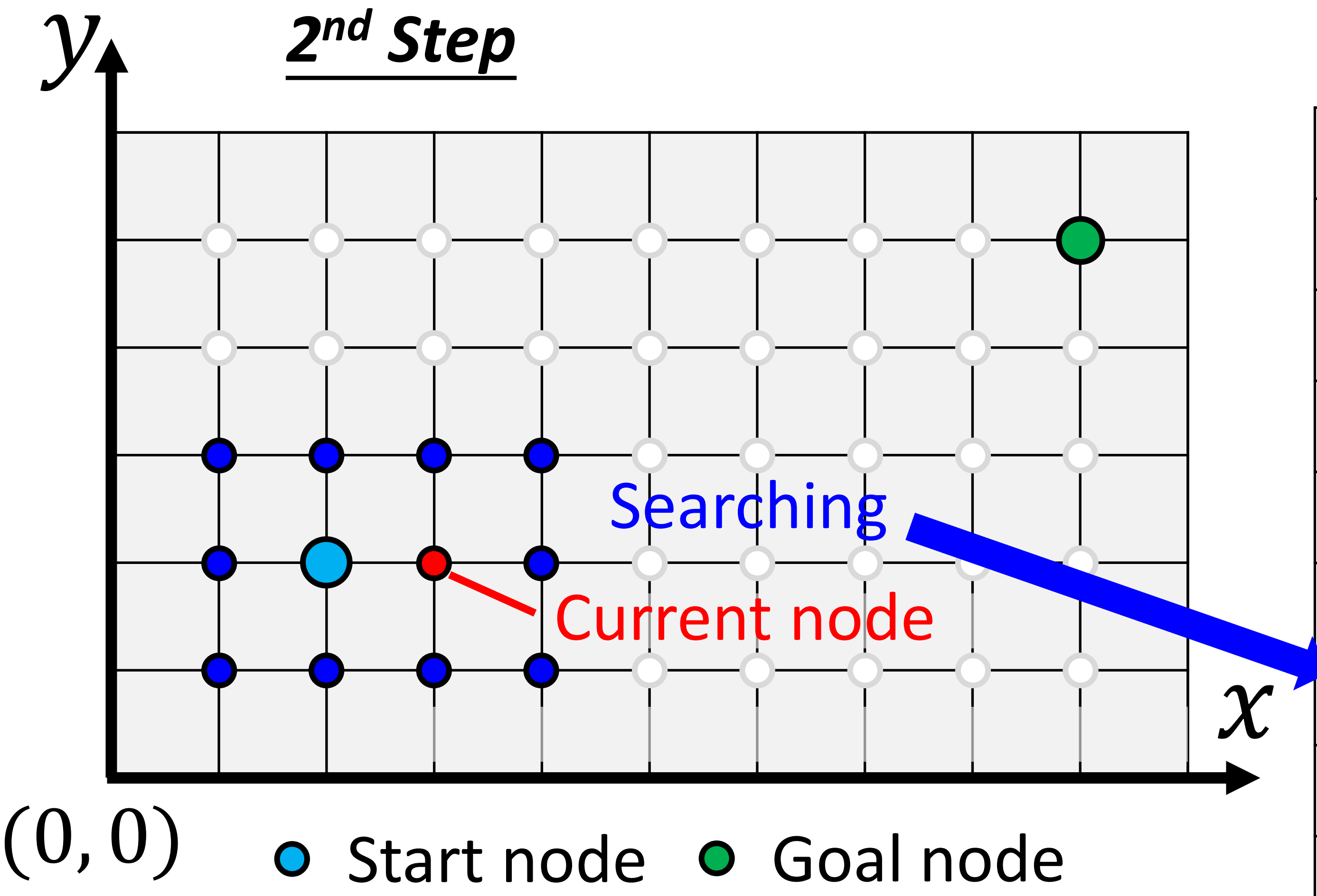

**Open List**
*(searched nodes)*

| Node | $f$ | Source |
|---|---|---|
| (1,1) | 8.9 | (2,2) |
| (1,2) | 8.5 | (2,2) |
| (1,3) | 8.2 | (2,2) |
| (2,1) | 8.1 | (2,2) |
| (2,3) | 7.3 | (2,2) |
| (3,1) | 7.2 | (2,2) |
| ~~(3,2)~~ | ~~6.7~~ | ~~(2,2)~~ |
| (3,3) | 6.3 | (2,2) |

**Close List**
*(arrived nodes)*

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| (3,2) | 6.7 | (2,2) |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# A-star (A*) Path Planning – Record Path



