

Chuyển đổi mã nguồn C++ sang Python Thuật toán LaCAM2 cho bài toán MAPF

Sinh viên: **Bùi Xuân Tùng** MSSV: **20224905**

Giáo viên hướng dẫn: **ThS. Nguyễn Tiến Thành**

<https://github.com/PakerB/Project-3---MAPF>

Nội dung

- 1 Bài toán
- 2 Kiến trúc dự án Python
- 3 Ánh xạ C++ → Python
- 4 Module Graph (graph.py)
- 5 Module Instance (instance.py)
- 6 Module DistTable (dist_table.py)
- 7 Core Planner (planner.py)
- 8 Interface & Hậu xử lý
- 9 So sánh & Kết luận

Bài toán MAPF (Multi-Agent Path Finding)

Đầu vào

Bản đồ lưới có vật cản + N agent (start/goal).

Ràng buộc

- **Va chạm đỉnh (vertex collision)**: không có 2 agent đứng cùng một ô tại cùng thời điểm.
- **Va chạm cạnh/hoán đổi (swap collision)**: không có 2 agent đổi chỗ cho nhau trong cùng 1 bước.

Mục tiêu

Tìm lộ trình cho tất cả agent đến goal; có thể tối ưu makespan hoặc tổng chi phí.

LaCAM2 là gì?

- LaCAM2: **Lazy Constraint Addition Search** cho MAPF.
- Ý tưởng: chỉ thêm ràng buộc khi thật sự cần, thay vì liệt kê mọi ràng buộc ngay từ đầu.
- Kết hợp **high-level search** (cấu hình toàn cục) và **low-level** (PIBT).

Cấu trúc module chính

File	Vai trò
graph.py	Đồ thị lưới: Vertex/Graph, neighbor, hash config
instance.py	Load map/scenario, tạo start/goal
dist_table.py	Heuristic distance table (BFS lazy)
planner.py	Core: LaCAM2 search + PIBT + xử lý swap
lacam2.py	Entry: hàm solve(...)
post_processing.py	Validate solution + metrics
utils.py	Deadline, log, random helper

Luồng chạy tổng quát

- ① Tạo Instance từ map/scenario (starts/goals).
- ② Tạo Planner và DistTable (heuristic).
- ③ Lặp tìm kiếm: tạo cấu hình kế tiếp bằng **PIBT**, tránh va chạm.
- ④ Nếu đạt cấu hình goal: backtrack để xuất đường đi.
- ⑤ Hậu kiểm nghiệm bằng post_processing.

Nguyên tắc port C++ → Python

- Giữ đúng **logic thuật toán** trước; tối ưu sau.
- Thay con trỏ/struct bằng **class + reference**.
- STL: vector/map/queue → Python: list/dict/deque.
- Tách module rõ ràng: Graph / Instance / Heuristic / Planner / Validate.

Bảng đối chiếu nhanh

Khối chức năng	C++ (gốc)	Python (port)
Đồ thị	Graph/Vertex	graph.py
Bài toán	Instance	instance.py
Heuristic	DistTable (BFS)	dist_table.py
Tìm kiếm	Planner/HNode/LNode	planner.py
API	solve(...)	lacam2.py
Hậu kiểm	feasible + metrics	post_processing.py

Graph/Vertex: biểu diễn bản đồ lưới

- Đọc file .map: lấy width/height + ma trận ký tự.
- Bỏ ô vật cản (@ hoặc T), tạo Vertex cho ô trống.
- Mỗi Vertex lưu danh sách neighbor (4 hướng) để agent có thể đi.

