采用异步执行与局部导航的多智能体取货与配送问题分布式规划

Yuki Miyashita
Shimizu Corporation 东京，日本
y-miyashita@shimz.co.jp

Tomoki Yamauchi 早稻田大学 东京，日本
t.yamauchi@isl.cs.waseda.ac.jp

Toshiharu Sugawara 早稻田大学 东京，日本
sugawara
@isl.cs.waseda.ac.jp

arXiv:2302.09250v1 [cs.MA] 18 Feb 2023

摘要

我们提出了一种针对环境中智能体活动偶尔出现延迟且端点灵活的多智能体取货与配送（MAPD）问题的异步执行分布式规划方法。MAPD 是一个具有众多应用的重要问题框架；然而，现有大多数研究都假设智能体行为和环境是理想的，例如智能体速度固定、移动同步，以及环境经过精心设计，包含许多较短的绕行路径，以便多个智能体轻松执行任务。然而，这样的环境往往难以实现；例如，智能体的移动速度可能受到天气和地面状况的影响，并且经常出现延迟。所提出的方法通过允许智能体动作的速度发生波动，以及允许工作位置（端点）灵活变化，放宽了一些难以实现的条件，从而能够在更现实的环境中应用 MAPD。实验结果表明，与基线方法相比，我们的方法能够使智能体在此类环境中高效地执行 MAPD。我们还分析了使用该方法的智能体的行为，并讨论了其局限性。

关键词

多智能体路径规划，多智能体取货与配送问题，分布式机器人规划

1 引言

多智能体寻路（MAPF）是指多个智能体通过避免碰撞移动至各自目的地，这是机器人学和游戏等诸多应用中出现的一个重要抽象问题 [8, 17, 19]。例如，在自动化仓库中，机器人（搬运智能体）移动到取货地点，装载材料，并将其运送至各自的卸货地点，期间不断为智能体连续分配新任务。这类反复求解MAPF问题的应用被表述为多智能体取货与配送（MAPD）问题 [13]。因此，MAPD的目标是使智能体在无碰撞的情况下，反复移动至各自的任务端点，即取货和配送地点 [13], 。

然而，复杂且具有诸多限制的环境可能会降低效率，从而增加发生碰撞的可能性。此外，实践中往往存在不确定性。例如，某些智能体移动时偶尔出现的延迟会影响其他智能体的规划。因此，许多智能体无法按照此前确定的规划移动。在这种情况下，这些智能体必须重新规划路径以避免碰撞。然而，当使用集中式方法，或在采用去中心化控制时假设移动同步进行，随着智能体数量增加，重新规划需要付出相当大的计算成本。与此同时，我们设想了在建筑工地和灾害区域进行机器人材料转运的应用，而此类不确定性是无法避免的。因此，我们专注于一种分布式规划与执行方法：智能体分别生成各自的规划，并仅与相关的本地智能体进行交互，移动至各自的目的地，以应对延迟和资源冲突带来的负面影响。尽管多智能体取放任务的分布式规划仍有一些问题需要解决，例如完备性以及避免活锁和死锁，但我们认为，分布式规划方法更为可取，因为它具有鲁棒性、可扩展性和可重构性等理想特性 [28]。

尽管已有一些关于MAPF和MAPD规划与执行的研究，但大多数研究都假设环境类似网格，例如自动化仓库，其中（1）端点数量多于智能体数量，并且（2）智能体以恒定速度移动且同步移动 [13]。此外，这些传统方法要求满足良构基础设施条件（WFI条件）[23]。例如，在保持任务端点（HTE）[13],中，连接一对端点的任何路径都不会经过其他端点。这类要求在简单的网格状环境中很容易满足，但在我们的目标应用中并不可行。例如，在建筑工地，重型机器人会在夜间将重达500千克及以上的建筑材料，在存储位置、特定工作位置和/或将其运送至其他楼层的电梯之间进行运输，以供第二天施工使用；这些端点（即存储位置和工作位置）的数量并不多。因此，多个智能体很可能会同时以少数特定端点作为目的地移动，但又不能彼此发生碰撞。此外，（1）由于工作位置多种多样，WFI条件不一定能够满足；（2）由于温度、湿度、湿地面、斜坡/小台阶以及传感器误差等各种原因，智能体通常也无法以恒定速度移动。

我们的贡献有两点。首先，我们提出了具有波动移动速度的多智能体取放任务（MAPDFS）问题，这是对目标应用的抽象：在建筑工地由多个机器人执行材料运输任务，同时考虑上述特征，例如端点数量较少以及智能体移动速度不确定性。