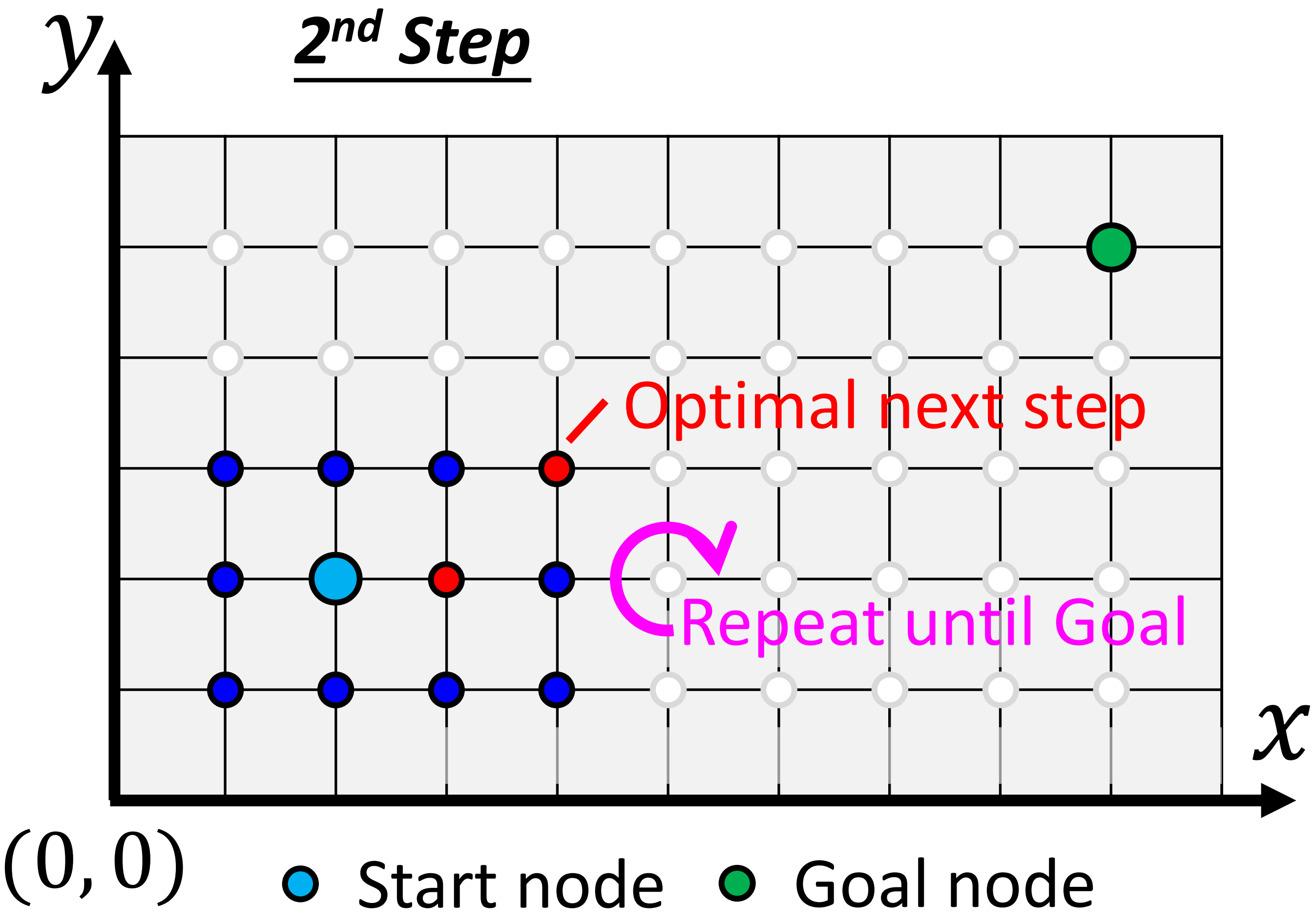

**Open List**
*(searched nodes)*

| Node | $f$ | Source |
|---|---|---|
| (1,1) | 8.9 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| (3,1) | 7.2 | (2,2) |
| (3,3) | 6.3 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| (4,1) | 7.8 | (2,2) |
| (4,2) | 7.8 | (2,2) |
| (4,3) | 8.8 | (2,2) |

**Close List**
*(arrived nodes)*

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| (3,2) | 6.7 | (2,2) |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# A-star (A*) Path Planning – Record Path



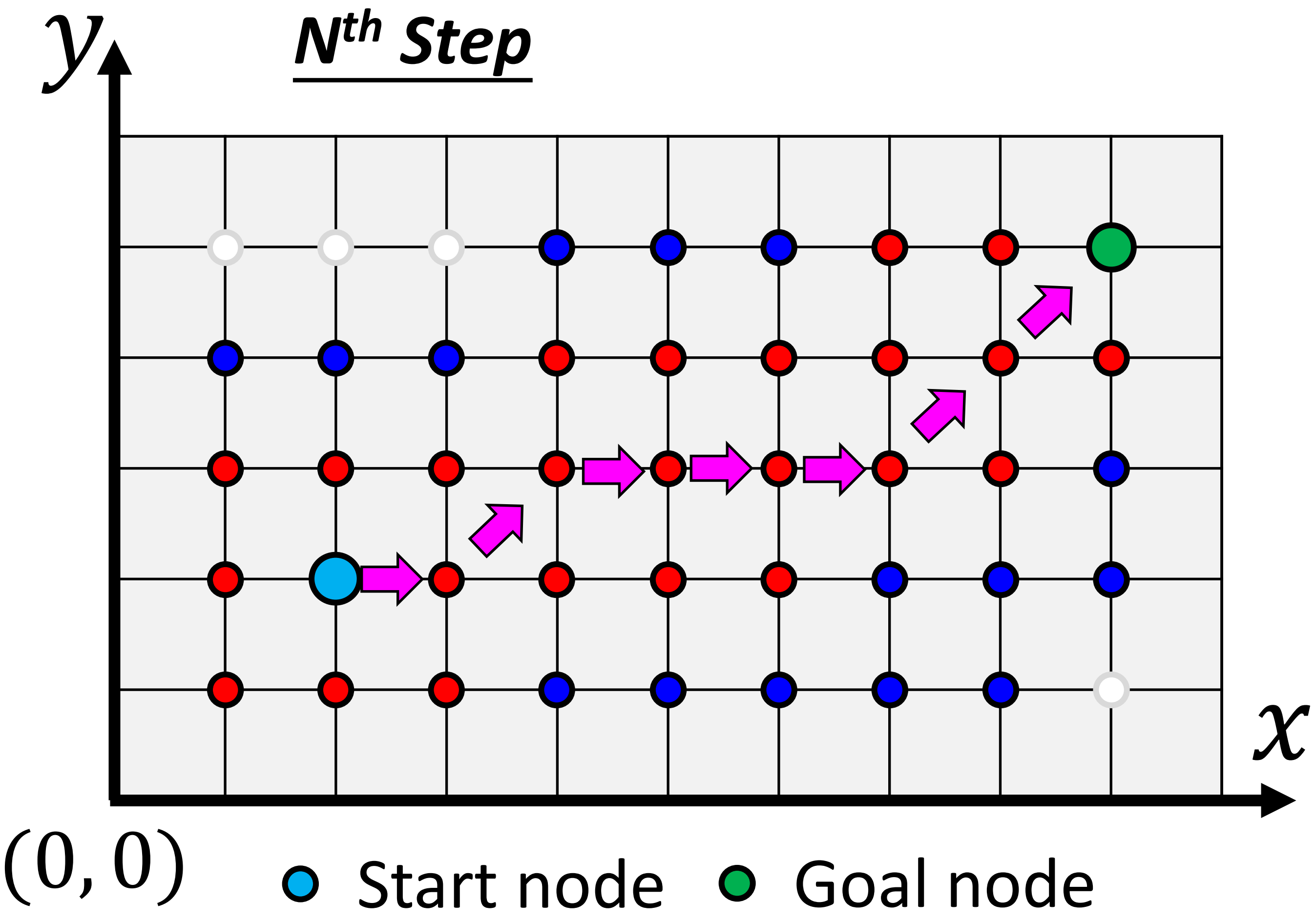

**Open List**
*(searched nodes)*

| Node | $f$ | Source |
|---|---|---|
| (1,1) | 8.9 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| (3,1) | 7.2 | (2,2) |
| (3,3) | 6.3 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| (4,1) | 8.8 | (3,2) |
| (4,2) | 7.8 | (3,2) |
| ~~(4,3)~~ | ~~7.8~~ | ~~(3,2)~~ |

**Close List**
*(arrived nodes)*

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| (3,2) | 6.7 | (2,2) |
| (4,3) | 7.8 | (3,2) |
| | | |
| | | |
| | | |
| | | |
| | | |

# A-star (A*) Path Planning – Retrieve Best Path




**Open List**
(searched nodes)

| Node | $f$ | Source |
|---|---|---|
| (1,1) | 8.9 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| (3,1) | 7.2 | (2,2) |
| (3,3) | 6.3 | (2,2) |
| ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ |
| ~~Goal~~ | ~~x~~ | ~~(x,y)~~ |

**Close List**
(arrived nodes)

| Node | $f$ | Source |
|---|---|---|
| Start | - | - |
| (3,2) | 6.7 | (2,2) |
| (4,3) | 7.8 | (3,2) |
| ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ |
| Goal | x | (x,y) |

