

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++ \rightarrow 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), height(0)
22 {
23     std::ifstream file(filename);
24     if (!file) {
25         std::cout << "file " << filename << " is not found." << std::endl;
26         return;
27     }
28     std::string line;
29     std::smatch results;
30
31     // read fundamental graph parameters
32     while (getline(file, line)) {
33         // for CRLF coding
34         if (*(line.end() - 1) == 0x0d) line.pop_back();
35
36         if (std::regex_match(line, results, r_height)) {
37             height = std::stoi(results[1].str());
38         }
39     }
40 }
```

```
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 class Vertex:
9     """Đỉnh của đồ thị đại diện cho một vị trí"""
10
11     def __init__(self, vertex_id: int, index: int):
12         self.id = vertex_id # chỉ số cho V trong Graph
13         self.index = index # chỉ số cho U, width * y + x, trong Graph
14         self.neighbor: List['Vertex'] = []
15
16     def __repr__(self):
17         return str(self.index)
18
19     def __str__(self):
20         return str(self.index)
21
22
23 # Định nghĩa kiểu dữ liệu
24 Vertices = List[Optional[Vertex]]
25 Config = List[Vertex] # tập hợp vị trí của tất cả agent
26
27
28 class Graph:
29     """Biểu diễn đồ thị dạng lưới"""
30
31     def __init__(self, filename: Optional[str] = None):
32         self.V: List[Vertex] = [] # without None
33         self.U: List[Optional[Vertex]] = [] # with None
34         self.width: int = 0
35         self.height: int = 0
36
37         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 ($ai \rightarrow \text{đâu}$) 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  planner.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
43
44     }
45 }
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

```
graph.py  planner.py 9+  lacam2_py > lacam2 > planner.py > ...
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

```
class LNode:
    """Low-level search node"""

    def __init__(self, parent: Optional['LNode'] = None, i: int = 0, v: Optional[Vertex] = None):
        if parent is None:
            self.who: List[int] = []
            self.where: List[Vertex] = []
        else:
            self.who = parent.who.copy()
            self.where = parent.where.copy()
            if v is not None:
                self.who.append(i)
                self.where.append(v)

        self.depth = len(self.who)

class HNode:
    """High-level search node"""

    HNODE_CNT = 0

    def __init__(self, config: Config, D: DistTable, parent: Optional['HNode'],
                 g: int, h: int):
        self.HNODE_CNT += 1
        self.C = config
        self.parent = parent
        self.neighbor: Set['HNode'] = set()
        self.g = g
        self.h = h
        self.f = g + h

        # Cho tìm kiếm low-level
        N = len(config)
        self.priorities: List[float] = [0.0] * N
        self.order: List[int] = list(range(N))
        self.search_tree: deque = deque()

        # Thiết lập độ ưu tiên
        if parent is None:
```

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+ | graph.py | planner.py 9+
lacam2 > lacam2 > src > C++ planner.cpp > get_new_config(HNode*, LNode*)
272 bool Planner::get_new_config(HNode* H, LNode* L)
273 {
315
316 bool Planner::funcPIBT(Agent* ai)
317 {
318     const auto i = ai->id;
319     const auto K = ai->v_now->neighbor.size();
320
321     // get candidates for next locations
322     for (auto k = 0; k < K; ++k) {
323         auto u = ai->v_now->neighbor[k];
324         C_next[i][k] = u;
325         if (MT != nullptr)
326             tie_breakers[u->id] = get_random_float(MT); // set tie-breaker
327     }
328     C_next[i][K] = ai->v_now;
329
330     // sort
331     std::sort(C_next[i].begin(), C_next[i].begin() + K + 1,
332             [&](Vertex* const v, Vertex* const u) {
333                 return D.get(i, v) + tie_breakers[v->id] <
334                        D.get(i, u) + tie_breakers[u->id];
335             });
336
337     Agent* swap_agent = swap_possible_and_required(ai);
338     if (swap_agent != nullptr)
339         std::reverse(C_next[i].begin(), C_next[i].begin() + K + 1);
340
341     // main operation
342     for (auto k = 0; k < K + 1; ++k) {
343         auto u = C_next[i][k];
344
345         // avoid vertex conflicts
346         if (occupied_next[u->id] != nullptr) continue;
347
348         auto& ak = occupied_now[u->id];
349
350         // avoid swap conflicts
351         if (ak != nullptr && ak->v_next == ai->v_now) continue;
352
353         // reserve next location
354         occupied_next[u->id] = ai;
355         if (swap_agent != nullptr)
356             swap_agent->v_next = u;
357     }
358     return true;
359 }
```

```
graph.py | planner.py 9+
lacam2_py > lacam2 > planner.py > ...
class Planner:
94
486
487     def funcPIBT(self, ai: Agent) -> bool:
488         """Hàm PIBT cho một agent"""
489         i = ai.id
490         K = len(ai.v_now.neighbor)
491
492         # Lấy các ứng viên cho vị trí tiếp theo
493         self.C_next[i] = []
494         for k in range(K):
495             u = ai.v_now.neighbor[k]
496             self.C_next[i].append(u)
497             if self.MT is not None:
498                 self.tie_breakers[u.id] = self.MT.random()
499             self.C_next[i].append(ai.v_now)
500
501         # Sắp xếp theo khoảng cách + tie-breaker
502         self.C_next[i].sort(
503             key=lambda v: self.D.get(i, v) + self.tie_breakers[v.id]
504         )
505
506         # Kiểm tra swap
507         swap_agent = self.swap_possible_and_required(ai)
508         if swap_agent is not None:
509             self.C_next[i].reverse()
510
511         # Thao tác chính
512         for k in range(K + 1):
513             u = self.C_next[i][k]
514
515             # Tránh xung đột đỉnh
516             if self.occupied_next[u.id] is not None:
517                 continue
518
519             ak = self.occupied_now[u.id]
520
521             # Avoid swap conflicts
522             if ak is not None and ak.v_next == ai.v_now:
523                 continue
524
525             # Reserve next location
526             self.occupied_next[u.id] = ai
527             if swap_agent is not None:
528                 swap_agent.v_next = u
529         return True
```


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