其次，我们提出了一种面向MAPD问题的新型分布式规划与异步执行方法。该方法的特点是：无需考虑其他智能体的规划即可进行低成本规划，并基于在由若干双连通分量和小型树状区域组成的图所描述的环境中对潜在冲突进行检测，从而实现多个智能体的相邻移动。我们在MAPD中引入了两类智能体：搬运智能体，简称智能体，它们在规划路径的同时搬运材料

，无需考虑其他智能体；以及节点智能体，它们管理图中相应的节点，并仅通过与相邻节点通信来检测发生冲突的可能性。搬运智能体规划出路径后，会与管理当前节点的节点智能体通信，随后该节点智能体请求下一个节点智能体预留该节点。根据响应，当前节点智能体会向搬运智能体建议其移动到下一个节点、移动到另一个节点（即进行绕行），或在当前节点等待一段时间。随后，搬运智能体会在不考虑其他智能体的情况下异步移动或等待。此外，当它移动到当前规划中未包含的另一个节点时，会忽略其他智能体并重新规划路径。

忽略其他智能体的基本理念在于这样一个事实：在可能存在不确定性的真实环境中，带有严格执行的详细规划很容易变得不可行。相反，为了减少碰撞，我们对环境进行了定向设计，使其主区域能够被视为一个强连通图。随后，利用有向边根据方向引导智能体，从而消除迎头碰撞以及死锁/活锁，例如环岛中的情况。这可能会迫使智能体进行绕行；然而，这简化了独立规划过程，使潜在碰撞更容易被检测，允许智能体延迟/停止移动，并在端点数量较少的情况下放宽WFI条件。这些特性对于现实世界中的应用不可或缺。随后，我们通过将所提出的方法与两种基线方法滚动时域碰撞消解（RHCR） [9] 和 HTE [13]进行比较，对所提出方法的性能进行了实验评估。我们发现，在满足这些方法所需条件的环境中，我们所提出的方法优于基线方法。我们还展示了，尽管智能体的速度偶尔会出现延迟，并且端点位置具有灵活性，但采用我们方法的智能体仍能无碰撞地完成所有任务。

2 相关工作

尽管寻找无碰撞路径是多智能体路径寻找/多智能体取放任务的基本要求，但这被认为是一个NP难问题 [29],，涉及较高的计算成本。因此，一些研究尝试通过放宽问题限制来寻找次优解 [2, 3, 6]。

通常通过集中式规划器反复进行路径搜索，将多智能体路径寻找整合到多智能体取放任务中 [4, 7, 10, 15]。例如，Nguyen 等人 [15] 使用答案集编程解决了任务分配和路径搜索问题。Cohen、Uras 和 Koenig [4] 扩展了基于冲突的搜索（ CBS ） [20]，加入了一组具有用户指定方向的边，称为高速通道，通过避免智能体朝相反方向行进来提供避免碰撞的引导。高速通道 还有助于降低仓库中自主机器人的多智能体路径寻找求解器的运行时间和解的成本。然而，由于智能体数量增加会带来较高的计算和通信成本，这些集中式规划器难以扩展。我们的方法可视为高速通道 与分布式规划和执行的结合，因为我们也会在环境中以算法方式为边设定方向，以便在局部引导智能体。

其他次优的集中式规划方法 [9, 21, 25] 将迭代式多智能体路径寻找分解为顺序路径规划问题，在该问题中，求解器会在每个时间步重新规划所有路径。由于这些方法的计算成本较高，一些研究尝试提升其可扩展性。Švancara 等人 [21] 研究了在线多智能体路径寻找，其中新的智能体目标可以动态添加，智能体到达目标后即可消失。随后，他们提出了在线独立性检测算法，用于生成规划方案，以减少对现有规划方案的干扰。Li 等人 [9] 提出了 RHCR，在该方法中，智能体会在 w个时间步的窗口内寻找发生的冲突，并每隔 h 个时间步重新规划路径。然而，如果环境密集且拥挤，则必须检查其他智能体的大量路径才能生成无碰撞路径，从而导致较高的计算成本。

无碰撞路径的去中心化规划也已有研究 [12, 13, 16, 25, 27]。例如，Ma等人 [13] 提出了HTE，这是一种优先级路径规划方法，其中智能体依次规划各自的路径，并要求新路径与其他智能体已有的路径保持无碰撞。带回溯的优先级继承（PIBT） [16] 基于优先级规划，在较短的时间窗口内关注多个智能体的相邻移动。这些方法要求环境满足特定条件，以保证多智能体取放任务实例的完备性。HTE要求满足WFI条件，而PIBT要求环境为双连通图。尽管这些环境条件适用于某些应用（例如自动化仓库中的多智能体取放任务），但其他应用（例如建筑工地和一些物流分拣中心 [24]）的环境无法满足这些条件。此外，在此类去中心化方法中，智能体被迫同步移动，因此无法处理移动速度的波动。