Trace back arrived nodes

# A-star Example

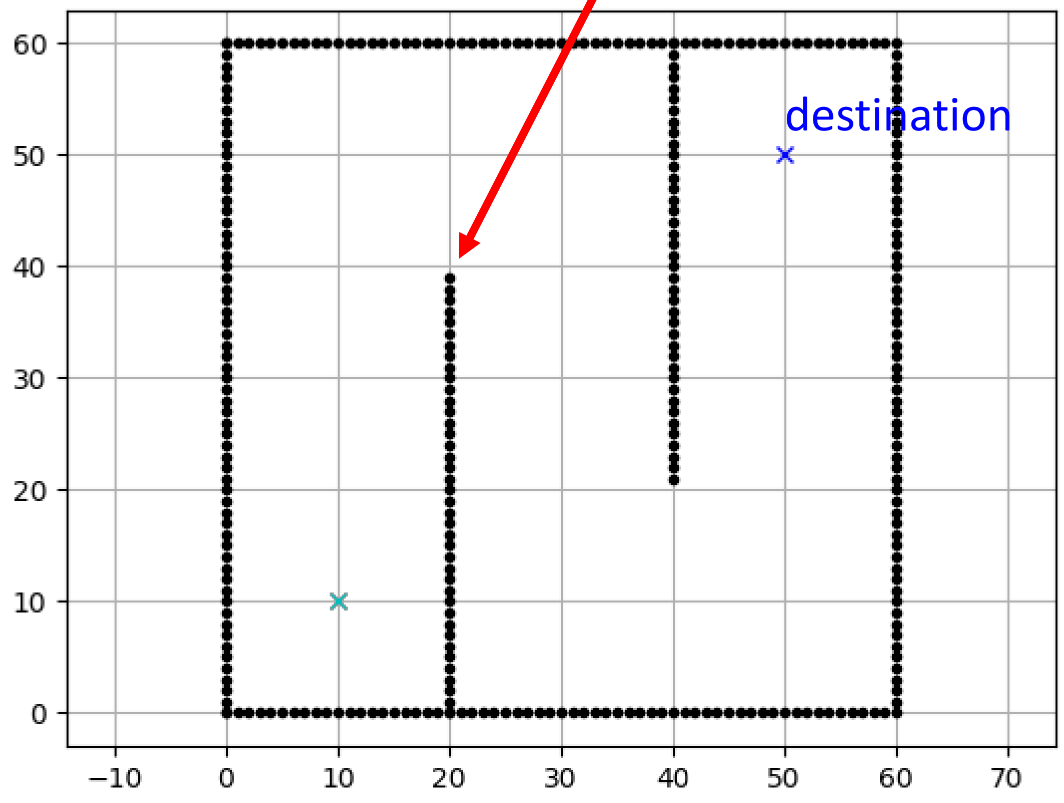
Each time A* enters a node, it calculates the cost: f(n) - n being the neighboring node

It travel to all of the neighboring nodes, and then enters the node with the lowest value of f(n)

These values we calculate using the following formula:

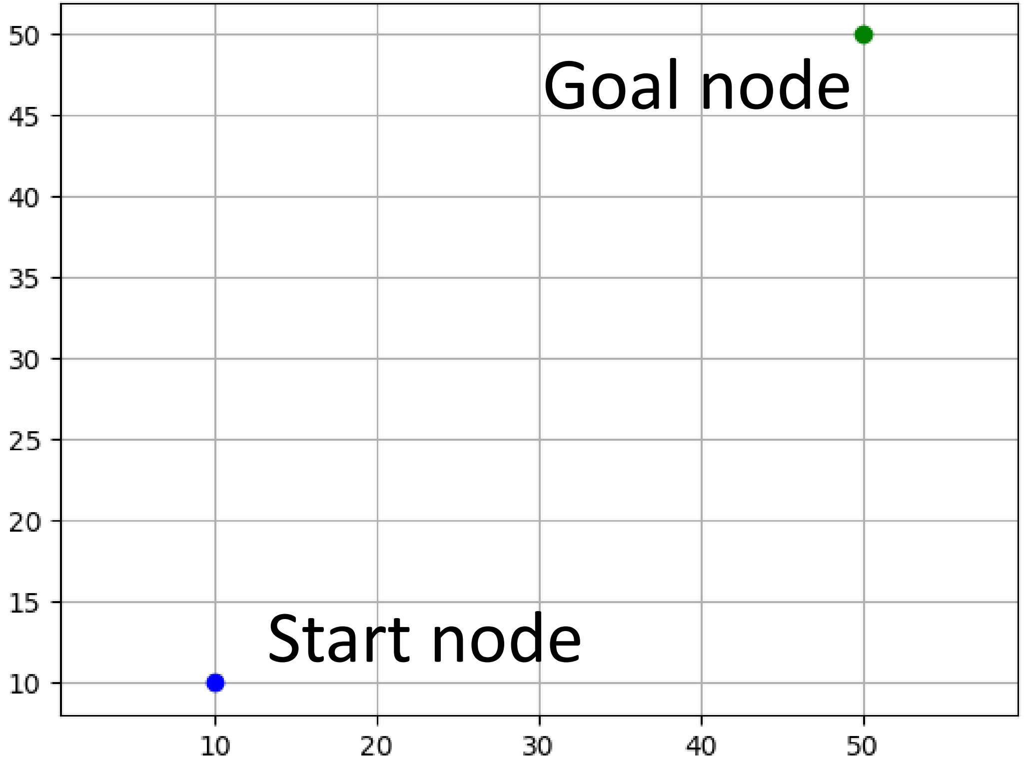
$$f(x,y) = g(x,y) + h(x,y)$$

Wall (obstacles) cannot go through!




https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb *Source: PythonRobotics*

# Code Example (1): Set Up Start and Goal Node

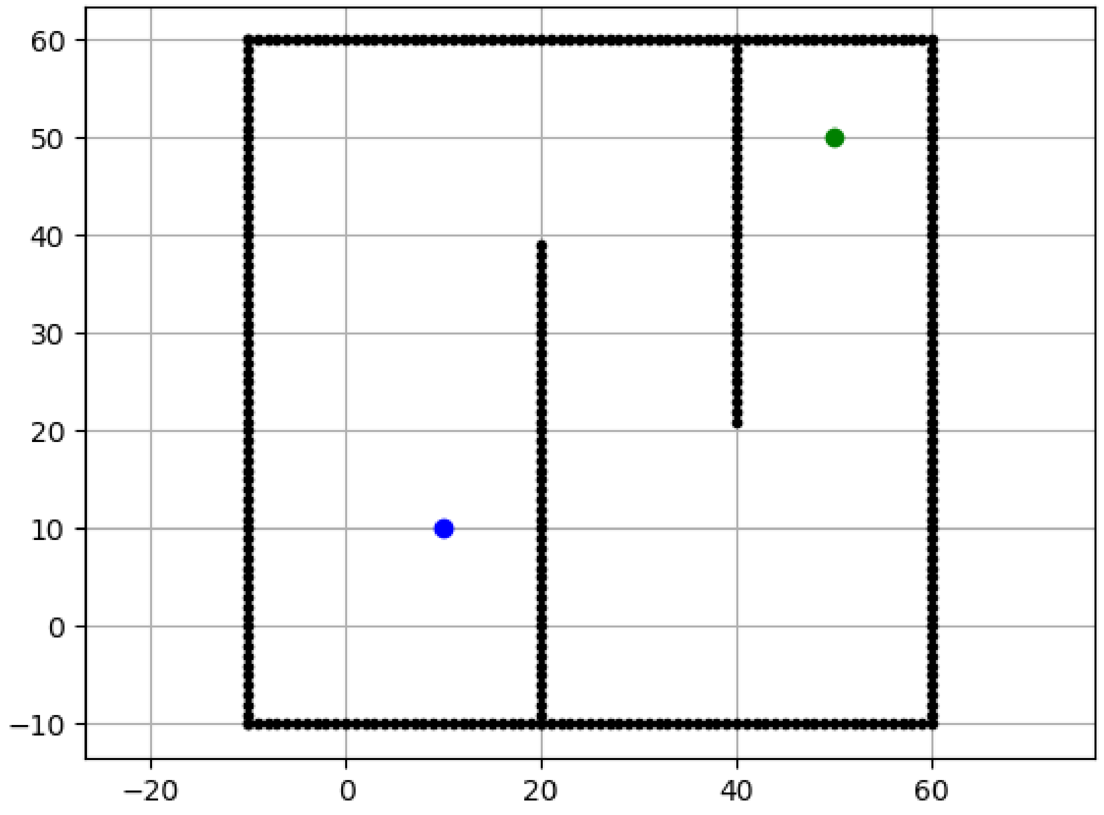



● Start node ● Goal node

```
# start and goal position
sx = 10.0  # [m]
sy = 10.0  # [m]
gx = 50.0  # [m]
gy = 50.0  # [m]
grid_size = 2  # [m]
```

