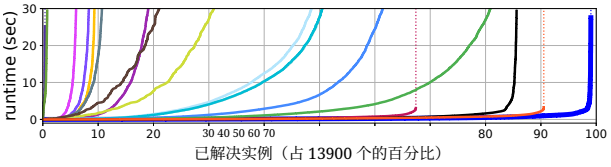
arXiv:2305.03632v1 [cs.AI] 5 May 2023

改进 LaC　　用于可扩展最终最优多智能体路径规划的 AM　　ng

Keisuke Okumura[1,2]
[1]日本产业技术综合研究所（AIST）[2]剑桥大学 ko393@cl.cam.ac.uk

摘要

本研究扩展了近期开发的多智能体路径规划（MAPF）算法 LaCAM。LaCAM 是一种次优的基于搜索的算法，利用惰性后继生成大幅减少规划工作量。我们提出了两项改进。首先，我们提出了其随时版本，称为 LaCAM*；当解的代价是累积转移代价时，该版本最终会收敛到最优解。其次，我们改进了后继生成，以快速获得初始解。详尽的实验展示了这些改进的实用性。例如，LaCAM* 在十秒内于标准台式电脑上次优地求解了从 MAPF基准测试中获取的 99% 实例，其中智能体数量最多达到一千，同时确保最终收敛到最优解；开创了多智能体路径寻找算法的新领域。



算法引用		可解性	最优性	指标
A*	[Hart 等, 1968]	完备	最优	m, l
[illegible]	[Standley, 2010]	完备	次优	（贪心）
算子分解rM*	[Wagner 和 Choset, 2015]	完成	最优	m, l
I			下界。次优。	m, l
BCP	[Lam等, 2022]	解比较最优		f
(	[Sharon等, 2015]	解比较最优		f、m、l
EECBS	[Li等人, 2021年c]	解比较有界次优	f、m、l	
[illegible]	[Silver, 2005]	不完备	次优	
LNS2	[Li等, 2022]	不完备	次优	
]		不完备	次优	
		完备	次优	
]		**完整**	**最终最优 m、l**	

图 1：MAPF基准测试上的性能。上方： 在 33 张四连通网格地图上的 13,900 个实例中已解决实例的数量，数据取自 [Stern 等人, 2019]。智能体数量最多为 1,000。“-f”、“-m”和“-l”分别表示算法尝试最小化流动时间、最大完成时间或损失总和。LaCAM* 的分数对应初始解。下方： 具有代表性或最先进的多智能体路径寻找算法。“solution cmp.” 表示算法保证为可求解实例找到解，但无法识别不可求解的实例。“bnd. sub-opt.” 表示有界次优算法。其次优性设为五。

1 引言

多智能体路径规划（*MAPF*）问题旨在为图上的多个智能体分配无碰撞路径。迄今为止，受仓库自动化等各种应用的推动，研究人员已开发出多种多智能体路径寻找算法 [Wurman等, 2008]。理想的多智能体路径寻找算法应具备完备性、最优性、快速性和可扩展性。然而，前两者与后两者之间通常存在权衡。相反，多智能体路径寻找算法发展的主要挑战在于，在抑制规划工作量以确保速度和可扩展性的同时，保证问题的可解性和解质量。

为打破这一权衡，我们对近期提出的一种名为*LaCAM*（面向多智能体路径寻找的惰性约束添加搜索） [Okumura, 2023]的算法进行了两项改进。该算法是完备的、次优的、基于搜索的（类似于 A* 搜索），并采用惰性后继生成。第一项改进是其 anytime 版本，名为*LaCAM**；如果解的代价由转移代价累积而成，该版本最终会收敛至最优解。由于以最优方式求解多智能体路径寻找在计算上难以处理 [Yu and LaValle, 2013], 因此，处理大型实例的一种实用方法是先求得次优解，再随着时间推移不断改进其质量。LaCAM* 满足了这类需求。第二项改进针对后继生成，即调整 PIBT 算法 [Okumura 等人, 2022],以便快速获得初始解。

通过这些改进，我们以实证结果证明，LaCAM* 能够打破这一权衡。例如，在标准台式电脑上，它在十秒内次优地求解了从 MAPF基准测试 [Stern 等人中获取的 99% 实例，同时保证最终能够找到最优解。如图 1 所示，这一结果超越了现有多智能体路径寻找算法的极限。接下来，我们将依次介绍预备知识、LaCAM*、改进的后继生成方法、实证结果和讨论。附录和代码见 https://kei18.github.io/lacam2/。

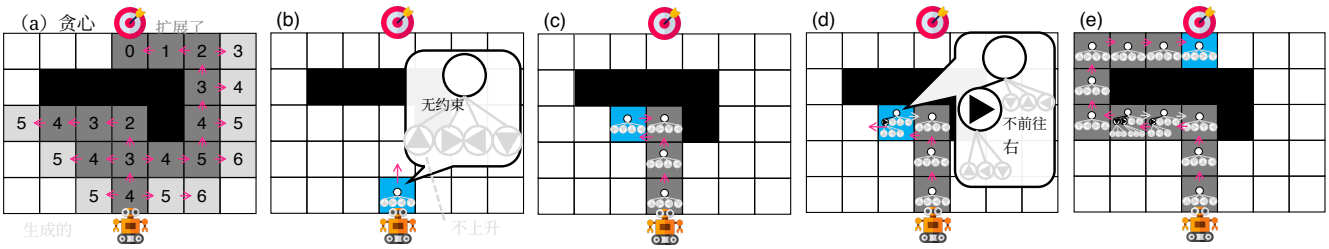


图 2：使用单智能体网格路径搜索的 LaCAM 示意图。

2 预备知识

2.1 符号、问题定义与假设

记号。$S[k]$ 表示序列S中的第 k个元素，索引从一开始。为方便起见，我们用 $\perp$表示“未定义”或“未找到”。

实例。 一个多智能体路径查找实例 由图 $G=(V,E)$、智能体集合 $A=\{1,\dots,n\}$、互不相同的起点元组 $\mathcal{S}=(s_i\in V)^{i\in A}$ 和目标 $\mathcal{G}=(g_i\in V)^{i\in A}$定义。

配置。 配置是所有智能体位置组成的元组。例如， $\mathcal{Q}=(v_1,v_2,\dots,v_n)\in V^{|A|}$ 是一个配置，其中 $\mathcal{Q}[i]=v_i$ 是智能体 $i\in A$ 的位置。起始配置和目标配置分别为 $\mathcal{S}$ 和 $\mathcal{G}$。

碰撞与连通性。 当存在一对智能体 $i,j\in A, i\neq j$，满足 $\mathcal{Q}[i]=\mathcal{Q}[j]$时，配置 $\mathcal{Q}$ 存在顶点碰撞 。当存在一对智能体 $i,j\in A, i\neq j$，满足 $X[i]=Y[j]\wedge Y[i]=X[j]$时，配置 X 和 Y 存在边碰撞 。令 neigh(v) 表示与 $v\in V$ 相邻的顶点集合。当对所有 $i\in A$， $Y[i]\in \text{neigh}(X[i])\cup\{X[i]\}$ ，且 X 和 Y中既不存在顶点碰撞也不存在边碰撞时，配置 X 和 Y是连通的 。

决策问题。给定一个多智能体路径查找实例，多智能体路径查找问题是一个决策问题，用于询问是否存在配置序列 $\Pi=(\mathcal{Q}_0,\ \mathcal{Q}_1,\ \dots\dots,\ \mathcal{Q}_k)$，使得 $\mathcal{Q}_0=\mathcal{S}$、 $\mathcal{Q}_k=\mathcal{G}$，并且 Π 中任意两个连续配置彼此连通。多智能体路径查找的解是满足这些条件的 Π 。为与文献保持一致， Π 的索引特例从零开始（即 $\Pi[0]=\mathcal{Q}_0$）。当一个算法为可解实例返回解，并对不可解实例报告不存在解时，称其为完备。否则，称其为不完备。

优化问题。 给定两个配置之间的转移（或边）代价 $\text{cost}_e: V^{|A|}\times V^{|A|}\mapsto\mathbb{R}_{\geq 0}$，我们的目标是最小化一个解 Π 的累积转移代价，记为 $\text{cost}(\Pi):=\sum_{t=0}^{k-1}\text{cost}_e(\Pi[t],\ \Pi[t+1])$。这种表示法可以表示各种优化指标。例如，当 $\text{cost}_e(X,Y):=1$ 时，cost 被称为 最大完成时间 。当 $\text{cost}(X,Y):=|\{i\in A;\ X[i]\neq Y[i]\}|$时，它被称为 燃料总和 （又称 总行驶距离）。损失总和 统计未停留在目标位置的动作次数，其定义为 $\text{cost}_e(X,Y):=|\{i\in A\mid\neg(X[i]=Y[i]=g_i)\}|$。 当不存在满足 $\text{cost}(\Pi')<\text{cost}(\Pi)$ 的解 Π' 时， 解 Π 被称为 最优解。

关于流动时间的说明。 MAPF 的另一个常见指标是流动时间 （又称代价和）： $\sum_{i\in A}t_i$， 其中$t_i\leq k$ 是满足 $\Pi[t_i][i]=\Pi[t_i+1][i]=\dots=\Pi[k][i]=g_i$。 它与损失总和的区别在于：智能体曾经到达目标并暂时停留在那里时，其代价贡献也会被计入。流动时间取决于各智能体路径的历史，因此无法像累积代价那样直接表示。[1] 相反，本文考虑损失总和，如 [Standley，2010 年；Wagner 和 Choset， 2015] 所述。

可采纳启发式。 我们假设存在一个可采纳启发式h： $V^{|A|}\mapsto\mathbb{R}_{\geq 0}$，使得 h($Q$) 始终不超过从 Q 到 $\mathcal{G}$ 的最优代价；例如，对于 {loss, fuels} 之和，可使用 $\text{h}(\mathcal{Q}):=\sum_{i\in A}\text{dist}(\mathcal{Q}[i],g_i)$，其中 dist： $V\times V\mapsto\mathbb{N}_{\geq 0}$ 表示在 G上的最短路径长度。

将多智能体路径寻找理解为图路径搜索。 使用配置，考虑一个图 H， 其顶点表示配置，边表示配置之间的连通性。然后，将 $\mathcal{S}$和 $\mathcal{G}$ 分别视为起点顶点和目标顶点，最优多智能体路径寻找等价于在 H上进行最短路径搜索的问题。这是理解 LaCAM$^{(*)}$的关键视角。