与此同时，我们认为分布式方法更适合我们的应用，因为智能体可以异步移动，并提高可扩展性和鲁棒性。然而，这种方法中的方法有时并不完备。例如，Wilt 和 Botea [26] 提出了一种空间分布式规划器，其中每个控制器智能体负责管理一个子区域，并与相邻的控制器通信，以确认将移动单元转移到另一个子区域。然而，为了使该方法完备，它采用了部分集中式的设计。因此，其环境条件有些不灵活，智能体也无法异步移动。Miyashita、Yamauchi 和 Sugawara [14] 提出了一种支持智能体异步移动的分布式规划方法。他们的方法引入了与我们类似的两类智能体：搬运智能体，负责搬运材料；以及节点智能体，负责管理资源冲突。然而，他们的方法实现起来非常复杂。我们的方法采用了类似的思路，但更加简单且成本更低，即使在拥挤区域，智能体也能有效移动。

3 问题形式化

3.1 多智能体取放任务

在MAPD问题中，搬运智能体（或简称为智能体）会在指定位置装载物料，将其运送至指定目的地并卸载。设 $I = \{1, \cdots n\}$ 为智能体集合，$\mathrm{T} = \{\tau_1, \cdots \tau_{|T|}\}$为一组应由智能体完成的任务。环境由一个可嵌入二维空间的连通图 $G = (V, E)$描述，其中$V = \{v_1, \cdots v_{|V|}\}$为对应于位置的节点集合，而 E 为对应于两个节点之间连接的边集合

相邻节点。我们的图由间接边和有向边共同构成，因此，连接 v_j 和 v_k 的边可表示为 $(v_j, v_k) \in E$（间接边），或表示为 $(v_j \to v_k) \in E$（有向边）。因此，$(v_j \to v_k)$ 表示智能体只能从 v_j 移动至 v_k 的路径，而在 (v_j, v_k) 上，智能体可以在 v_j 和 v_k 之间双向移动。注意，我们假设智能体 i 不能停留在边上。仅当两个智能体位于同一节点上，或沿同一条边向相反方向移动时，它们才会发生碰撞；这意味着在有向边上不会发生碰撞。

任务 $\tau_k \in \mathrm{T}$ 由元组 $\tau_k = (v_k^l, v_k^u, \mu_k)$ 指定，其中 μ_k 是要搬运的材料，v_k^l 是用于装载 μ_k 的取货节点，v_k^u 是用于卸载 μ_k 的目标节点。智能体装卸材料的节点称为任务端点。我们引入离散时间 $t \geq 0$，其单位为时间步。当智能体 $i \in I$ 在时间 t 被分配任务 $\tau_k \in \mathrm{T}$ 时，它会独立规划：生成一条从节点 $v^i{}_c(t)$ 到 v_k^l 以装载 μ_k 的路径，以及一条从 v_k^l 到 v_k^u 以卸载它的路径，其中 $v^i{}_c(t)$ 是 i 在 t 时所在的节点。随后，i 沿着路径移动到 v_k^l 和 v_k^u，并装载/卸载 μ_k。如果可能发生碰撞，i 必须修改路径以避免碰撞。

最初（$t = 0$），智能体 i 从其专属停车节点 $v_{pk\,i} (= v^i{}_c(0))$ 出发，然后与其他智能体一起开始执行 T 中给定的任务。如果可能，智能体可以在 T_{mv} 个时间步内移动到一个相邻节点，但装卸可能需要更长的时间 $T_{lu} (\geq T_{mv})$。它也可以在节点处等待任意时间步。我们可以假设智能体速度存在波动；也就是说，它有时需要比 T_{mv} 更长的时间才能移动到相邻节点。当 i 完成当前任务后，T 中的一个元素会被分配给它。当 $\mathrm{T} = \varnothing$ 时，i 返回其停车节点。我们将端点定义为任务端点或停车节点。