Base code tutorial:
https://www.youtube.com/watch?v=PRKLhcG2kB0&ab_channel=POLYUIPNL

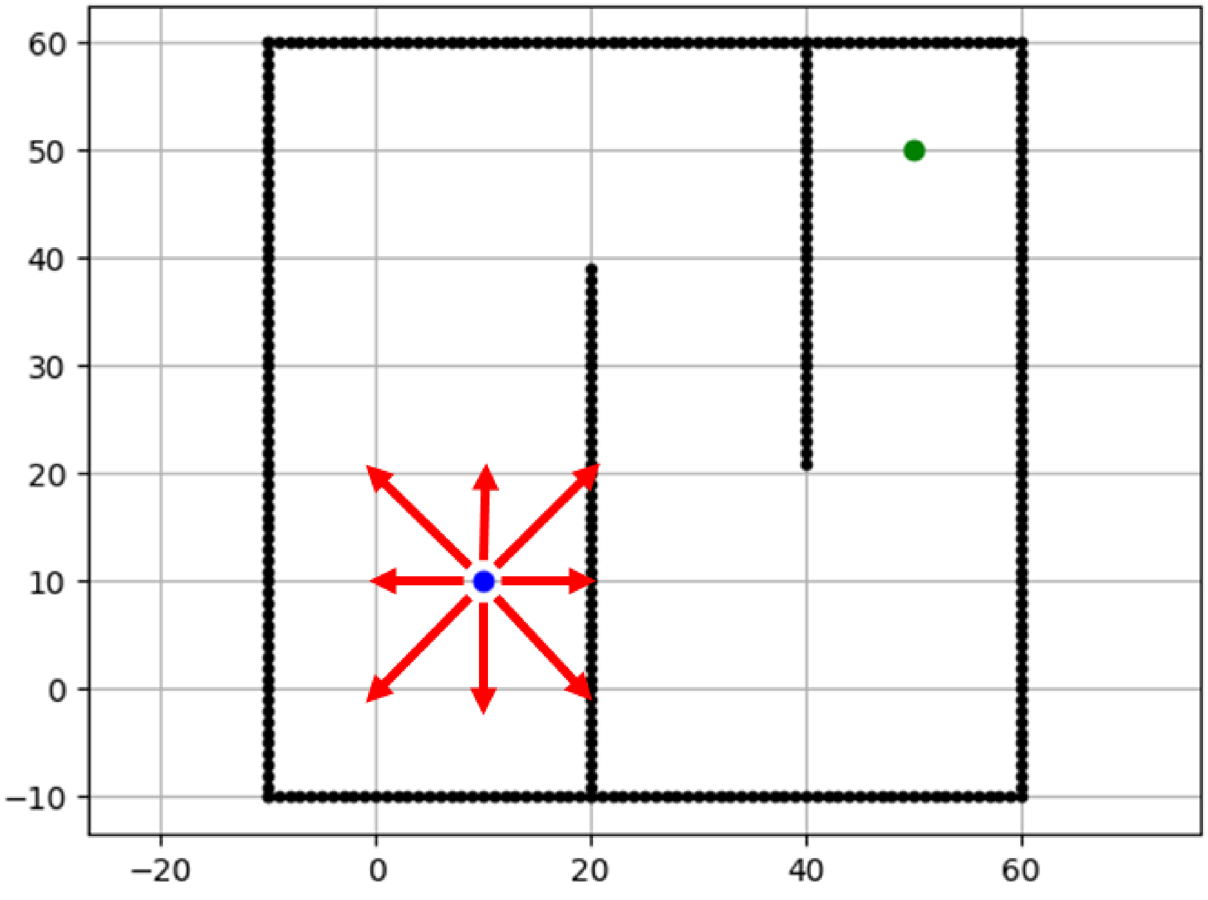
# Code Example (2): Set Up Obstacle




```python
# set obstacle positions
ox, oy = [], []
for i in range(-10, 60): # draw the button border
    ox.append(i)
    oy.append(-10.0)
for i in range(-10, 60):
    ox.append(60.0)
    oy.append(i)
for i in range(-10, 61):
    ox.append(i)
    oy.append(60.0)
for i in range(-10, 61):
    ox.append(-10.0)
    oy.append(i)
for i in range(-10, 40):
    ox.append(20.0)
    oy.append(i)
for i in range(0, 40):
    ox.append(40.0)
    oy.append(60.0 - i)
```

● Start node ● Goal node Obstacle (wall)

# Code Example (3): Neighboring Node Search




```
def get_neighbouring_node(): # the cost of the surrounding 8 points
    # dx, dy, cost
    motion = [[1, 0, 1],
              [0, 1, 1],
              [-1, 0, 1],
              [0, -1, 1],
              [-1, -1, math.sqrt(2)],
              [-1, 1, math.sqrt(2)],
              [1, -1, math.sqrt(2)],
              [1, 1, math.sqrt(2)]]

    return motion
```

Start node   Goal node   Obstacle (wall)

# Code Example (4): Cost Calculation

Exact cost $g(x, y)$ calculation

```
node = self.Node(current.x + self.motion[i][0],
                 current.y + self.motion[i][1],
                 current.cost + self.motion[i][2], c_id)
```

Heuristic cost $h(x, y)$ calculation

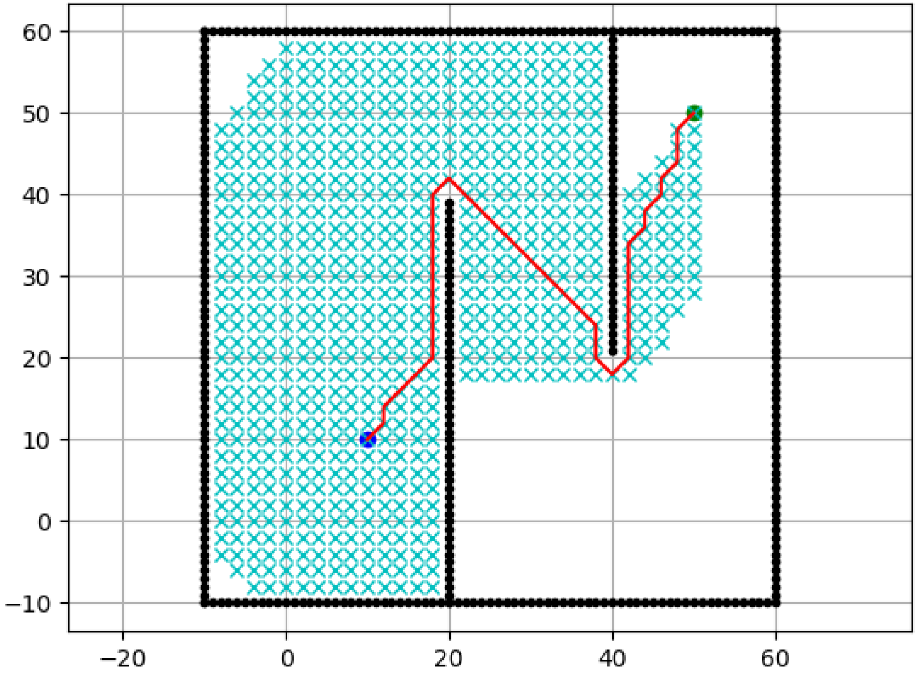
```
def calc_heuristic(n1, n2):
    w = 1.0  # weight of heuristic
    d = w * math.hypot(n1.x - n2.x, n1.y - n2.y)
    return d
```