2.2 LaCAM

LaCAM [Okumura，2023] 最初被开发为一种次优完备的多智能体路径寻找算法。简而言之，它是一种图寻路算法，类似于 A*，但其中一些部分是多智能体路径寻找所特有的。下面，我们以单智能体网格路径搜索为例，介绍图路径搜索部分的核心内容。多智能体路径寻找特有部分的详细信息见附录。

经典搜索。 请参见图 2a，其中展示了典型搜索方案如何解决网格路径规划问题。具体而言，我们展示了使用曼哈顿距离作为启发式函数的贪心最佳优先搜索。在这里，智能体的位置对应搜索中的一个配 置 （亦称状态）。从起始配置出发，搜索会生成三个后继节点：向左、向上和向右。每个节点对应一个配置。然后，搜索根据启发式函数选取一个已生成的节点，并生成其后继节点。这个过程持续进行，直到找到目标配置。

分支因子。可以通过考察生成的节点数量来估算搜索工作量。尽管解的长度为 8，仍生成了 22 个节点。这个数量与连通配置的数量（即分支因子）有关。在网格寻路中，分支因子为 4，因此节点

[1]然而，值得注意的是，可以通过引入虚拟目标来定义流动时间；一旦智能体到达该处，之后便无法移动。

生成数量仍在可接受范围内。然而，多智能体路径寻找的分支因子会随着智能体数量呈指数级增长。因此，单是生成节点就会变得难以处理。这就是为什么普通的 A* 无法求解大型多智能体路径查找实例。

配置生成器。 当配置生成器可用时，LaCAM 会尝试缓解这种巨大的分支因子问题。给定一个配置和约束，它会生成符合约束的连通配置。约束应由各个领域具体定义。在本例中，可以将约束视为对移动方向的禁止，例如不向上、向左、向右或向下移动。

约束树。 LaCAM 的每个搜索节点不仅包含一个配置，还包含约束，并以树结构的形式组织。对于每次节点调用，LaCAM 都会通过低层搜索逐步扩展这棵树，例如采用广度优先搜索（BFS）。树上的一个节点包含一项约束，并通过追溯到根节点的路径表示多项约束。

搜索流程。 现在我们结合图 2b–e，以深度优先搜索（DFS）的方式说明 LaCAM。寻找配置序列的过程称为高层搜索。首先，检查起点的搜索节点（图 2b）。首次调用时，该节点不包含任何约束，这意味着配置生成器可以输出任意连通配置。假设生成器遵循曼哈顿距离指引，输出一个“向上”配置，如粉色箭头所示。为第二次调用做准备时，该节点会扩展一棵约束树，并加入新的约束（例如“不能向上移动”）。高层搜索不会立即丢弃已检查的节点，而是在所有连通配置都生成完毕时（即低层搜索完成时）才将其丢弃。接下来，图 2c–d 展示了节点第二次调用的示例。在图 2c 中，生成器输出了一个已知配置。由于本例采用 DFS，LaCAM 会在图 2d 中再次检查蓝色节点。这次，生成器必须遵循“不能向右移动”这一约束及其父节点的“无约束”。随后，该示例生成一个“向左”配置。搜索继续进行，直到找到目标（图 2e），并通过回溯获得一条解路径。

对多智能体路径寻找的适配。 通过恰当设计约束和树的构建方式，LaCAM 可以对图路径搜索问题进行穷尽搜索，并保证完备性。在 [Okumura, 2023], 通过让约束指定下一个配置中每个智能体的位置，展示了一个将此方法用于多智能体路径寻找的例子。此外，如果配置生成器能够生成接近目标的配置，LaCAM 就能大幅减少节点生成数量。这种削减可能是实现快速规划的关键，尤其适用于像多智能体路径寻找这样分支因子巨大的规划问题。剩下的问题是如何实现优秀的配置生成器。在原论文中，*PIBT*（带回溯的优先级继承） [Okumura 等人 2022] 将其用作配置生成器，相关说明见第2.3节。

伪代码。 算法 1展示了基于深度优先搜索的 LaCAM。每个搜索节点 N 存储 （*i*）一个配置、（*ii*）一个由队列体现的约束树（假设采用广度优先搜索），以及（*iii*）一个指向父节点的指针（见第 2 行）。节点存储在 $Open$ 列表中，并且

算法 1 LaCAM。 $\mathcal{C}_{\text{init}}$ 表示“无约束”。

输入：多智能体路径查找实例

输出：解 或 NO_SOLUTION

1：初始化 $Open$， $Explored$ ▷ 栈、哈希表

2： $\mathcal{N}^{\text{init}} \leftarrow \langle\, config:\ \mathcal{S},\ tree:\ [\![\, \mathcal{C}^{\text{init}} \,]\!],\ parent:\ \bot \,\rangle$

3 $Open.$ $(\mathcal{N}^{\text{init}})$ $Explored[\mathcal{S}] = \mathcal{N}^{\text{init}}$：压入；：当

4 $Open \neq \emptyset$ 时执行 *5*：$\mathcal{N} \leftarrow Open.$栈顶() *6*：如果 $\mathcal{N}.config = \mathcal{G}$ 则返回回溯($\mathcal{N}$) *7*：如果 $\mathcal{N}.tree = \emptyset$ 则 $Open.$弹出()；继续 ▷ 丢弃节点 *8*：$\mathcal{C} \leftarrow \mathcal{N}.tree.$弹出() ▷ 获取约束 *9*：低_层_扩展($\mathcal{N}$， $\mathcal{C}$) ▷ 继续低层搜索 *10*：$\mathcal{Q}^{\text{new}} \leftarrow$ 配置_生成器($\mathcal{N}$， $\mathcal{C}$) *11*：如果 $\mathcal{Q}^{\text{new}} = \bot$ 则继续▷生成器可能失败 *12*：如果 $Explored\,[\mathcal{Q}^{\text{new}}] \neq \bot$ 则继续

13 $\mathcal{N}$ $\leftarrow \langle\, config$ $\mathcal{Q}^{\text{new}}$ $tree$ $[\![\, \mathcal{C}^{\text{init}} \,]\!]$ $parent$ $\mathcal{N} \,\rangle$：新建：，：，：：

14 $Open.$ $\mathcal{N}^{\text{new}}$ $Explored\,[\mathcal{Q}^{\text{new}}] = \mathcal{N}$

压入 ()；新建 *15*：返回无解_

算法 2PIBT

输入：配置 $\mathcal{Q}^{\text{from}}$，智能体 A，目标（g_1， $\ldots g_n$）

输出：配置 $\mathcal{Q}^{\text{to}}$（每个元素均初始化为 $\bot$）

1: **当** $i \in A$ **时执行**；**若** $\mathcal{Q}^{\text{to}}[i] = \bot$ **则** PIBT(i)

2: **返回** $\mathcal{Q}^{\text{to}}$

3: **过程**PIBT(i) ▷ 返回有效或无效

4: $C \leftarrow$ 邻居 $\left(\mathcal{Q}^{\text{from}}[i]\right) \cup \left\{\mathcal{Q}^{\text{from}}[i]\right\}$ ▷候选顶点

5: 按距离(u, g_i)升序排序 C，其中 $u \in C$

6: **对** $v \in C$ **执行**

7: **如果** $\mathcal{Q}^{\text{to}}$ 中存在碰撞，假设 $\mathcal{Q}^{\text{to}}[i] = v$ **则继续**

8: $\mathcal{Q}^{\text{to}}[i] \leftarrow v$

9: **若** $\exists j \in A$ 满足 $j \neq i \wedge \mathcal{Q}^{\text{from}}[j] = v \wedge \mathcal{Q}^{\text{to}}[j] = \bot$ **则**

10: **若**PIBT(j) $=$ 无效 **则继续**

11: **返回** 有效 ▷赋值完成

12: $\mathcal{Q}^{\text{to}}[i] \leftarrow \mathcal{Q}^{\text{from}}[i]$; **返回**无效

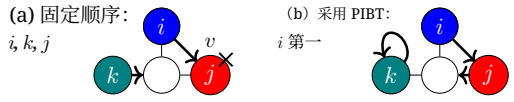


图 3：PIBT 的概念。图中展示了 $\mathcal{Q}^{\text{from}}$。粗箭头表示 $\mathcal{Q}^{\text{to}}$ 的分配。*(a)* 考虑对 i、 k 和 j 固定的分配顺序。如果按照图示箭头分配 i 和 k，则 j 没有候选顶点，因为 $\mathcal{Q}^{\text{to}}[j]$（用 $\times$ 标注）。*(b)* 通过优先为 j 进行分配，以响应 i 的分配请求，从而避免了这一问题。

已探索集合 表，类似于一般搜索模式，并且逐一进行处理。我们通过第 9 行抽象出如何为多智能体路径寻找创建约束树，具体内容在附录中详细说明。

2.3 PIBT

PIBT [Okumura 等人， 2022] 最初是为迭代求解多智能体路径寻找（MAPF）而开发的。简而言之，它是一个配置生成器，用于生成一个新的连通配置（$\mathcal{Q}^{to}$），给定另一个配置（$\mathcal{Q}^{from}$）作为输入。通过持续生成配置，PIBT 可以为多智能体路径寻找生成一个解。

概念。 为了确定 $\mathcal{Q}^{\text{to}}$，PIBT 会依次为每个智能体分配一个顶点，同时避免分配结果引发碰撞。该分配顺序会自适应变化。具体而言——

具体而言，在将 $Q^{to}[i]$ 固定到 $v \in \text{neigh}(Q^{from}[i]) \cup \{Q^{from}[i]\}$之前，PIBT首先检查是否存在满足 $Q^{from}[j] = v$的 $j \in A$。如果存在这样的 j，由于碰撞规避，j 可能没有可用于 $Q^{to}[j]$的候选顶点（见图3a）。因此，PIBT接下来会优先确定 $Q^{to}[j]$，而不是其他智能体的位置（例如图3中的 k）。这种方案称为优先级继承。如果 $Q^{to}[j]$ 成功分配，$Q^{to}[i]$ 就固定到 v（图3b）；否则，$Q^{to}[i]$ 需要选择除 v之外的另一个顶点。

伪代码。 算法 2 通过递归调用 PIBT 过程（第 3 行），实现了上述概念。该过程接收智能体 i，并最终为其分配 $Q^{to}[i]$。第 6–11 行的分配尝试会针对与 $Q^{from}[i]$ 相关的候选顶点进行，顺序按其到 g_i 的距离从小到大排列。尝试会持续进行，直到确定 $Q^{to}[i]$，并得到有效结果（第 11 行）。若所有尝试均失败，PIBT(i) 会将 $Q^{from}[i]$ 分配给 $Q^{to}[i]$，并返回无效（第 12 行）。必要时会触发优先级继承，具体做法是调用 PIBT 为另一个智能体 j 进行分配（第 10 行）。优先级继承成功时会返回有效；否则返回无效，然后 i 继续尝试分配。