So sánh: Graph/Vertex (C++ vs Python)

```
graph.cpp 9
lacam2 > lacam2 > src > graph.cpp > ...
1 #include "../include/graph.hpp"
2
3 Vertex::Vertex(uint _id, uint _index)
4   : id(_id), index(_index), neighbor(Vertices())
5 {
6 }
7
8 Graph::Graph() : v(Vertices()), width(0), height(0) {}
9 Graph::~Graph()
10 {
11   for (auto& v : v)
12     if (v != nullptr) delete v;
13   v.clear();
14 }
15
16 // to load graph
17 static const std::regex r_height = std::regex(R"(height\s(\d+))");
18 static const std::regex r_width = std::regex(R"(width\s(\d+))");
19 static const std::regex r_map = std::regex(R"(map)");
20
21 Graph::Graph(const std::string& filename) : v(Vertices()), width(0),
22 height(0)
23 {
24   std::ifstream file(filename);
25   if (!file) {
26     std::cout << "file " << filename << " is not found." << std::endl;
27     return;
28   }
29   std::string line;
30   std::smatch results;
31
32   // read fundamental graph parameters
33   while (getline(file, line)) {
34     // for CRLF coding
35     if ((*line.end() - 1) == 0xd) line.pop_back();
36
37     if (std::regex_match(line, results, r_height)) {
38       height = std::stoi(results[1].str());
39     }
40   }
41 }
```

```
graph.py
lacam2_py > lacam2 > graph.py > ...
1     """
2     Định nghĩa đồ thị cho LaCAM2
3     """
4
5     import re
6     from typing import List, Optional
7
8
9     class Vertex:
10         """Định của đồ thị đại diện cho một vị trí"""
11
12         def __init__(self, vertex_id: int, index: int):
13             self.id = vertex_id # chỉ số cho V trong Graph
14             self.index = index # chỉ số cho U, width * y + x, trong Graph
15             self.neighbor: List['Vertex'] = []
16
17         def __repr__(self):
18             return str(self.index)
19
20         def __str__(self):
21             return str(self.index)
22
23
24     # Định nghĩa kiểu dữ liệu
25     Vertices = List[Optional[Vertex]]
26     Config = List[Vertex] # tập hợp vị trí của tất cả agent
27
28
29     class Graph:
30         """Biểu diễn đồ thị dạng lưới"""
31
32         def __init__(self, filename: Optional[str] = None):
33             self.V: List[Vertex] = [] # without None
34             self.U: List[Optional[Vertex]] = [] # with None
35             self.width: int = 0
36             self.height: int = 0
37
38             if filename:
```

Instance: định nghĩa bài toán MAPF

- Lưu **Graph** + **starts/goals** cho N agent.
- 3 cách khởi tạo: từ scenario, từ chỉ số (test), hoặc sinh ngẫu nhiên.
- Hàm `is_valid` kiểm tra nhất quán dữ liệu.

DistTable: heuristic khoảng cách (BFS + lazy)

- Mục tiêu: truy vấn nhanh khoảng cách ngắn nhất từ một vertex về goal của agent.
- BFS được mở rộng **lazy**: chỉ chạy đến khi chạm vertex cần truy vấn.
- Lợi ích: tiết kiệm thời gian/bộ nhớ khi không cần full all-pairs.

Planner: trái tim của LaCAM2

- **HNode**: cấu hình toàn cục C của tất cả agent.
- **LNode**: gán một phần ($ai \rightarrow$ đâu) trong 1 timestep.
- Tạo cấu hình kế tiếp bằng **PIBT**, kiểm soát vertex + swap conflicts.