# Code Example (5): Calculation of Final Path

```
def calc_final_path(self, goal_node, closed_set):
    # generate final course
    rx, ry = [self.calc_grid_position(goal_node.x, self.min_x)], [
        self.calc_grid_position(goal_node.y, self.min_y)] # save the goal node as the first point
    parent_index = goal_node.parent_index
    while parent_index != -1:
        n = closed_set[parent_index]
        rx.append(self.calc_grid_position(n.x, self.min_x))
        ry.append(self.calc_grid_position(n.y, self.min_y))
        parent_index = n.parent_index

    return rx, ry
```

Retrieve the optimal path from all passing through nodes (once being the current node)



# Play with Example Codes ☺

# Trip Cost of Flight

# Trip Cost of Flight

The fundamental rationale of the cost index concept is to achieve minimum **trip cost** by means of a trade-off between operating costs per hour and incremental fuel burn.

**Trip Cost:** $\boxed{C = C_F \cdot \Delta F \cdot T + C_T \cdot T + C_c}$

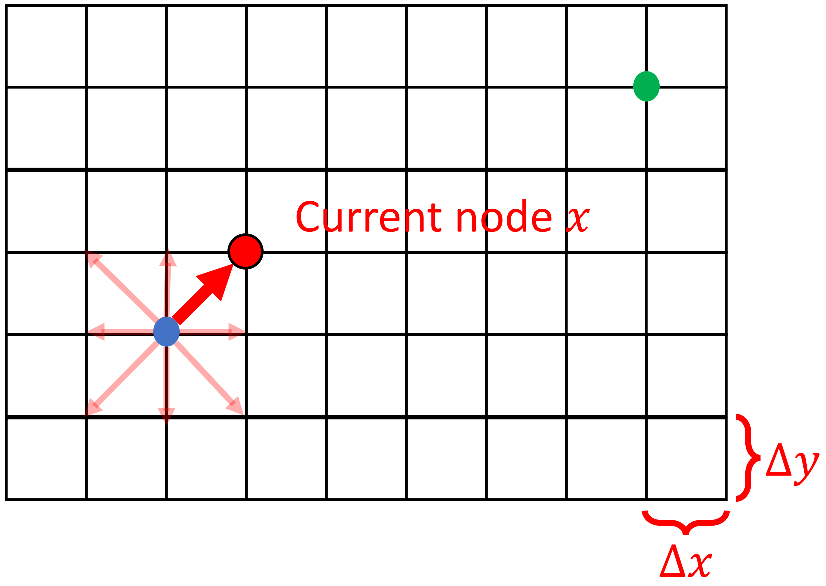
* Related to travelling time

With

- $C_F$=cost of fuel per kg
- $C_T$=time related cost per minute of flight
- $C_c$=fixed cost independent of time
- $\Delta F$=trip fuel (e.g., 3000kg/h)
- $T$=trip Time (e.g., 8 hours from Hong Kong to Paris)

Consider this relationship to imitate the path planning for flights?

# Conventional Path Planning Task




Total cost: $f(x,y) = g(x,y) + h(x,y)$

Distance information (conventional)

*Example for one step:*

$$g(x,y) = \sqrt{\Delta x^2 + \Delta y^2}$$

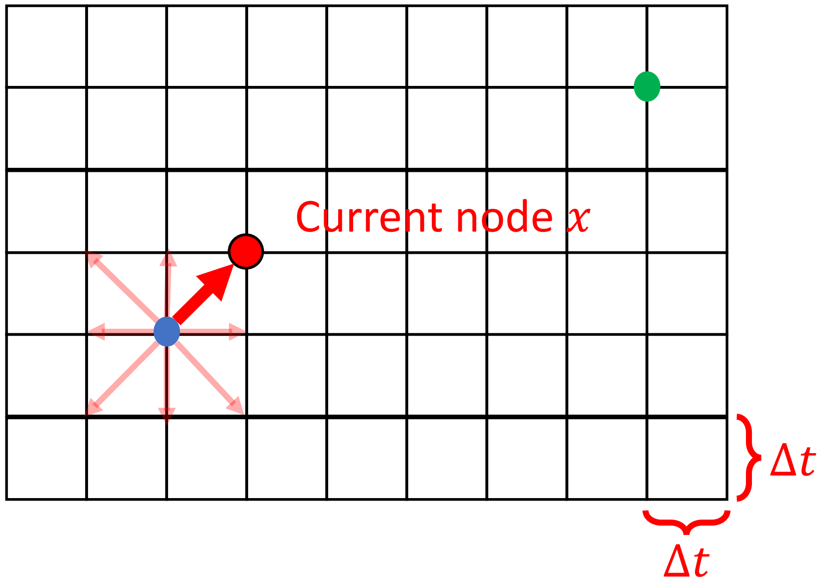
Objective: Find the path with the lowest traveling distance. (in the unit of meter)

Goal node   Start node

# Path Planning Task for Out Flight

Total cost: $f(n) = \underbrace{g(n) + h(n)}$

Traveling time (our case for flight)




Time cost on each step:

$$s(n) = \begin{cases} \Delta t, & vertical/horizontal\ motion \\ \sqrt{2}\Delta t, & diagonal\ motion \end{cases}$$

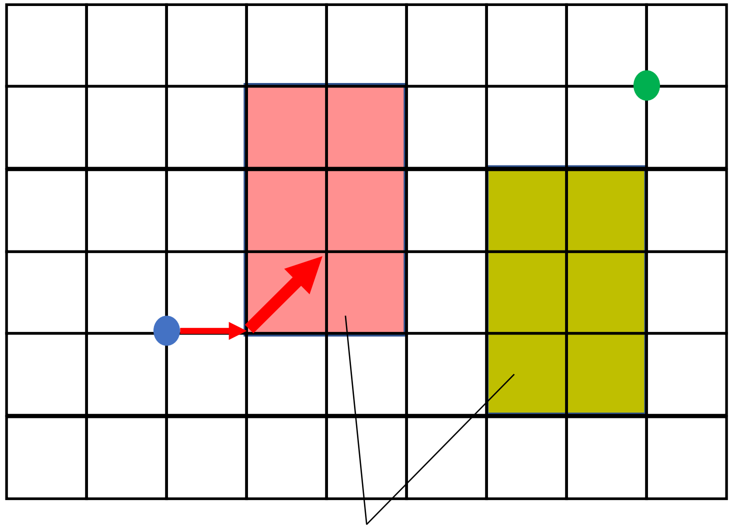
*Example for first step:*

$$g(n) = s(n) = \sqrt{2}\Delta t$$

Objective: Find the path with the lowest traveling time. (in the unit of min)

● Goal node ● Start node $n$ – index of nodes

# Flight Planning Considering Cost Intensive Areas



Cost intensive area
(addition time)

**<u>Cost Intensive Areas</u>**: The cost for flying through such area is **increased** due to airflow, legal restrictions and other reasons.