动态优先级。 除了优先级继承外，PIBT 还会优先为尚未到达目标的智能体进行分配，这通过按此方式对第 1 行的 A 排序来实现。此方案适用于终身场景，因为在这类场景中，所有智能体不一定会同时到达目标。

3 LaCAM*：最终最优算法

算法 3 展示了 LaCAM*。与 LaCAM（算法 1）相同的行以灰色标出。蓝色行从理论角度来看并非必要，但能有效加快搜索。LaCAM 与其主要有两点不同：*(i)* 找到目标配置$\mathcal{G}$后继续搜索；以及 *(ii)* 在必要时重写搜索节点之间的父节点关系。为方便起见，转移代价 cost_e 和可采纳启发式 h 可以将节点而非配置作为参数。下面将解释更新的部分。

保留目标节点。 LaCAM* 保留目标节点 $\mathcal{N}^{\text{goal}}$，而不是在首次找到目标 $\mathcal{G}$ 时立即返回解（第 6 行）。当 $Open$ 中没有剩余节点时，搜索终止；否则，当用户发出中断请求（例如超时）时，搜索也会中断（第 4 行）。随后，通过从 $\mathcal{N}^{\text{goal}}$ 回溯来构造解（第 27–28 行）。这样，LaCAM* 就成为一种随时算法，即找到目标节点后，只要需要解，便可随时中断，同时随着可用时间增加逐步改进解质量。

搜索节点组成。 每个高层节点都包含一个集合 $neigh$，用于存储连通配置（即节点），以及表示从起点 $\mathcal{S}$到达该节点的成本的 g值（第2行）。在发现新配置时，它们会被初始化并进行适当更新（第24–26行）。

更新父节点和成本。 为保持最优性，当找到已知配置时，LaCAM* 会更新$neigh$（第14行）。随后，Dijkstra算法会更新g值和父节点（第15–21行）。图4展示了更新过程。

算法 3 LaCAM*。 $\mathcal{C}_{\text{init}}$ 表示“无约束”。输入：多智能体路径查找实例，边代价 cost_e，可采纳启发式 h输出：解、NO_SOLUTION 或 FAILURE符号说明：f($\mathcal{N}$) := $\mathcal{N}.g$ + h($\mathcal{N}$)； ♠：$= (\mathcal{N}_{\text{goal}} \neq \bot)$1 2 $Open$ $Explored$ $\mathcal{N}^{\text{goal}} \leftarrow \bot$: 初始化 , ; $\mathcal{S}$ $tree$: $\mathcal{N}^{\text{init}} \leftarrow \langle config$ $[\![\mathcal{C}^{\text{init}}]\!]$ $\bot$ $neigh$ $\emptyset$ g $0\rangle$: , : , 3 $Open$. $(\mathcal{N}$ $)$ 父节点 : , : , : : 推入 init ; $Explored[\mathcal{S}] = \mathcal{N}^{\text{init}}$ 4 : while $Open \neq \emptyset \wedge \neg$ interrupt() do5: $\mathcal{N} \leftarrow Open$.top() 6 $\mathcal{N}.config = \mathcal{G}$ $\mathcal{N}$ $\leftarrow \mathcal{N}$: if then 目标 : if f 目标 f ( ) 7 ♠ $\wedge$ $(\mathcal{N}$ $) \leq$ $\mathcal{N}$ then $Open$.pop(); continue8: if $\mathcal{N}.tree = \emptyset$ then $Open$.pop(); continue9: $\mathcal{C} \leftarrow \mathcal{N}.tree$.pop()10: 低层级扩展($\mathcal{N}, \mathcal{C}$)11: $\mathcal{Q}^{\text{new}} \leftarrow$ 配置生成器($\mathcal{N}, \mathcal{C}$)12: if $\mathcal{Q}^{\text{new}} = \bot$ then continue13: if $Explored[\mathcal{Q}^{\text{new}}] \neq \bot$ then14: $\mathcal{N}.neigh$.追加 ($Explored[\mathcal{Q}^{\text{new}}]$)15: $\mathcal{D} \leftarrow [\![\mathcal{N}]\!]$ ▷ Dijkstra，按 g 值排序的优先队列16: while $\mathcal{D} \neq \emptyset$ do17: $\mathcal{N}$从 $\leftarrow \mathcal{D}$.pop()18: for $\mathcal{N}$to $\in \mathcal{N}$from $.neigh$ do 19 $g \leftarrow \mathcal{N}^{\text{from}}.g +$ $_e$ $(\mathcal{N}$ $\mathcal{N}$ $)$: 从 到 的 20 $g < \mathcal{N}$ $.g$ 代价 : if 到 then21: $\mathcal{N}$到.$g \leftarrow g$; $\mathcal{N}$到 $.parent \leftarrow \mathcal{N}$从; $\mathcal{D}$.推入 ($\mathcal{N}$到)22: if ♠ $\wedge$ f ($\mathcal{N}$到) $<$ f $(\mathcal{N}$目标) then $Open$.推入 ($\mathcal{N}$到) 23: else24: $\mathcal{N}$新建 $\leftarrow \langle config$: $\mathcal{Q}^{\text{new}}$, $tree$: $[\![\mathcal{C}^{\text{init}}]\!]$, $parent$: $\mathcal{N}$, $neigh$ $\emptyset$ g $\mathcal{N}.g +$ $_e$ $\mathcal{N}$ $\mathcal{Q}^{\text{new}}$ $\rangle$: , : 代价 (, 25 $Open$. $\mathcal{N}$ $Explored[\mathcal{Q}^{\text{new}}] = \mathcal{N}$) : 推入 (新建); 新建 26: $\mathcal{N}.neigh$. 追加 ($\mathcal{N}^{\text{new}}$)27: if ♠ $\wedge$ $Open = \emptyset$ then return 回溯 $(\mathcal{N}$目标) ▷最优28: else if ♠ then return 回溯 $(\mathcal{N}$目标) ▷次优29: else if $Open = \emptyset$ then return NO_SOLUTION30: else return FAILURE

舍弃冗余节点。 找到目标节点后，LaCAM* 会舍弃不利于提升解质量的节点（第7行）。当其 g值更新时，它还会根据需要重新激活这些节点（第22行）。

定理1。 *LaCAM** （算法*3*）是完备且最优的。

证明。 考虑去除蓝色线（第7行和第22行）后的算法3；这些蓝色线只会加快搜索，而不会破坏最优搜索结构。在本证明中，“路径”一词指的是连通配置的序列。

首先，我们引入一个有向图 H，其中每个顶点对应一个配置。初始时，H 仅包含一个起点顶点 $\mathcal{S}$。随后，搜索迭代次数逐步构建 H。创建新节点时（第24行），其配置会被添加到 H中。当搜索发现从 X到 Y的连接时，即 $\mathcal{N}^Y \in \mathcal{N}^X.neigh$，$H$中就会出现一条弧（$X$，$Y$）。

我们现在证明：**(♣**对于 H 中的任意配置 $\mathcal{Q}$，通过回溯（即遵循 $\mathcal{N}.parent$）构造的从 $\mathcal{S}$ 到 $\mathcal{Q}$ 的路径，在每次搜索迭代开始时，按照累计转移代价计算，都是 H 中的最短路径。我们通过归纳法证明这一点。初始时，H 仅包含 $\mathcal{S}$，满足 ♣ 现在假设在 Lines 4–26 的上一次迭代中，♣ 成立。在下一次迭代中，H 是

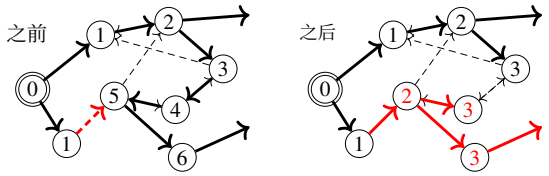


图4：更新父节点和代价。每个圆圈都是一个搜索节点（即配置），其中包括其最大完成时间的g值。箭头表示已知的邻接关系。其中，实线表示 *parent*。更新的部分以红色标出。左图：发现了新的邻接关系，即一条红色虚线箭头。右图：重写搜索树。注意，由于g值剪枝（第20行），重写仅发生在树的有限部分。

更新方式包括：*(i)* 生成新的配置（第 24–26 行），或 *(ii)* 查找已知配置（第 14–21 行）。情况 *(i)*成立 ♣，因为只需向 H 添加一个新顶点以及一条指向该顶点的弧。情况 *(ii)* 成立 ♣，因为只需为节点 $\mathcal{N}$ 添加一条弧（第 14 行），随后第 15–21 行执行从 $\mathcal{N}$ 开始的 Dijkstra 算法，同时维护最短路径的树结构。

LaCAM* 的搜索空间是有限的（参见 [Okumura, 2023]）；搜索将在有限时间内终止。每个节点最终都会检查所有连通配置。因此，终止时，H 包含从 $\mathcal{S}$ 到 $\mathcal{G}$的所有可能路径，适用于可解实例。结合 ♣ LaCAM* 会返回一个最优解；否则，报告不存在解。 □

实现技巧。遵循 [Okumura的做法， 2023], 在第13行发现已知配置时，我们的实现会将相应节点重新插入$Open$。此外，以较小的概率（例如0.1%），实现会重新插入 $\mathcal{S}$ 中的一个节点，而不是刚刚找到的节点。这样做可以使搜索从作为瓶颈的配置中“逃脱”。依赖非确定性的此类技术已见于其他搜索问题 [Kautz 等，以及多智能体路径寻找研究 [Cohen 等，2018b；Andreychuk和Yakovlev， 2018]。事实上，我们通过非正式观察发现，这种随机替换略微提高了成功率。请注意，进行这些修改后，最优性仍然成立。

4 改进配置生成器

LaCAM 的性能在很大程度上依赖于配置生成器，因此，开发优秀的生成器至关重要。 [Okumura，2023] 中的实现采用了算法2的标准 PIBT，在若干场景中表现不佳，尤其是在包含狭窄走廊的实例中。这是因为 PIBT 本身在这类场景中经常失败，因此无法为 LaCAM 提供有效的引导。本节将详细说明这一现象，并介绍一个改进版本。

4.1 PIBT 的失败分析

如第2.3节所示，PIBT 会依次为智能体分配下一位置。由于这种顺序会优先处理尚未到达目标的智能体，因此可能引发活锁情形。如图5所示，两个智能体周期性地到达各自的目标顶点，但 PIBT 始终无法到达目标配置。