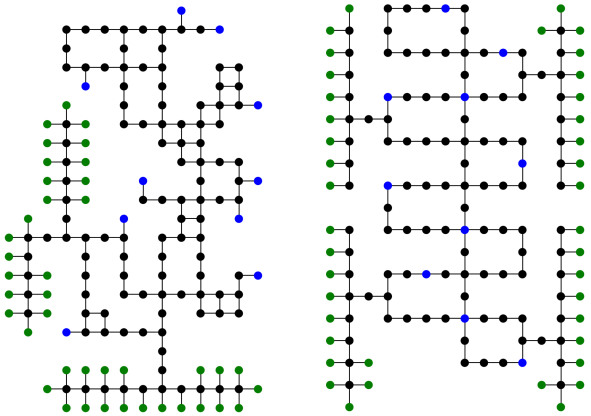
3.2 所需的环境条件

我们假设环境 $G = (V, E)$ 由一些带有小型树的双连通分量组成。下面给出其更正式的定义。设 G 是一个无向图，$G' = (V', E')$ 是与 $V' \subset V$ 相关联的 G 的一个子图，即 $(v_1, v_2) \in E'$ 当且仅当 $v_1, v_2 \in V'$ 且 $(v_1, v_2) \in E$。

定义 *3.1*。（双连通分量）
(a) 若对于 $\forall v_1, v_2 \in V'$，G' 包含一个连接 v_1 和 v_2 的环，则 G' 是 G 的一个双连通子图。(b) 若不存在 $\check{S}v \in V \setminus V'$，使得与 $V' \cup \{v\}$ 相关联的 G 的子图是双连通子图，则 G 的双连通子图 G' 是极大的。(c) G 的极大双连通子图称为双连通分量。

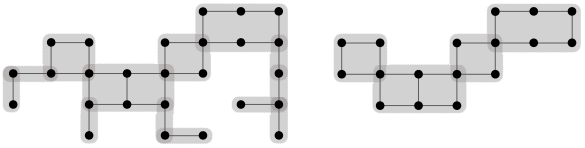
令 G_1、……、G_K 为 G 的所有双连通分量。我们记作 $G_k = (V_k, E_k)$。我们将 $G_{主} = G_1 \cup \cdots \cup G_K$ 称为 G 的主区域，其中图的并集定义为 $G_k \cup G_{k'} = (V_k \cup V_{k'}, E_k \cup E_{k'})$。我们假设环境 G 满足以下结构（SC）和智能体（AC）条件：

SC1. $G_{主图}$ 是一个连通图。SC2. 对于 $\forall v \in V \setminus V_{主图}$，$v$ 是 L 个树结构子图中的一个节点，即 G、$G^1{}_{树}, \ldots,$ $G^L{}_{树}$ 中的一个节点；对于 $1 \leq k \leq L$、$G^k{}_{树} \cap G_{主图}$ 是一个单元素集合，其元素是 $G^k{}_{树}$ 的根节点。



（a）环境1（环境1）　（b）环境2（环境2）

图1：示例环境。



（a）图结构。　（b）主区域。

图 2：示例环境。

SC3. 停车节点是树结构子图中的末端节点，且这些子图不包含任务端点。
AC1. 智能体数量至少比主区域中的节点少两个，$|I| \leq |G_{主区域}| - 2$，但为提高效率，我们建议智能体数量少于 $|G_{主区域}|$ 的一半。该主题将在后文讨论。

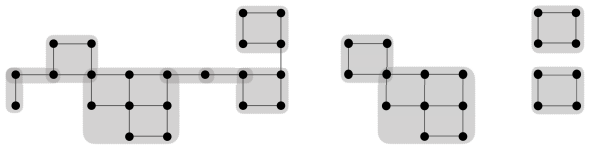
我们将所有根节点的集合记为 $G_{根节点} = G^1{}_{树} \cup \cdots \cup G^L{}_{树}$。主区域。将边缘区域定义为 $G_{边缘区域} = G \setminus G_{根节点}$，$G = G_{主区域} \cup G_{边缘区域}$ 显然是不交并。于是，下面的命题成立。

命题 1。假设 $G_{主区域} = G_1 \cup \cdots \cup G_K$ 是连通图，且 G_k 是双连通分量。如果 $V_k \cap V_{k'} \neq \varnothing$，则 $V_k \cap V_{k'}$ 是单节点。

证明。如果 $V_k \cap V_{k'}$ 有两个节点，$V_k \cup V_{k'}$ 是双连通的；因此，V_k 和 $V_{k'}$ 不是双连通分量，这与命题矛盾（详细证明见附录）。□

我们可以将任务端点设置在 G 中的任意节点上，但须满足条件 SC3：如果智能体在主区域的节点上装载或卸载材料，其他智能体的移动可能会暂时受到阻碍。然而，由于建筑工地上的任何位置都可能成为作业地点，我们认为这种情况不可避免，智能体应将所需材料运送到这些地点。