Total cost for one node:

$$f(n) = \underline{g(n)} + \underline{\alpha \cdot s(n)} + h(n)$$

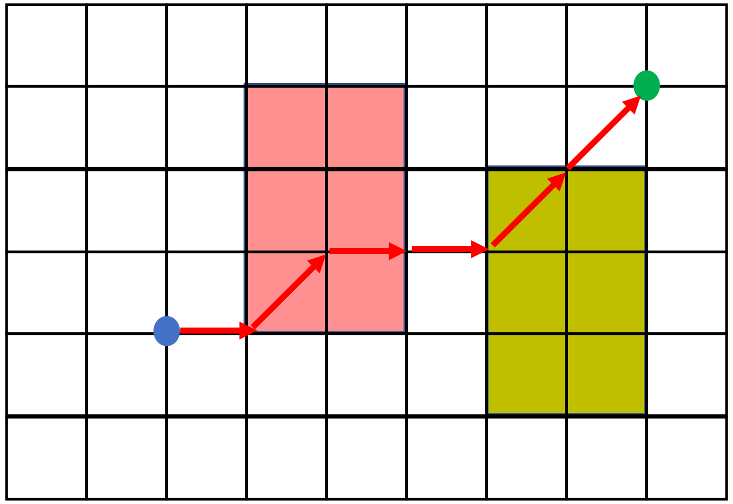
Previous cost from start

<u>Additional time at current step</u>

α – additional time factor (equal zero for normal area)

Goal node Start node

# Flight Path Planning Trip Cost



**<u>Cost Intensive Areas</u>**: The cost for flying through such area is **increased** due to airflow, legal restrictions and other reasons.

Total cost for one node:

$$f(n) = g(n) + \alpha \cdot s(n) + h(n)$$

*Path planning*

$$T_{best} = \min[f(goal)]$$

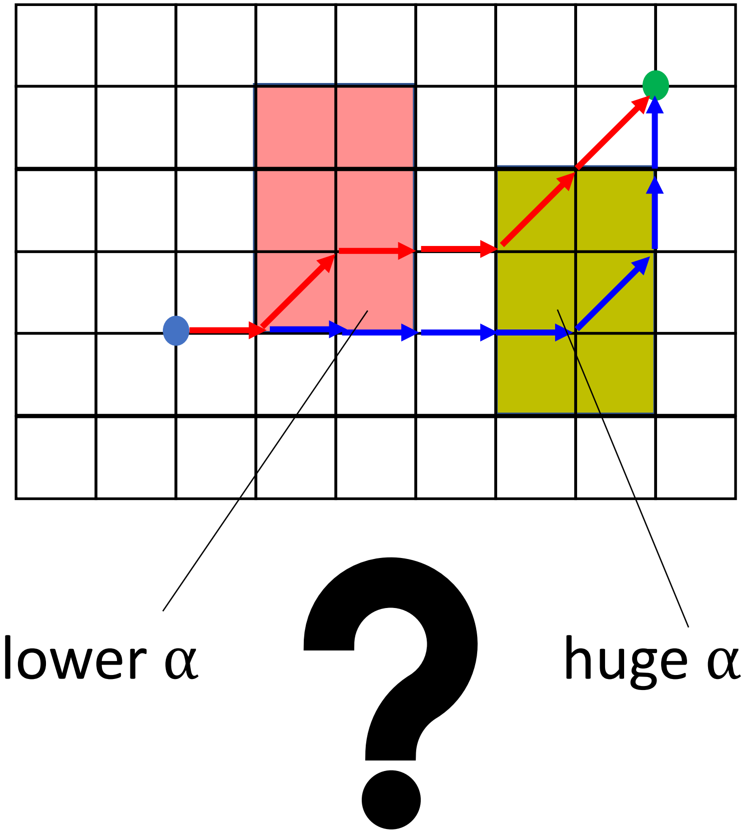
*Additional cost on each node*

$$= \min\left[g(goal) + \alpha_1 s(n_1) + \alpha_2 s(n_2) + \cdots + \alpha_{goal} s\left(n_{goal}\right)\right]$$

**Trip cost for planned path:** $\boxed{C = C_F \cdot \Delta F \cdot T_{best} + C_T \cdot T_{best} + C_c}$

Goal node Start node $\alpha$ – additional time factor

# How it choose different routes?




**<u>Cost Intensive Areas</u>**: The cost for flying through such area is **increased** due to airflow, legal restrictions and other reasons.

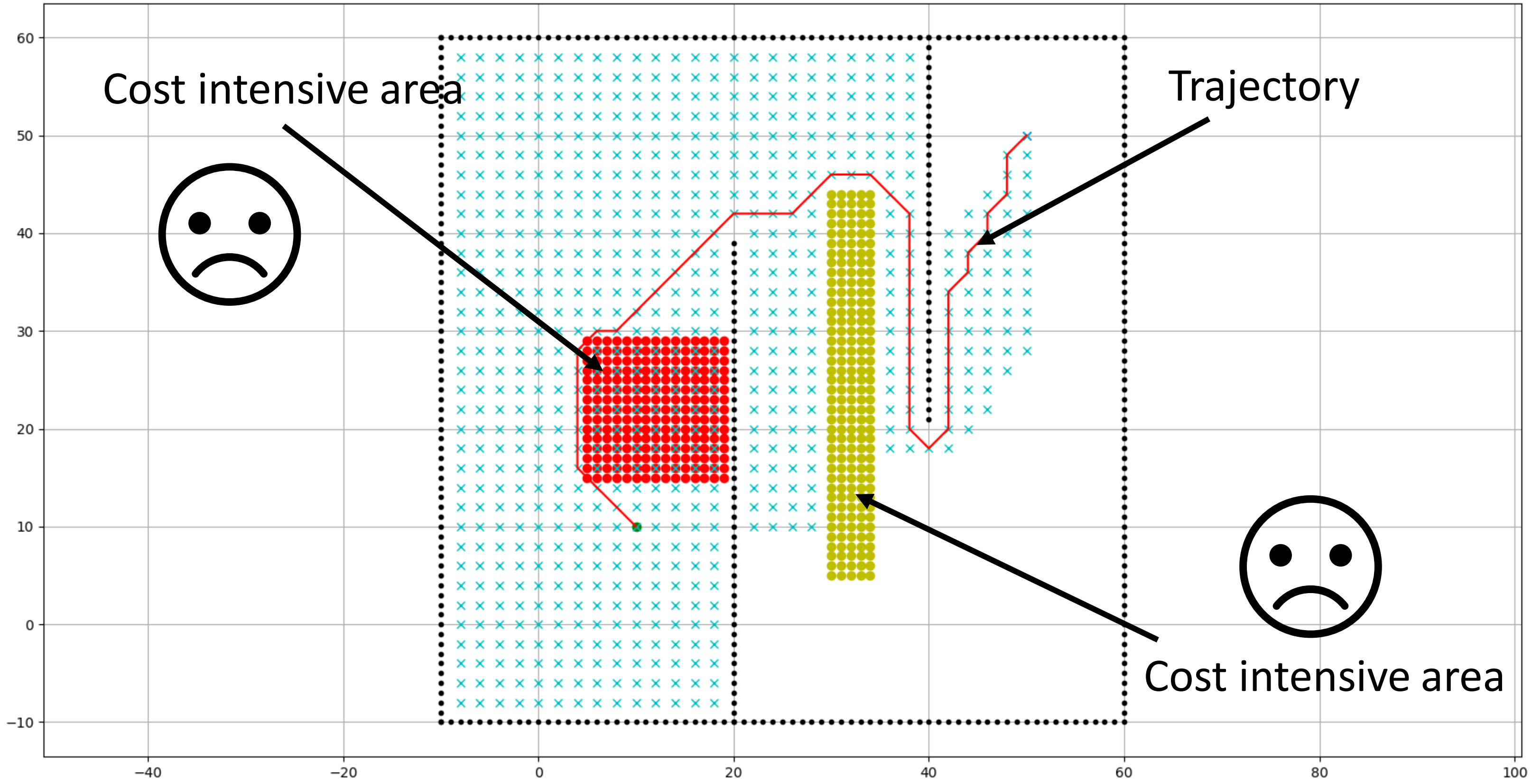
Time for the planned path:

$$T_{best} = \min[g(goal) + \underline{\alpha_1 s(n_1)} + \underline{\alpha_2 s(n_2)} + \cdots + \underline{\alpha_{goal} s(n_{goal})}]$$