So sánh: HNode & NNode (C++ vs Python)

```
graph.cpp 9 lacam2> lacam2> src > planner.cpp > get_new_config(HNode*, LNode*)  
1 #include "../include/planner.hpp"  
2  
3 LNode::LNode(LNode* parent, uint i, Vertex* v)  
4 : who(), where(), depth(parent == nullptr ? 0 : parent->depth + 1)  
5 {  
6     if (parent != nullptr) {  
7         who = parent->who;  
8         who.push_back(i);  
9         where = parent->where;  
10        where.push_back(v);  
11    }  
12 }  
13  
14 uint HNode::HNODE_CNT = 0;  
15  
16 // for high-level  
17 HNode::HNode(const Config& _C, DistTable& D, HNode* _parent, const uint _g,  
18               const uint _h)  
19 : C(_C),  
20   parent(_parent),  
21   neighbor(),  
22   g(_g),  
23   h(_h),  
24   f(g + h),  
25   priorities(C.size()),  
26   order(C.size(), 0),  
27   search_tree(std::queue<LNode*>())  
28 {  
29     ++HNODE_CNT;  
30  
31     search_tree.push(new LNode());  
32     const auto N = C.size();  
33  
34     // update neighbor  
35     if (parent != nullptr) parent->neighbor.insert(this);  
36  
37     // set priorities  
38     if (parent == nullptr) {  
39         // initialize  
40         for (uint i = 0; i < N; ++i) priorities[i] = (float)D.get(i, C[i]) / N;  
41     } else {  
42         // dynamic priorities, akin to PIBT
```

```
graph.py 9+ lacam2> lacam2> planner.py ...  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32 class LNode:  
33     """Low-level search node"""  
34  
35     def __init__(self, parent: Optional['LNode'] = None, i: int = 0, v: Optional[Vertex] = None):  
36         if parent is None:  
37             self.who: List[int] = []  
38             self.where: List[Vertex] = []  
39         else:  
40             self.who = parent.who.copy()  
41             self.where = parent.where.copy()  
42             if v is not None:  
43                 self.who.append(i)  
44                 self.where.append(v)  
45  
46         self.depth = len(self.who)  
47  
48  
49 class HNode:  
50     """High-level search node"""  
51  
52     HNODE_CNT = 0  
53  
54     def __init__(self, config: Config, D: DistTable, parent: Optional['HNode'] = None, g: int, h: int):  
55         HNode.HNODE_CNT += 1  
56         self.C = config  
57         self.parent = parent  
58         self.neighbor: Set['HNode'] = set()  
59         self.g = g  
60         self.h = h  
61         self.f = g + h  
62  
63         # Cho tìm kiếm low-level  
64         N = len(config)  
65         self.priorities: List[float] = [0.0] * N  
66         self.order: List[int] = list(range(N))  
67         self.search_tree: deque = deque()  
68  
69         # Thiết lập độ ưu tiên  
70         if parent is None:  
71             for (uint i = 0; i < N; ++i) priorities[i] = (float)D.get(i, C[i]) / N;
```

PIBT (Priority Inheritance with Backtracking)

- Mỗi agent chọn candidate: đứng yên hoặc đi sang neighbor.
- Ưu tiên candidate gần goal hơn (theo DistTable) + tie-breaker ngẫu nhiên.
- Nếu bị chặn, agent có thể “đẩy” agent khác đi (kế thừa ưu tiên) để tránh kẹt.