算法 4 带交换的 PIBT 过程

1: $C \leftarrow$ 邻居 $(\mathcal{Q}^{\text{from}}[i]) \cup \{\mathcal{Q}^{\text{from}}[i]\}$
2: 按 C 的 距离(u, g_i) 递增排序，其中 $u \in C$
3: $j \leftarrow$ 交换_必需_且_可能 $(i, C[1], \mathcal{Q}^{\text{from}})$
4: **如果** $j \neq \perp$**那么** 反转 C
5: **对于** $v \in C$ **执行**
6: 算法2的第7–10行
7: **如果** $v = C[1] \wedge j \neq \perp \wedge \mathcal{Q}^{\text{to}}[j] = \perp$，**则** $\mathcal{Q}^{\text{to}}[j] \leftarrow \mathcal{Q}^{\text{from}}[i]$。
8: **return** 有效
9: $\mathcal{Q}^{\text{to}}[i] \leftarrow \mathcal{Q}^{\text{from}}[i]$; **返回**无效

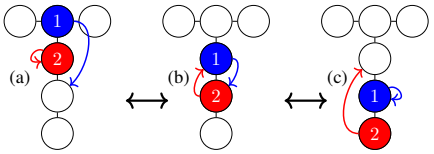


图 5：PIBT 的失败示例。目标以 y a箭头表示。

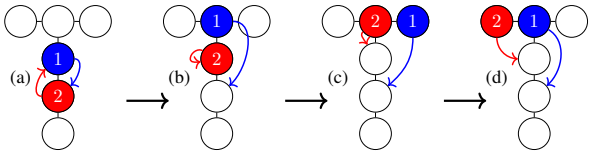
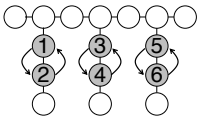


图6：交换操作。最后两个步骤被省略，因为这只是将两个智能体朝其目标移动。



	$\|A\|$=2	$\|A\|$=4	$\|A\|$=6
w/算法2	128	23,907	287,440
使用算法4	6	8	8

表1：LaCAM 求解这些实例所需的搜索迭代次数。当 $|A| = 2$ 时，只出现智能体-$\{1, 2\}$，依此类推。

LaCAM 可以通过施加约束来打破这类活锁。然而，这可能需要大量工作，因为必须探索合适的约束组合。更糟的是，随着智能体数量增加，搜索工作量会急剧上升，如表1所示。

4.2 通过交换增强 PIBT

PIBT 中的活锁可以通过交换操作来解决，该操作最初由基于规则的多智能体路径规划算法提出 [Luna 和 Bekris，2011；De Wilde 等人， 2014]。简而言之，此操作利用度为三或更高的顶点，交换两个智能体的位置。图6展示了一个示例。这里，我们提取其核心思想并将其融入 PIBT。具体而言，这是通过调整算法2第5行的顶点评分来实现的。

算法 4 扩展了算法2中的 PIBT 过程。修改很简单：如果智能体 i 和相邻智能体 j被判定需要交换位置（第 3 行），则 i 会反转候选顶点 C 的顺序（第 4 行）；也就是说， i 会尝试与 g_i拉开距离。随后，如果 i 成功移动到候选项中的第一个顶点 C， i 会拉动 j 移动到当前占据的顶点（第 7 行）。通过适当实现所需的 swap_required_and_possible 函数，PIBT 不会陷入图5所示的活锁，而是能够生成一个序列

图6所示的配置。

交换_所需_且_可能的函数取决于模式检测器和实现。我们的目标不是设计完整的检测器，因为检测器的缺陷可以由*LaCAM*弥补。不过，经过良好调优的实现可以减少LaCAM的搜索工作量。下面我们展示一个示例实现，同时省略细微的调优。

模式检测器实现

假设检测器是为 i调用的。进一步假设，另一智能体 j 位于 $C[1]$，用于 i，并且在算法4第2行时其度不超过2。我们的检测器使用两次仿真。

第一次仿真询问是否有必要进行交换。具体而言，在忽略其他智能体的情况下，持续将 i 移动到 j的位置，同时将 j 移动到一个不同于 i所在位置的另一个顶点。仿真在以下两种情况下停止：*(i)* 当 j所在位置的度大于二时，交换并非必需。*(ii)* 当 j所在位置的度为一，或者当 i 到达 g_i 且 j朝向其目标的最近邻接顶点为 g_i时，交换是必需的。

如果需要交换，第二次仿真会询问交换的可行性。具体做法是反转仿真方向，即在将 i移动到另一个顶点的同时，持续将 j 移动到 i的位置。该过程会在以下两种情况下停止：*(i)* 当 i所在位置的度大于二时，交换可行 。*(ii)* 当 i 位于度为一的顶点上时，交换不可行 。

当需要交换且交换可行时，该函数返回 j。例如，在图6(a、b)的配置中，两个智能体都需要交换。然而，交换仅对智能体1可行，因此智能体1的候选顶点顺序会反转（第4行）。于是，算法4生成图6(b、c)的配置。图6c是一个例外，此时智能体2需要反转其候选顶点，以生成图6d的配置。我们注意到，通过应用两次仿真同样可以检测出这种情况。

5 评估

本节通过实证评估这两项改进，包括：*(i)* 改进的配置生成器如何减少规划开销，*(ii)* LaCAM* 如何改进解，*(iii)* 舍弃冗余节点如何加快收敛，*(iv)* 使用小型复杂实例进行评估，*(v)* 使用MAPF基准测试进行评估，*(vi)* 与另一种任意时刻MAPF算法进行比较，以及*(vii)* 在极高密度场景下进行评估。

设置。 实验是在一台配备英特尔处理器的台式电脑上进行的。tel Core i7-7820X 3.6GHz CPU 和 32GB RAM。最多使用多线程并行运行 16 个不同的实例。LaCAM* 使用 C++ 编写。所有实验均使用从 MAPF 基准测试 [Stern *etal.*, 2019] 获取的四连通网格地图。除非另有说明，本节使用 30s 的超时时间来求解多智能体路径寻找。基线多智能体路径寻找算法总结见图 1。其实现细节见附录。

	搜索迭代次数		运行时间（毫秒）	
$\|A\|$	使用算法2	使用算法4	使用算法2	使用算法4
100	374 (344,54468)	366 (338,401)	65 (31,1218)	112 (34,216)
300	54802 (388,369131)	392 (357,482)	3049 (291,18858)	301 (187,409)
500	181459 (44534,268724)	410 (391,432)	18063 (4598,29820)	500 (347,574)

表2：配置生成器的效果。对于 $|A|$，列出了两种算法均已求解的实例中的中位数、最小值和最大值分数；这些实例是从 [Stern 等人的 25 个实例中检索得到的， 2019], 位于 *仓库-20-40-10-2-1* 上，如图 8 所示。

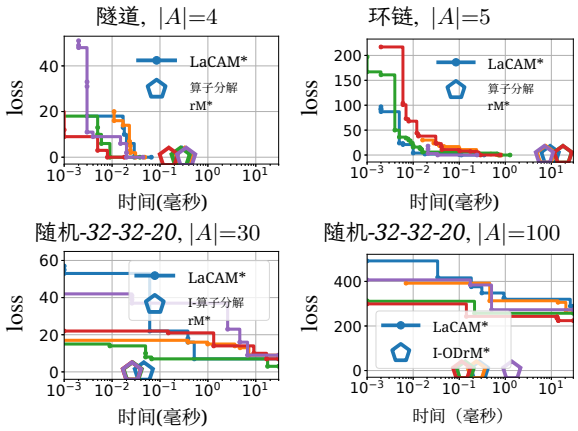


图 7：LaCAM* 的细化。使用了三张地图，如表 4 和图 8 所示。每张图表均使用五个相同的实例，起点和目标均随机设置。优化目标为损失总和。“loss”表示与 (I-)ODrM* 得分之间的差距。在 随机-*32-32-20* 中，由于 ODrM* 未能求解这些实例，因此使用了次优性为 1.5 的有界次优版本。LaCAM* 使用算法 2 作为配置生成器。

	树角	隧道	字符串	环链	连接器
不丢弃	10K	2M	410M	19M	N/A	N/A
使用算法2	1K	28K	287K	103K	N/A	N/A
使用算法4	**1K**	**1K**	**199K**	**103K**	N/A	N/A

表3：终止所需的搜索迭代次数。“不丢弃”是省略蓝线版本的 LaCAM*。该指标针对最大完成时间。实例列于表 4 中。在最后两个实例中，LaCAM* 在超时前未能终止。

改进的配置生成器的效果。 表1展示了使用原版 PIBT（算法2）和改进版 PIBT（算法4）作为配置生成器时，LaCAM 在一个需要“交换”的实例上的搜索迭代次数。表2进一步比较了这些生成器在更大规模实例上的表现。结果表明，算法4大幅减少了 LaCAM 的搜索迭代次数，从而缩短了大规模实例的计算时间。不过请注意，模式检测器会带来运行时间开销，如表2的 $|A| = 100$ 所示。

LaCAM 的细化*。 图 7 展示了 LaCAM*如何细化解。作为基线方法，我们采用了一种名为 (I-)ODrM* [Wagner 和 Choset, 2015] 的完备且最优算法的得分。在小型实例中，LaCAM* 能够快速