**Depending on the extra time accumulated more specifically, the additional time factor**

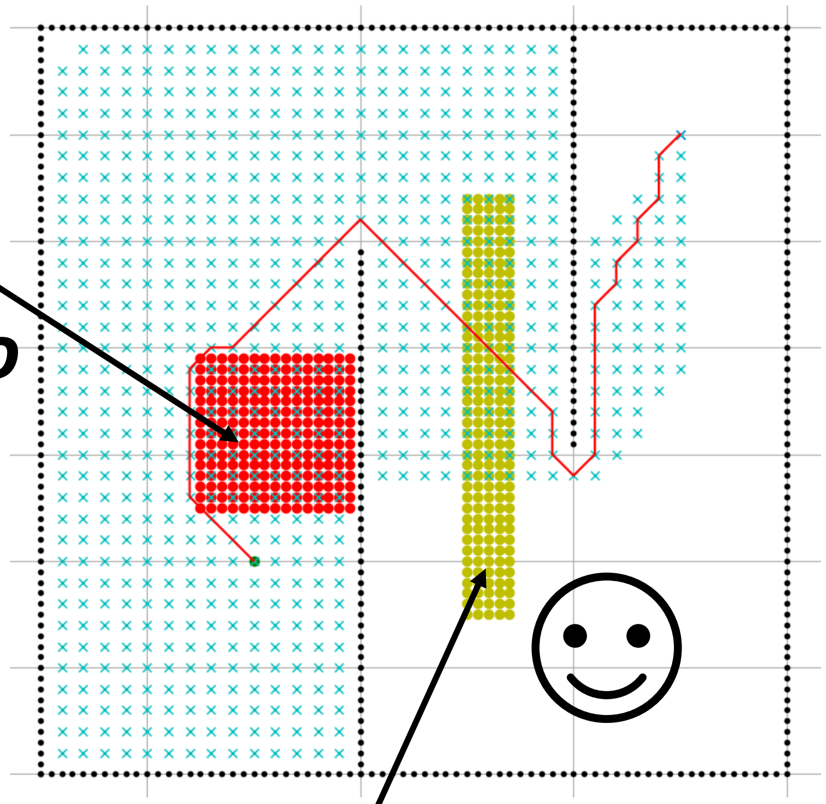
Goal node    Start node    α – additional time factor

# Example of Flight Path Planning



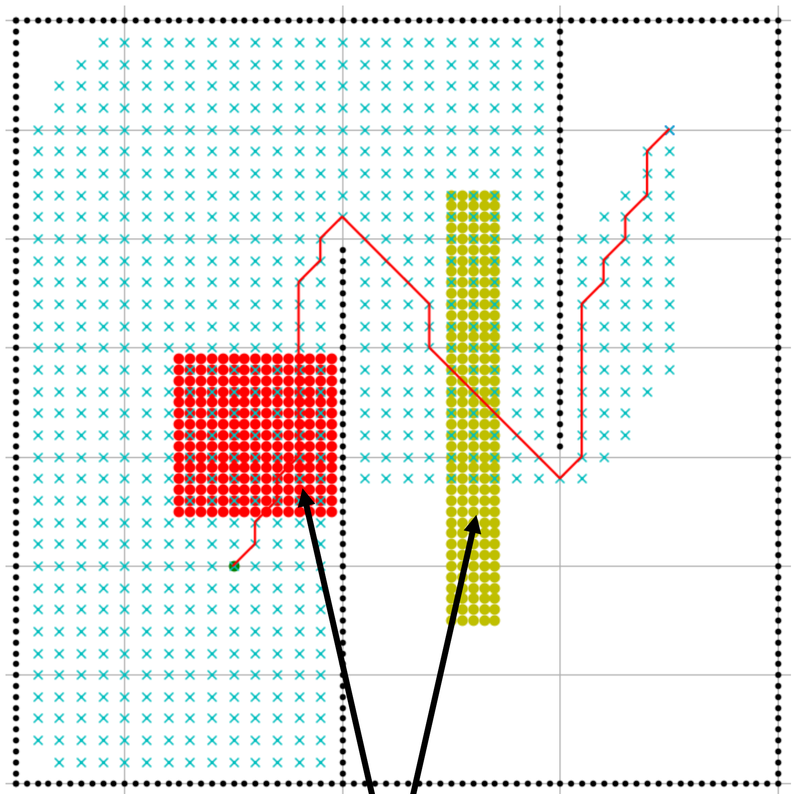

***Cost is way too high for going through, better to avoid!***

# Example of Flight Path Planning



Cost too much

Not too much and acceptable



Both acceptable

# Flight Path Planning Project

You will be creating and completing your own path planning program based on groups

You can find the project tasks / requirements in this slide after the code tutorial