So sánh: PIBT (C++ vs Python)

```
graph.cpp 9 | planner.cpp 9+ |
lacam2 > lacam2 > src > planner.cpp > get_new_config(HNode*, LNode*)
272     bool Planner::get_new_config(HNode* H, LNode* L)
273     {
274         ...
275     }
276
277     const auto i = ai->id;
278     const auto K = ai->v_now->neighbor.size();
279
280     // get candidates for next locations
281     for (auto k = 0; k < K; ++k) {
282         auto u = ai->v_now->neighbor[k];
283         C_next[i][k] = u;
284         if (MT != nullptr)
285             tie_breakers[u->id] = get_random_float(MT); // set tie-breaker
286     }
287     C_next[i][K] = ai->v_now;
288
289     // sort
290     std::sort(C_next[i].begin(), C_next[i].begin() + K + 1,
291               [&](Vertex* const v, Vertex* const u) {
292                 return D.get(i, v) + tie_breakers[v->id] <
293                     D.get(i, u) + tie_breakers[u->id];
294             });
295
296     Agent* swap_agent = swap_possible_and_required(ai);
297     if (swap_agent != nullptr)
298         std::reverse(C_next[i].begin(), C_next[i].begin() + K + 1);
299
300     // main operation
301     for (auto k = 0; k < K + 1; ++k) {
302         auto u = C_next[i][k];
303
304         // avoid vertex conflicts
305         if (occupied_next[u->id] != nullptr) continue;
306
307         auto& ak = occupied_now[u->id];
308
309         // avoid swap conflicts
310         if (ak != nullptr && ak->v_next == ai->v_now) continue;
311
312         // reserve next location
313         occupied_next[u->id] = ai;
314     }
315
316     bool Planner::funcPIBT(Agent* ai)
317     {
318         ...
319     }
320
321     const auto i = ai->id;
322     const auto K = ai->v_now->neighbor.size();
323
324     // get candidates for next locations
325     for (auto k = 0; k < K; ++k) {
326         auto u = ai->v_now->neighbor[k];
327         C_next[i][k] = u;
328         if (MT != nullptr)
329             tie_breakers[u->id] = get_random_float(MT); // set tie-breaker
330     }
331     C_next[i][K] = ai->v_now;
332
333     // sort
334     std::sort(C_next[i].begin(), C_next[i].begin() + K + 1,
335               [&](Vertex* const v, Vertex* const u) {
336                 return D.get(i, v) + tie_breakers[v->id] <
337                     D.get(i, u) + tie_breakers[u->id];
338             });
339
340     Agent* swap_agent = swap_possible_and_required(ai);
341     if (swap_agent != nullptr)
342         std::reverse(C_next[i].begin(), C_next[i].begin() + K + 1);
343
344     // main operation
345     for (auto k = 0; k < K + 1; ++k) {
346         auto u = C_next[i][k];
347
348         // avoid vertex conflicts
349         if (occupied_next[u->id] != nullptr) continue;
350
351         auto& ak = occupied_now[u->id];
352
353         // avoid swap conflicts
354         if (ak != nullptr && ak->v_next == ai->v_now) continue;
355
356         // reserve next location
357         occupied_next[u->id] = ai;
358     }
359
360     ...
361 }
```

```
graph.py
lacam2.py > lacam2 > planner.py > ...
94     class Planner:
95
96         def funcPIBT(self, ai: Agent) -> bool:
97             """Hàm PIBT cho một agent"""
98             i = ai.id
99             K = len(ai.v_now.neighbor)
100
101             # Lấy các ứng viên cho vị trí tiếp theo
102             self.C_next[i] = []
103             for k in range(K):
104                 u = ai.v_now.neighbor[k]
105                 self.C_next[i].append(u)
106                 if self.MT is not None:
107                     self.tie_breakers[u.id] = self.MT.random()
108             self.C_next[i].append(ai.v_now)
109
110             # Sắp xếp theo khoảng cách + tie-breaker
111             self.C_next[i].sort(
112                 key=lambda v: self.D.get(i, v) + self.tie_breakers[v.id])
113
114             # Kiểm tra swap
115             swap_agent = self.swap_possible_and_required(ai)
116             if swap_agent is not None:
117                 self.C_next[i].reverse()
118
119             # Thao tác chính
120             for k in range(K + 1):
121                 u = self.C_next[i][k]
122
123                 # Tránh xung đột đỉnh
124                 if self.occupied_next[u.id] is not None:
125                     continue
126
127                 ak = self.occupied_now[u.id]
128
129                 # Avoid swap conflicts
130                 if ak is not None and ak.v_next == ai.v_now:
131                     continue
132
133                 # Reserve next location
134                 self.occupied_next[u.id] = ai
135
136             ...
137 }
```

Tránh swap collision

- Swap: $A : u \rightarrow v$ và $B : v \rightarrow u$ trong cùng một bước.
- Planner có các hàm mô phỏng cục bộ để quyết định:
 - swap có **thể** xảy ra không?
 - swap có **bắt buộc** không?
- Khi swap “required”: thay đổi thứ tự candidate/ưu tiên để phá bế tắc.

Entry point: lacam2.solve(...)

- Hàm solve là API chính: tạo Planner và gọi planner.solve().
- Tách interface/core giúp dễ dùng và dễ test.

Hậu kiểm nghiệm (post_processing.py)

- Kiểm tra start/goal đúng.
- Kiểm tra vertex collision theo từng timestep.
- Kiểm tra move hợp lệ (đứng yên hoặc sang ô kề).
- Kiểm tra swap collision.

So sánh C++ vs Python

C++

- Nhanh, tối ưu bộ nhớ
- Mã phức tạp (con trỏ/template)

Python

- Dễ đọc, dễ debug/test
- Chậm hơn do overhead runtime