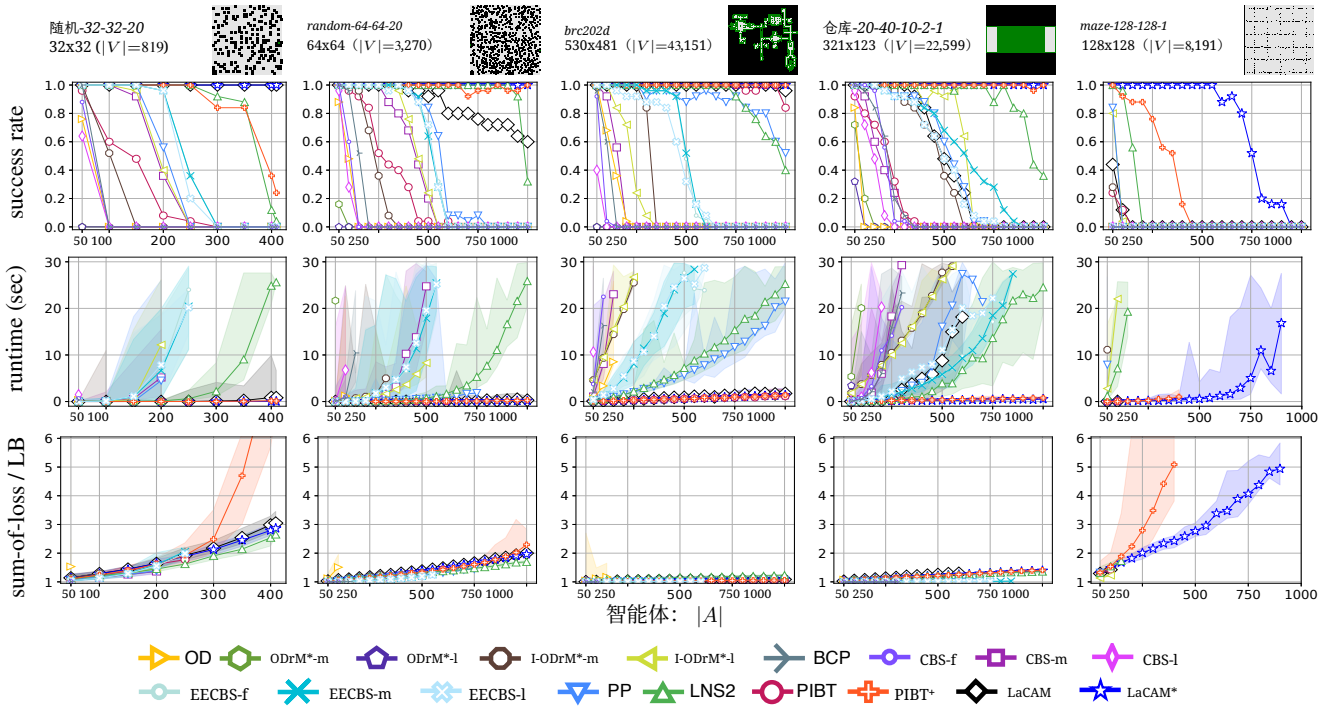
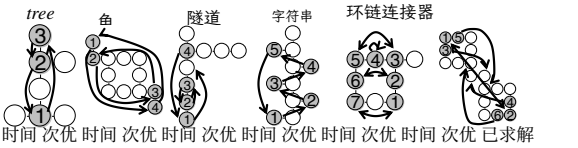


图8：多智能体路径寻找基准测试的结果。损失总和分数按 $\sum_{i \in A}$ 距离$(s_i,\ g_i)$ 归一化。对于运行时间和损失总和，展示各求解器已解决实例的中位数、最小值和最大值分数。LaCAM* 的分数来自初始解。



LaCAM*	0 1.20	0 1.23	0 1.41	0 1.81	2 6.58	01.62	
1 秒后	0 1.00	2 1.00	6 1.00	7 1.00	578 1.35	226 1.00	**6/6**
A*	0 1.00	0 1.00	30 1.00	27 1.0011125	1.00 不适用	不适用	5/6
算子分解rM*	5 1.00	2 1.00	396 1.00	402 1.00	不适用	不适用 不适用 不适用	4/6
I-算子分解rM*	1 1.00	0 1.50	70 1.07	2 1.25	不适用	不适用 不适用 不适用	4/6
CBS	71 1.00	0 1.00	不适用 不适用	149 1.00	不适用	不适用 不适用 不适用	3/6
EECBS	2 1.00	1 1.00	不适用 不适用	0 1.00	不适用	不适用 不适用 不适用	3/6
OD	0 1.00	0 1.88	14 2.73	0 1.25	2133 31.22	5 1.50	**6/6**
LaCAM	0 1.17	1 2.12	92 2.00	0 2.25	55 17.83	0 1.56	**6/6**
PP	不适用 不适用	0 1.00	不适用 不适用	0 1.00	不适用	不适用 不适用 不适用	2/6
LNS2	不适用 不适用	0 1.00	不适用 不适用	01.00	不适用	不适用 291.00	3/6
PIBT	不适用 不适用 不适用 不适用 不适用 不适用			不适用 不适用 不适用 不适用 不适用 不适用			0/6
PIBT+	0 3.50	0 1.25	0 4.07	02.12	不适用	不适用	0 1.81 5/6
BCP	194	- 150	- 不适用	- 117	-	N/A	- 不适用 - 3/6

表 4：小型复杂实例的结果。“次优”是按最优解归一化后的最大完成时间。最小值为一。损失总和版本见附录。两行展示了 LaCAM* 的结果：*(i)* 初始解的得分，以及*(ii)* 在 1s 时的解质量和获得该解时的运行时间；这些结果是采用不同随机种子的 10 次试验的平均值。算法分为 LaCAM*、优化最大完成时间的算法、次优算法，以及优化其他指标（即流动时间）的 BCP。

找到初始解并收敛至最优解。与此同时，在智能体数量较多的大规模实例中，收敛速度较慢。这是因为已知配置之间出现新连接的情况变得越来越少，从而导致……

重写搜索树的可能性。

丢弃冗余节点的影响。 表3展示了丢弃冗余搜索节点（算法3中的蓝色线条）对搜索最优解的影响。无论生成器如何，丢弃冗余搜索节点都显著减少了搜索工作量。算法4带来的减少幅度更大，因为与算法2相比，算法4能以更少的搜索迭代次数找到初始解。请注意，如果不丢弃冗余搜索节点，算法2和算法4的搜索迭代次数相同，因为它们的搜索空间完全一致。在下文中，LaCAM* 使用算法4，而 LaCAM 指使用算法2的原始实现。

小型复杂实例。 表 4 展示了 LaCAM* 在从 [Luna 和 Bekris,2011]检索到的实例上的结果。总体而言，它不仅立即找到了初始解，还找到了（近似）最优解。相比之下，基线方法未能解决部分实例，或返回了质量较低的解。

MAPF基准测试。 我们在 MAPF 基准测试上测试了 LaCAM* ，该基准测试包含 33 张地图，每张地图有 25 个指定起点-目标点对的“随机场景”。对于每个场景，我们将智能体数量每次增加 50，直至达到最大值（大多数情况下为 1,000），由此提取实例，总计获得 13,900 个实例。已解决实例的比例汇总于图 1。图 8 展示了各地图的部分结果。LaCAM* 仅未能解决 *maze-128-128-1* 中的实例，并在 10 秒内以次优解解决了所有其他实例，表现优于其他算法。改进模式检测器或许可以减少失败情况；不过，我们认为，此类实现对于该基准测试而言过于针对性

地进行了优化。如图 9 所示，细化过程稳定但提升并不显著，原因与图 7 相同。更多讨论见附录。

与任意时长MAPF求解器的比较。我们将 LaCAM* 与 AFS [Cohen 等人，2018a], 提出的一种基于 CBS、能够保证收敛至最优解的任意时长MAPF求解器进行了比较。表5总结了结果。与 LaCAM*不同，AFS 从一开始就能获得合理的解，但其可扩展性会因此受到影响。我们认为，通过开发 PIBT 之外的更优配置生成器，可以弥补这一质量差距。

极度密集场景。表6报告了 LaCAM* 在高度拥挤场景中的表现，而现有求解器大多无法处理这些场景。即使面对如此具有挑战性的案例，LaCAM* 仍求解出了许多实例，展现出出色的可扩展性。

6 结论与讨论

多智能体路径寻找面临的主要挑战，是在抑制规划工作量的同时维持可求解性和解质量。为打破这一权衡，本文对 LaCAM 提出了两项增强：LaCAM* 最终能够收敛至最优解，以及一种有效的配置生成器。我们对这些增强进行了全面评估，并取得了显著成果。根据实证结果，我们认为 LaCAM* 在多智能体路径寻找领域开辟了一个新的前沿。

相关工作。 LaCAM(*) 与搜索过程中的部分后继扩展有关，这一点可见于 [Goldenberg 等人，2014；Wagner 和 Choset, 2015], 因为它同样会生成部分后继，但在约束和配置生成器的使用上有所不同。能够收敛至最优解的任意时长多智能体路径寻找算法已有研究 [Standley 和 Korf，2011；Cohen 等人，2018a；Vedder 和 Biswas， 2021]。然而，它们的可扩展性有限；正如我们的实证结果所示，它们往往无法得出初始解。改进任意多智能体路径寻找解的技术也已有研究 [Surynek, 2013；De Wilde 等人，2014；Okumura 等人，2021；Li 等人，2021a]但这些技术无法保证最优性。在最优运动规划中，重写搜索树结构十分常见 [Karaman 和 Frazzoli，2011；Shome 等人， 2020], LaCAM*的设计部分受此启发。将交换机制融入 PIBT，则受到次优基于规则的MAPF算法的启发 [Luna 和 Bekris，2011；De Wilde 等人， 2014]与此同时，基于规则的方法本身往往会严重牺牲解质量。

未来方向。 我们希望探索更有效的构型生成器，其性能优于 PIBT 变体，并能输出近似最优的初始解。提高 LaCAM* 的收敛速度也很重要。此外，研究 LaCAM* 在 MAPF 变体中的应用也值得关注，例如多机器人运动规划 [Okumura 和 Défago， 2023], 。除多智能体路径寻找外，由于 LaCAM* 本身只是图寻路算法，将其理念应用于其他规划领域或许也会很有趣。

致谢

感谢 Yasumasa Tarmura 对初稿提出的意见。本研究部分得到了 JSPS KAKENHI Grant Numbers 20J23011 和 JST ACT-X Grant 的资助。

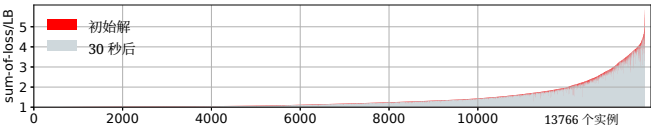


图 9：LaCAM* 对 MAPF基准测试进行的细化。图中在横轴上按初始解质量对 13,900 个实例中的 13,766 个已解决实例进行排序，并用红色柱条显示其得分。对于每个实例，我们还用灰色柱条绘制了其在 30 秒时的解质量。因此，细化的效果以微小的红色区域呈现。

	已求解（%）		初始时间（毫秒）		初始损失		30 秒时的损失	
$\|A\|$	AFS	LaCAM*	AFS	LaCAM*	AFS	LaCAM*	AFS	LaCAM*
50	100		100	88	**1**	**25**	159	**24** 118
100	56		**100** 7223		**2**	**140**	609	**139** 545
150	0		**100** 不适用		**4** 不适用		1463 不适用	1368