Additional resources could be found inside the course GitHub repository

- Video tutorial
- Tutorial slides

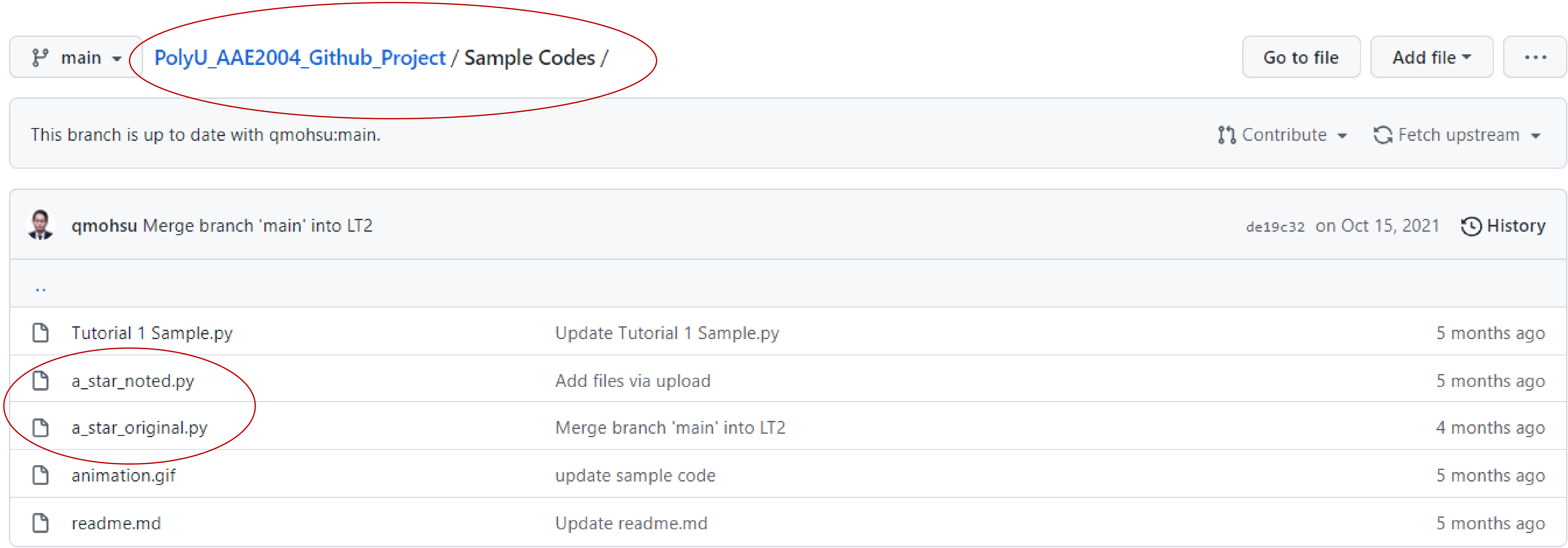
# Path Planning Programming Guide

# The Path Planning Code

- You can find the path planning code inside the course GitHub repository
- There are 2 set of codes:
  - A default one
  - A noted one
- The default one is a basic A* path planning code without any extra information and features
- The noted one provides an example of what your code should look like after modifications (Remember each group should complete a different set of obstacles and requirements)
- Repository link: https://github.com/IPNL-POLYU/PolyU_AAE2004_Github_Project

Base code tutorial: https://www.youtube.com/watch?v=PRKLhcG2kB0&ab_channel=POLYUIPNL

# Where you can find the code



# Noted Version Guide

- Line 50,51: Declaration of cost intensive area cost modifier
- Line 53: Declare cost per grid

```python
34
35         self.resolution = resolution # get resolution of the grid
36         self.rr = rr # robot radis
37         self.min_x, self.min_y = 0, 0
38         self.max_x, self.max_y = 0, 0
39         self.obstacle_map = None
40         self.x_width, self.y_width = 0, 0
41         self.motion = self.get_motion_model() # motion model for grid search expansion
42         self.calc_obstacle_map(ox, oy)
43
44         self.fc_x = fc_x
45         self.fc_y = fc_y
46         self.tc_x = tc_x
47         self.tc_y = tc_y
48
49
50         self.Delta_C1 = 0.2 # cost intensive area 1 modifier
51         self.Delta_C2 = 0.4 # cost intensive area 2 modifier
52
53         self.costPerGrid = 1
54
```

# Noted Version Guide

- Line 115: Showing the final calculation of total trip time
- Line 135-144: Adding additional cost during cost intensive area

```
103         if show_animation:  # pragma: no cover
104             plt.plot(self.calc_grid_position(current.x, self.min_x),
105                      self.calc_grid_position(current.y, self.min_y), "xc")
106             # for stopping simulation with the esc key.
107             plt.gcf().canvas.mpl_connect('key_release_event',
108                                          lambda event: [exit(
109                                              0) if event.key == 'escape' else None])
110             if len(closed_set.keys()) % 10 == 0:
111                 plt.pause(0.001)
112 
113         # reaching goal
114         if current.x == goal_node.x and current.y == goal_node.y:
115             print("Total Trip time required -> ",current.cost )
116             goal_node.parent_index = current.parent_index
117             goal_node.cost = current.cost
118             break
119 
120         # Remove the item from the open set
121         del open_set[c_id]
122 
123         # Add it to the closed set
124         closed_set[c_id] = current
125 
126         # print(len(closed_set))
127 
128         # expand_grid search grid based on motion model
129         for i, _ in enumerate(self.motion): # tranverse the motion matrix
130             node = self.Node(current.x + self.motion[i][0],
131                              current.y + self.motion[i][1],
132                              current.cost + self.motion[i][2] * self.costPerGrid, c_id)
133 
134             ## add more cost in cost intensive area 1
135             if self.calc_grid_position(node.x, self.min_x) in self.tc_x:
136                 if self.calc_grid_position(node.y, self.min_y) in self.tc_y:
137                     # print("cost intensive area!!")
138                     node.cost = node.cost + self.Delta_C1 * self.motion[i][2]
139 
140             # add more cost in cost intensive area 2
141             if self.calc_grid_position(node.x, self.min_x) in self.fc_x:
142                 if self.calc_grid_position(node.y, self.min_y) in self.fc_y:
143                     # print("cost intensive area!!")
144                     node.cost = node.cost + self.Delta_C2 * self.motion[i][2]
145                 # print()
```

# Noted Version Guide

- Line 263-270: Declaring motions for the aircraft
- Line 279-284: Declaring starting point and end point

```
260         @staticmethod
261         def get_motion_model(): # the cost of the surrounding 8 points
262             # dx, dy, cost
263             motion = [[1, 0, 1],
264                       [0, 1, 1],
265                       [-1, 0, 1],
266                       [0, -1, 1],
267                       [-1, -1, math.sqrt(2)],
268                       [-1, 1, math.sqrt(2)],
269                       [1, -1, math.sqrt(2)],
270                       [1, 1, math.sqrt(2)]]
271 
272             return motion
273 
274 
275     def main():
276         print(__file__ + " start the A star algorithm demo !!") # print simple notes
277 
278         # start and goal position
279         sx = 0.0  # [m]
280         sy = 0.0  # [m]
281         gx = 50.0  # [m]
282         gy = 0.0  # [m]
283         grid_size = 1  # [m]
284         robot_radius = 1.0  # [m]
```

# Noted Version Guide

- Line 309-329: Adding obstacles
- Line 337-348, Adding cost intensive areas (Hint: Refer to this part for your task 2!)

```
308     # set obstacle positions for group 9
309     ox, oy = [], []
310     for i in range(-10, 60): # draw the button border
311         ox.append(i)
312         oy.append(-10.0)
313     for i in range(-10, 60): # draw the right border
314         ox.append(60.0)
315         oy.append(i)
316     for i in range(-10, 60): # draw the top border
317         ox.append(i)
318         oy.append(60.0)
319     for i in range(-10, 60): # draw the left border
320         ox.append(-10.0)
321         oy.append(i)
322
323     for i in range(-10, 30): # draw the free border
324         ox.append(20.0)
325         oy.append(i)
326
327     for i in range(0, 20):
328         ox.append(i)
329         oy.append(-1 * i + 10)
330
331     # for i in range(40, 45): # draw the button border
332     #     ox.append(i)
333     #     oy.append(30.0)
334
335
336     # set cost intesive area 1
337     fc_x, fc_y = [], []
338     for i in range(30, 40):
339         for j in range(0, 40):
340             fc_x.append(i)
341             fc_y.append(j)
342
343     # set cost intesive area 1
344     tc_x, tc_y = [], []
345     for i in range(10, 20):
346         for j in range(20, 50):
347             tc_x.append(i)
348             tc_y.append(j)
```

# Bonus !

- If you wish to do the trip cost (in terms of expense) calculation using the program, you should add the calculation function under line 117, inside the reaching goal condition

- It would be even better if the program could distinguish viable and non-viable aircraft types!

- Use the noted version as your example to modify your own code!

# Bonus Showcase

When you add in a cost calculation function, the output should look something like this, it should be able to:

1. Calculate and show each aircraft types' operating costs
2. Mention which type might not be viable for certain scenarios

```
min_x: -10
min_y: -10
max_x: 60
max_y: 60
x_width: 70
y_width: 70
Total travelling time ->  93.35575746753788
A321 not viable!
Total cost of operating A330 in this scenario:  27360.167918740684
Total cost of operating A350 in this scenario:  30752.648960130347
```