表5：任意时长MAPF算法比较。我们使用了损失和以及随机-*32-32-20*的25个“随机”场景。“初始”显示与初始解相关的分数。“损失”是来自$\sum_{i \in A}$距离(s_i, g_i)的差距分数。分数取两个求解器都已求解的实例的平均值，但 $|A| = 150$ 除外，因为AFS全部失败。

map	$\|A\|$	%	时间(s)	其他算法
empty-8-8	58	100	0.00	PIBT&LaCAM（100%；0.00秒）
随机-*32-32-20*	737	100	0.63	LaCAM（4%；14.81秒）
random-64-64-20	2943	68	11.64	不适用
maze-128-128-10	9772	100	55.54	不适用

表6：极度密集场景的结果。 $|A|$ 已调整，使得 $|A|/|V| = 0.9$。对于每种场景，我们通过随机放置起点和目标，准备了 25 个实例。“%”表示 LaCAM* 在 60 秒超时限制下的成功率。“时间”是获得初始解的中位运行时间。我们还测试了图 1 中的其他求解器，并报告了至少在一个实例中求得解的求解器。

编号 JPMJAX22A1。感谢我攻读博士期间获得吉田奖学基金会的支持。

参考文献

[Andreychuk 和 Yakovlev， 2018] Anton Andreychuk 和 Konstantin Yakovlev。两种提升多机器人优先级路径规划性能的技术。发表于自主智能体与多智能体系统国际联合会议（*AAMAS*）论文集，2018 年。[Cohen 等，2018a] Liron Cohen、Matias Greco、Hang Ma、Carlos Hernández、Ariel Felner、TK Satish Kumar 和 Sven Koenig。具有应用的随时焦点搜索。发表于 国际人工智能联合会议（*IJCAI*）论文集，2018 年。[Cohen 等，2018b] Liron Cohen、Glenn Wagner、David Chan、Howie Choset、Nathan Sturtevant、Sven Koenig 和 TK Satish Kumar。用于多智能体路径规划求解器的快速随机重启。发表于组合搜索年度研讨会（*SOCS*）论文集，2018 年。

[De Wilde等人， 2014] Boris De Wilde、Adriaan W Ter Mors 和 Cees Witteveen。推送与旋转：一种

完整的多智能体路径规划算法。《人工智能研究期刊》（JAIR），2014年。[Erdmann 和 Lozano-Perez，1987] Michael Erdmann 和 Tomas Lozano-Perez。关于多个移动物体。《Algorithmica》，1987年。[Goldenberg 等人，2014] Meir Goldenberg、Ariel Felner、Roni Stern、Guni Sharon、Nathan Sturtevant、Robert Holte 和 Jonathan Schaeffer。增强型部分扩展 A*。《人工智能研究期刊》（JAIR），2014年。[Hart 等人，1968] Peter E Hart、Nils J Nilsson 和 Bertram Raphael。启发式确定最小成本路径的形式化基础。《IEEE系统科学与控制论汇刊》，1968 年。[Karaman 和 Frazzoli，2011] Sertac Karaman 和 Emilio Frazzoli。基于采样的最优运动规划算法。《国际机器人研究期刊》（IJRR），2011年。[Kautz 等人，2002] Henry Kautz、Eric Horvitz、Yongshao Ruan、Carla Gomes 和 Bart Selman。动态重启策略。载于 AAAI人工智能会议（AAAI）论文集，2002年。[Lam 等人，2022] Edward Lam、Pierre Le Bodic、Daniel Harabor 和 Peter J Stuckey。面向多智能体路径规划的分支定界与列生成。《计算机与运筹研究》（COR），2022。[Li 等，2021a] Jiaoyang Li、Zhe Chen、Daniel Harabor、P Stuckey 和 Sven Koenig。通过大邻域搜索实现任意时长多智能体路径规划。载于国际人工智能联合会议（IJCAI）论文集，2021年。[Li 等，2021b] Jiaoyang Li、Daniel Harabor、Peter J Stuckey 和 Sven Koenig。多智能体路径规划搜索中的成对对称性推理。《人工智能》（AIJ），2021年。[Li 等，2021c] Jiaoyang Li、Wheeler Ruml 和 Sven Koenig。Eecbs：一种用于多智能体路径规划的有界次优搜索。载于 AAAI 人工智能会议（ AAAI ）论文集，2021 年。[Li 等人，2022] Jiaoyang Li、Zhe Chen、Daniel Harabor、Peter J Stuckey 和 Sven Koenig。Mapf-lns2：通过大邻域搜索快速修复多智能体路径规划。载于 AAAI人工智能会议（AAAI）论文集，2022。[Luna 和 Bekris，2011] Ryan Luna 和 Kostas E Bekris。推送与交换：具有完备性保证的快速协作路径规划。载于国际人工智能联合会议（IJCAI）论文集，2011年。[Okumura 和 Défago，2023] Keisuke Okumura 和 Xavier Défago。结合采样与搜索的快速多机器人运动规划。载于国际人工智能联合会议（IJCAI）论文集，2023年。[Okumura 等人，2021] Keisuke Okumura、Yasumasa Tamura 和 Xavier Défago。用于实时多机器人路径规划的迭代细化。载于IEEE/RSJ智能机器人与系统国际会议（IROS）论文集，2021年。

[Okumura 等人，2022] Keisuke Okumura、Manao Machida、Xavier Défago 和 Yasumasa Tamura。用于迭代式多智能体路径规划的带回溯优先级继承。Artificial Intelligence (AIJ)，2022。[Okumura，2023] Keisuke Okumura。LaCAM：用于快速多智能体路径规划的基于搜索的算法。载于 AAAI人工智能会议（AAAI）论文集，2023。[Sharon 等人，2015] Guni Sharon、Roni Stern、Ariel Felner 和 Nathan R Sturtevant。用于最优多智能体路径规划的基于冲突的搜索。Artificial Intelligence (AIJ)，2015。[Shome 等人，2020] Rahul Shome、Kiril Solovey、Andrew Dobson、Dan Halperin 和 Kostas E Bekris。drrt*：可扩展且具信息性的渐近最优多机器人运动规划。Autonomous Robots (AURO)，2020。[Silver，2005] David Silver。协作式路径规划。载于 AAAI人工智能与互动数字娱乐会议（ AIIDE）论文集，2005年。[Standley 和 Korf，2011] Trevor Standley 和 Richard Korf。协作式路径规划问题的完备算法。载于国际人工智能联合会议（IJCAI）论文集，2011年。[Standley，2010] Trevor Scott Standley。求解协作式路径规划问题的最优解。载于 AAAI人工智能会议（AAAI）论文集，2010年。[Stern 等人，2019] Roni Stern、Nathan Sturtevant、Ariel Felner、Sven Koenig、Hang Ma、Thayne Walker、Jiaoyang Li、Dor Atzmon、Liron Cohen、TK Kumar 等人。多智能体路径规划：定义、变体与基准测试。载于组合搜索年会（SOCS）论文集，2019年。[Surynek，2013] Pavel Surynek。在高度并行的运动协调问题解中消除冗余。International Journal on Artificial Intelligence Tools (IJAIT)，2013年。[Van Den Berg 和 Overmars，2005] Jur P Van Den Berg 和 Mark H Overmars。多机器人的优先级运动规划。载于 IEEE/RSJ 智能机器人与系统国际会议（IROS）论文集，2005年。[Vedder 和 Biswas，2021] Kyle Vedder 和 Joydeep Biswas。X*：通过基于窗口的迭代修复，在稀疏域中进行任意时长多智能体路径规划。Artificial Intelligence (AIJ)，2021年。[Wagner 和 Choset，2015] Glenn Wagner 和 Howie Choset。多机器人路径规划的子维度扩展。Artificial Intelligence (AIJ)，2015年。[Wurman 等人，2008] Peter R Wurman、Raffaello D’Andrea 和 Mick Mountz。在仓库中协调数百辆协作式自主车辆。AI magazine，2008年。[Yu 和 LaValle，2013] Jingjin Yu 和 Steven M LaValle。图上最优多机器人路径规划的结构与难解性。载于 AAAI 人工智能会议（AAAI）论文集，2013年。

附录

LaCAM 的 MAPF 特定部分

算法 5 展示了 LaCAM 的完整版本（算法 1）。为突出差异，伪代码将与算法 1 相同的行以灰色显示。下文将补充算法 5 中多智能体路径规划特有的部分。

约束。 请注意，LaCAM 中的约束由各个领域定义。在多智能体路径寻找中，约束指定下一个配置中哪个智能体位于何处。因此，约束树中的节点 C 指定下一个配置中多个智能体的位置。配置生成器必须遵循此规范。

约束生成。 为确定要约束哪个智能体，高层节点 N 包含顺序，即根据特定标准排序的所有智能体枚举列表。此顺序由两个函数指定：get_init_order（第 2 行）和 get_order（第 17 行）。随后，根据顺序以及约束树中低层节点的深度选择智能体 i，该深度由函数 depth（第 10 行）获取。这样可以确保约束树的每条路径上都没有重复的智能体。对 $N.$配置$[i]$ 中所有可能的位置生成约束（第 11–13 行）。当低层节点的深度等于 $|A|$时，约束树不会继续展开（第 9 行），因为该节点中所有智能体都已有约束。

算法 5 LaCAM

输入: 多智能体路径查找实例

output: 所以luti开启或NO SOLUTION_

前言: $C^{init} = \langle parent \perp who \perp where \perp \rangle$: : , : , : :

1 $Open$ $Explored$ ▷

初始化，栈，哈希表2: $N^{init} \leftarrow \langle config$: $S, tree$: $[\![C^{init}]\!]$, $parent$: $\perp$, $order$ _ _ $\rangle$: get init

3 $Open.$ $(N$ $)$ $Explored[S] = N$

order(): push init; init4: while $Open \neq \emptyset$ do5: $N \leftarrow Open.$ top()6: if $N.config = G$ then return 回溯(N)7: if $N.tree = \emptyset$ then $Open.$pop(); continue8: $C \leftarrow N.tree.$ pop()9: if 深度(C) $\leq |A|$ then ▷low_level_扩展

10 $i \leftarrow N.order[$ C]: 深度(): for neigh($N.config[i]$)

11 $u \in$

$\cup \{N.config[i]\}$ do

12 $C^{new} \leftarrow \langle parent$ C who i $where$ $u \rangle$: : , : , : :

13 $N.tree.$ C^{new}

push()14: $Q^{new} \leftarrow$ 配置_generator(N, C)15: if $Q^{new} = \perp$ t hen continue16: if $Explored[Q^{new}] \neq \perp$ then continue17: $N^{new} \leftarrow \langle config$: $Q^{new}, tree$: $[\![C^{init}, parent$: N, $order$ _ Q^{new} N $\rangle]\!]$: get order(,):

18 $Open.$ N^{new} $Explored[Q^{new}] = N$

push(); new19: return 无解_

B 评估

B.1 基线方法

我们测试了多种具有代表性或最先进的多智能体路径寻找（MAPF）算法，这些算法在可解性和最优性方面具有不同的特性，具体如下。另请参见图 1。

- **A*** [Hart 等人， 1968]作为一种基础搜索算法。它是完备且最优的。所用目标为最大完成时间（A*-m）和损失总和（A*-l）。

•A* 采用算子分解（OD） [Standley，2010] 将通用搜索改编用于多智能体路径寻找。该算法采用贪心最佳优先搜索，以尽可能多地获得解。启发式函数是到目标的距离之和。

- **算子分解rM*** [Wagner 和 Choset， 2015] 作为一种最先进的最优完备算法，类似于 A*。使用的目标函数为损失总和（算子分解rM*-l）和最大完成时间（算子分解rM*-m）。实现来自 https:// github.com/gswagner/mstar_public。原始实现使用损失总和作为目标函数。最大完成时间版本由此改编而来。

•膨胀的算子分解* (I-ODrM*) [Wagner 和 Choset， 2015] 作为一种最先进的有界次优完备算法，是 ODrM* 的变体。使用的目标函数是最大完成时间（I-ODrM*-m）和损失总和（I-ODrM*-l）。次优性设为五，以尽可能多地找到解。

- **BCP** [Lam 等人， 2022] 作为一种最先进的最优求解器，通过将问题转化为数学优化问题来实现求解。BCP 是解完备的，也就是说，它无法区分无解的实例。其实现..... 该实现使用 CPLEX 进行数学优化，代码取自 https://github.com/ed-lam/bcp-mapf。按照作者的实现，采用的目标是流动时间（又称代价和）。

- **CBS** [Sharon 等人， 2015] 采用了 Li 等人在 2021b], 中提出的多种改进技术，是一种最先进的最优求解器。它是两层组合搜索的代表。CBS 是解完备的。实现代码取自 https://github.com/Jiaoyang-Li/CBSH2-RTC。采用的目标包括流动时间（CBS-f）、最大完成时间（CBS-m）和损失总和（CBS-l）。原始实现使用流动时间作为目标函数。最大完成时间和损失总和版本均由此改编而来。

- **EECBS** [Li 等人于 2021c] 提出的 EECBS 是一种最先进的有界次优但解完备算法，也是 CBS 的一个变体。采用的目标包括流动时间（EECBS-f）、最大完成时间（EECBS-m）和损失总和（EECBS-l）。次优性设为五。实现代码取自 https://github.com/Jiaoyang-Li/EECBS。原始实现使用流动时间作为目标函数。最大完成时间和损失总和版本均由此改编而来。

•优先规划（PP） [Erdmann 和 Lozano-Perez， 1987；Silver， 2005] 作为多智能体路径寻找的基础方法。该实现首先使用距离启发式 [Van Den Berg 和 Overmars， 2005] 确定规划顺序。此外，它会以随机优先级反复执行优先规划，直到问题已求解。该实现改编自 https://github.com/Kei18/pibt2。

• **MAPF-LNS2（LNS2）** [Li 等人， 2022] 这是一种基于大邻域搜索的先进次优且不完备求解器。其实现代码取自 https://github.com/Jiaoyang-Li/MAPF-LNS2。

• **PIBT** [Okumura等， 2022], 这是一种不完备且次优的算法。由于 LaCAM 使用了 PIBT，因此测试了原版 PIBT。为检测规划失败，当 PIBT 达到预先设定的足够大的时间步数时，便将其视为失败。实现代码取自 https://github.com/Kei18/pibt2。

• **PIBT**$^+$ [Okumura 等， 2022], 这是一种最先进的可扩展多智能体路径规划求解器，但并不完备且为次优算法。它使用推送与交换 [Luna 和 Bekris， 2011]来补充解。

• **LaCAM** [Okumura， 2023], LaCAM* 基于此算法。这是一种完备的次优算法。它使用原版 PIBT 作为配置生成器。实现代码取自 https://github.com/Kei18/lacam。

• **AFS** [Cohen 等，2018a], 这是 CBS 的 anytime 版本，最终会收敛到最优解。其实现由论文作者提供。原始实现使用流动时间作为目标函数。损失总和版本则由此改编而来。

B.2 小型复杂实例的损失总和

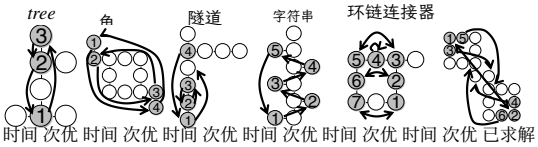
表 7 展示了小型复杂实例在损失总和方面的结果，对应表 4。总体而言，结果与最大完成时间方面的结果相似；许多求解器在某些实例上未能求解，或牺牲了解质量，而 LaCAM* 则立即找到初始解，并收敛到（近似）最优解。

B.3 多智能体路径寻找基准测试

如何测试各算法。 每次将智能体数量增加 50 后，如果某些算法在上一轮中未能求解所有实例，我们便停止测试这些算法。

结果。 图 10 至图 12 展示了多智能体路径寻找基准测试的完整结果，补充了图 8。除损失总和外，我们还展示了以 $\max_{i\in A} \mathrm{dist}(s_i, g_i)$ 归一化的最大完成时间。LaCAM* 的得分对应初始解。总体而言，LaCAM* 在数秒内求解了各种问题实例，这些实例的图规模、智能体数量、稀疏度和复杂度各不相同。LaCAM* 的解质量与其他次优算法相当。具体而言，其解质量与 PIBT 和 LaCAM 相似，因为它基于这些算法。

其他算法的讨论。 尽管近年来多智能体路径寻找研究取得了显著进展，但最优算法（例如 ODrM*、BCP、CBS）的可扩展性有限；在大多数情况下，它们无法处理数百个智能体。有界次优算法，如 I-ODrM* 和 EECBS，可以解决各种实例，但在解决具有挑战性的实例（例如包含1000个智能体的实例）时仍有困难。次优算法，如 PP、LNS2 或 PIBT$^{(+)}$ 有时能够处理这类具有挑战性的实例，但仍有许多实例无法解决。从另一个角度来看，理想的算法应当是完备的，



	时间	次优	时间	次优	时间	次优	时间	次优	时间	次优	时间	次优	已求解	
LaCAM*	0	1.25	0	1.12	0	1.46	0	2.36	2	7.12	0	(1.48)		
1 秒后	0	1.00	13	1.00	77	1.00	8	1.00	700	1.44	408	(1.08)		**6/6**
A*	0	1.00	149	1.00	35	1.00	25	1.00	15124	1.00	不适用		不适用	5/6
算子分解rM*	3	1.00	40	1.00	675	1.00	0	1.00	不适用	不适用	不适用		不适用	4/6
I-算子分解rM*	0	1.00	0	1.31	257	1.08	0	1.00	不适用	不适用	124	(1.39)	5/6	
CBS	70	1.00	0	1.00	不适用	不适用	180	1.00	不适用	不适用	不适用	不适用	3/6	
EECBS	2	1.00	0	1.00	不适用	不适用	0	1.00	不适用	不适用	86	(1.61)	4/6	
OD	0	1.00	0	1.50	14	2.57	0	1.20	2133	30.62	5	(1.38)	**6/6**	
LaCAM	0	1.23	1	1.69	92	1.91	0	3.30	5519.15		0	(1.45)	**6/6**	
PP	不适用	不适用	0	1.00	不适用	不适用	0	1.00	不适用	不适用	不适用	不适用	2/6	
LNS2	不适用	不适用	0	1.00	不适用	不适用	0	1.00	不适用	不适用	29	(1.00)	3/6	
PIBT	不适用	不适用	不适用	不适用	不适用	不适用	不适用	不适用	不适用	不适用	不适用	不适用	0/6	
PIBT$^+$	04.38		01.12		03.91		0	2.20	不适用	不适用	0	(1.68)	5/6	
BCP	194		- 150		- 不适用		- 117		- 不适用		- 不适用		- 3/6	

表 7：小型复杂实例的损失总和结果。时间单位为毫秒。另请参阅表 4 的标题。关于 连接器 （以括号标注）的损失总和，由于未能获得最优分数，因此呈现按已知最佳值（80）归一化的分数。

然而，完整性可能成为实现速度的瓶颈。这一现象在 A*、算子分解和 ODrM* 中均有所体现。

一般而言，获得最优完工时间解比获得损失总和最优解更容易。这是因为，对于给定的实例，最优完工时间解的数量多于损失总和最优解的数量。图 1 中通过 ODrM* 和 CBS 的实验结果验证了这一点。例如，CBS-m 求解的实例数量远多于 CBS-l。与此同时，我们可以在 I-ODrM* 中观察到相反的趋势。I-ODrM* 是一种有界次优完备算法；I-ODrM*-l 求解的实例数量多于 I-ODrM*-m。我们认为这是可采纳启发式函数所产生的影响。损失总和优化使用 $\sum_{i\in A} \mathrm{dist}(s_i, g_i)$，而最大完成时间优化使用 $\max_{i\in A} \mathrm{dist}(s_i, g_i)$；前者比后者提供了更多的信息指导。

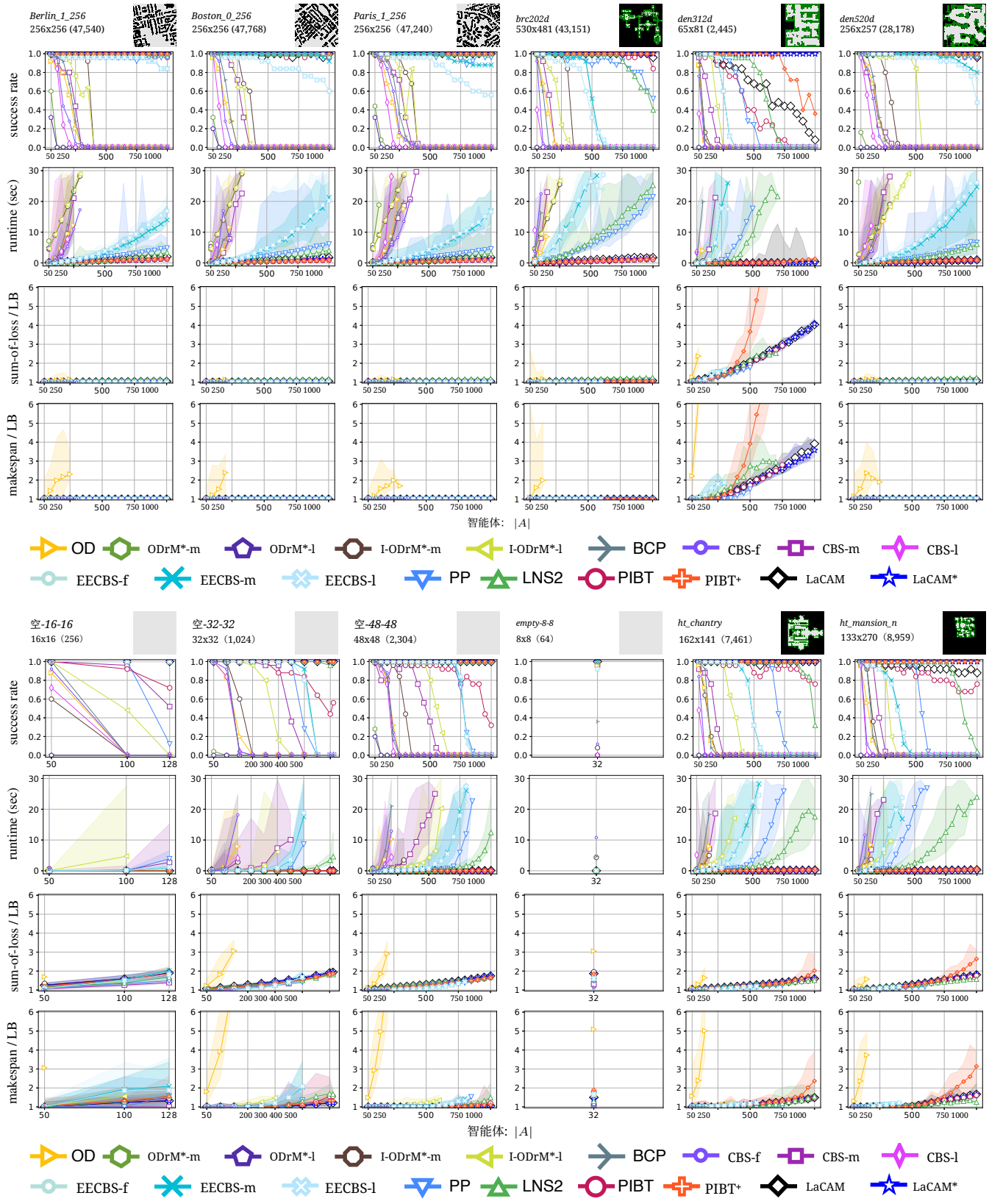


图 10：MAPF基准测试结果（1/3）。另请参见图 8 的图注。 $|V|$ 显示在括号中。

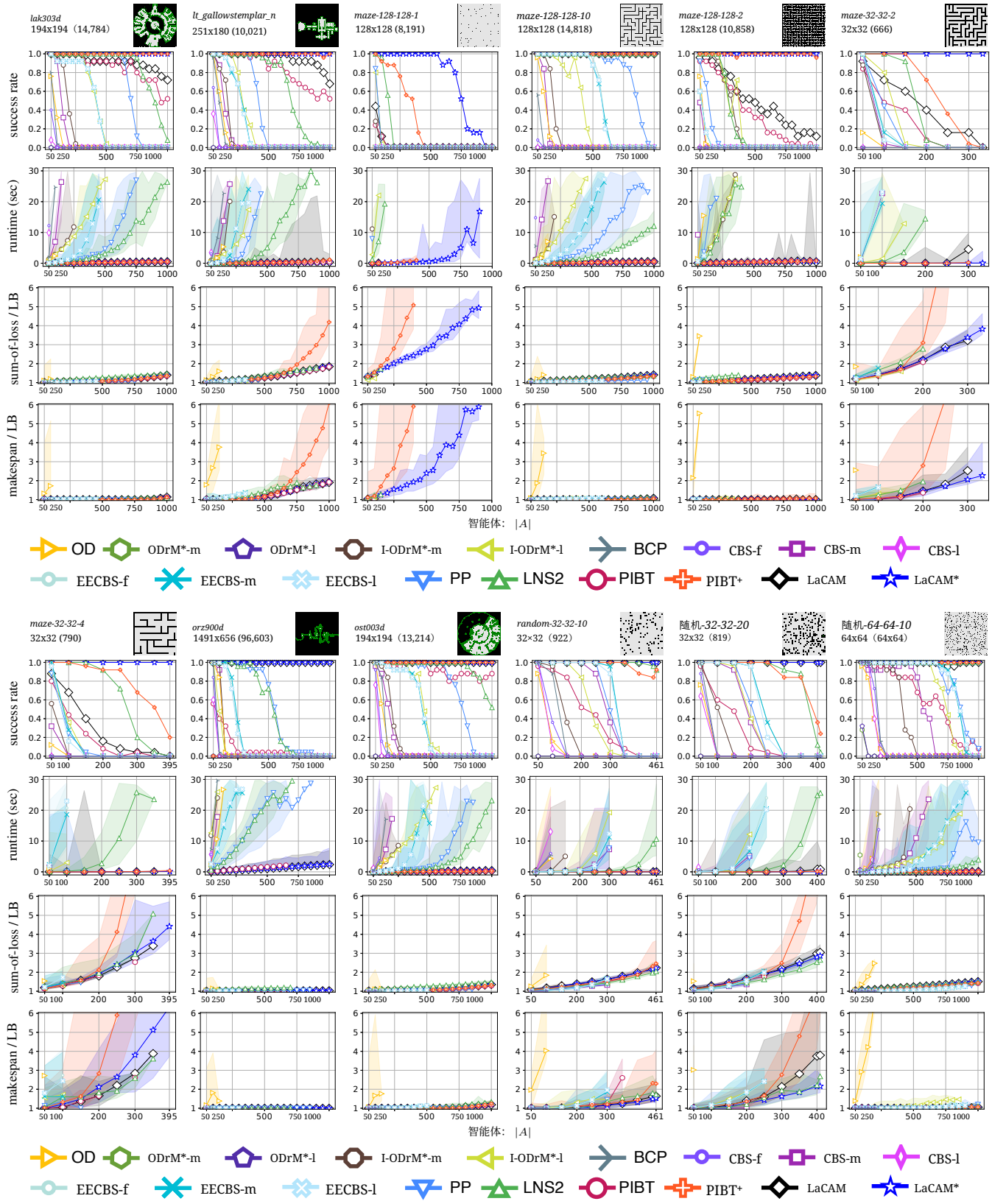


图 11：MAPF基准测试的结果（2/3）。另请参阅图 8 的说明。

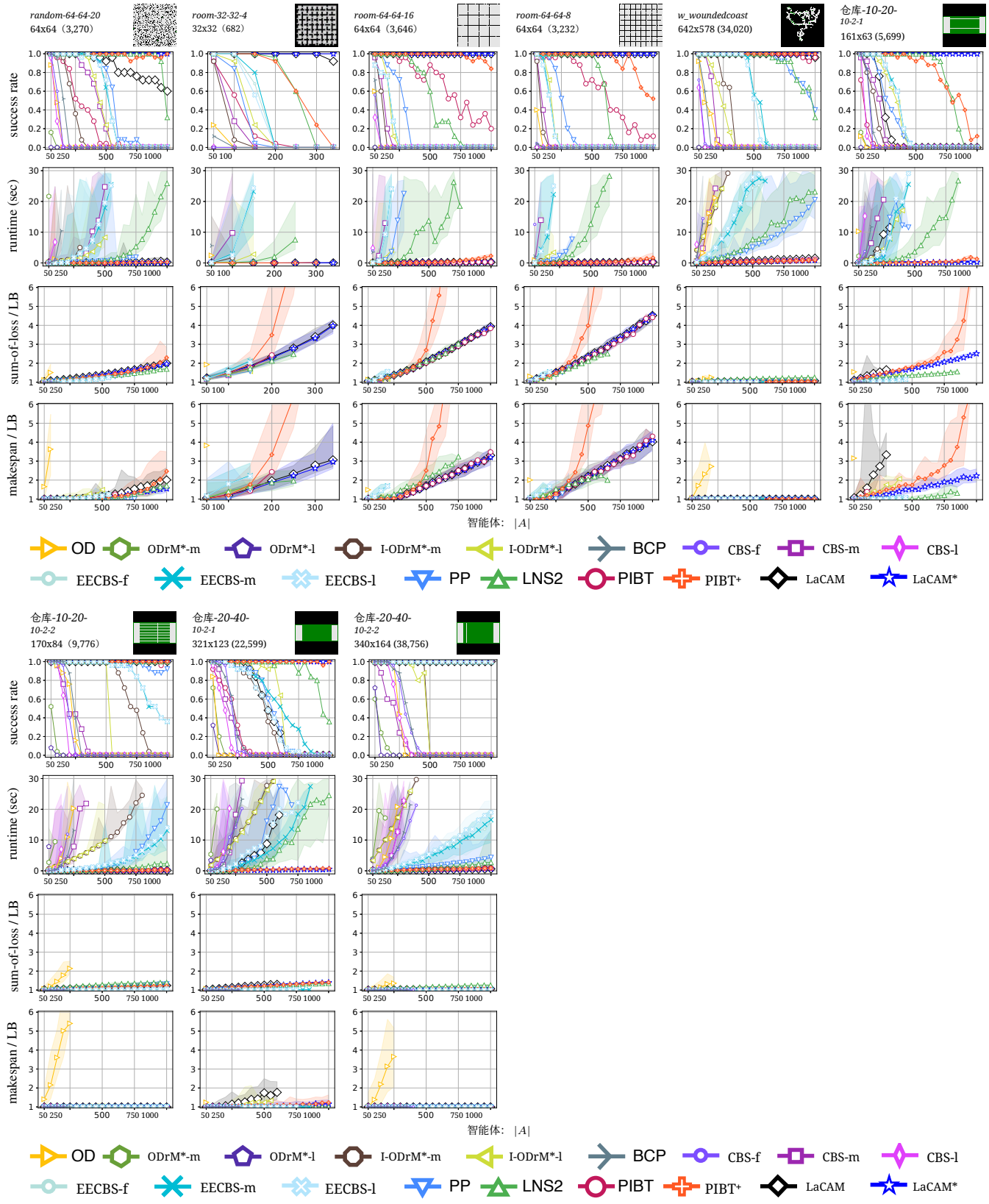


图 12：MAPF基准测试结果（3/3）。另请参见图 8 的说明。