图 1 展示了我们的环境示例，其中（四十个）绿色圆点是单个智能体的停车节点，蓝色圆点是任务端点（即装卸位置）。图 2 展示了另一个示例；图 2a 展示了环境，而



(a) 图　　(b) 主区域不连通。

图3：不满足条件的图。

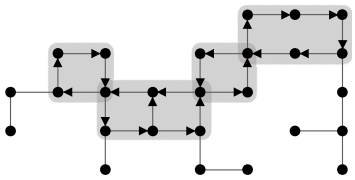


图4：图定向示例。

图2b 是其主区域。图 3 展示了不符合我们条件的图：实际上，主区域并不连通（图3b），并且其中一些节点G_{mar} 不是树中的节点。

4 所提出的方法

所提出的算法的一个特点是忽略其他智能体的当前计划，从而进行简单规划。相反，对于$\forall v \in V_{\text{主区域}}$，我们引入节点智能体，它管理 v 的资源并检测发生冲突的可能性，即智能体之间发生碰撞的可能性。虽然（载体）智能体只与管理当前节点资源的当前节点智能体 通信，但节点智能体会与相邻节点智能体通信，并检查智能体在自身节点上移动的可能性。下文也将节点智能体表示为 v。载体智能体独立且异步地生成一条路径，即由一系列节点组成的通往目的地的路径，并在沿路径移动到下一个节点的同时，通过询问当前节点智能体来确认其他智能体不会停留在下一个节点，也不会向其移动。

请注意，节点智能体不一定位于相应节点上，因为它们只管理相应节点的预留。因此，它们可以运行在单台机器上、云中的不同服务器上，或运行在位置附近的智能传感器上。唯一的要求是，节点智能体应能够与管理相邻节点的节点智能体，以及当前正在预留该节点的载体智能体通信。

4.1 通过可达性进行图定向

我们的基本想法是在主区域引入强定向，以避免经过同一条边，尤其是避免在由多条边组成的长直路径上沿相反方向移动。这可能会导致绕行，但反过来也能为移动流提供一致的结构方向。因此，智能体只需利用局部信息和资源分配，就能避免碰撞，并以机会主义的方式应对行程延误或突然停止。这也消除了为考虑其他智能体的路径并通过协商避免碰撞而进行高成本规划的必要。

首先，我们介绍与图论相关的基本概念。

定义*4.1*。 如果任意一对节点之间都存在两个方向的路径，则称有向图是强连通的。

定义*4.2*。 在无向连通图中，如果删除某条边后图将不再连通，则称该边为桥。

显然，我们的主区域$G_{\text{主区域}}$是无桥（或*2-*边连通）且连通的。因此，下面的定理被称为单向街道定理 [18]。

定理4.3。 无桥连通无向图可以通过一致地确定边的方向转化为强连通图（反之亦然）。

已有多种高效算法（线性时间和对数时间 $|E|$）可将无桥连通无向图定向为强连通图 [1, 5, 22]。我们使用其中一种算法对 $G_{\text{主}}$ 进行定向。请注意，G_{mar}中的边仍保持无向，这意味着智能体可沿双向边向两个方向行驶。图 2a 所示定向环境的示例如图 4 所示。

4.2 载体智能体的行为

我们假设环境 G已按照上一节所述完成定向。当任务 $\tau_k = (v_p^k, v_d^k, \mu^k)$ 被分配给载体智能体 i时， i 将移动到其装载位置 v_p^k，再移动到其卸载位置 v_d^k。因此， i 会根据任务进展阶段，在 τ_k中将目标节点 $v_{\text{dst}} \in V$设为 v_p^k或 v_d^k，然后使用传统方法（例如在（部分）有向图 G中采用 A^*-搜索）生成从当前节点 v^i_{c} 到 v_{dst} 的最短路径 p_i （或者从另一角度看，生成适当的路径）。在此，我们将从节点 v到节点 v'的路径 p定义为节点序列 $p = (v_0^p, v_1^p, \ldots v_{|p|}^p)$，其中 $v_0^p = v$和 $v_k^p = v'$以及 v_{k-1}^p和 v_k^p 通过边（ v_{k-1}^p, v_k^p ）或（ $v_{k-1}^p \to v_k^p$ ）相连。请注意，与其他多智能体取放任务（MAPD）方法不同，我们可以忽略时间信息来生成路径，即忽略智能体到达和离开节点的时间。当生成路径 p_i后， i会按照 p_i尝试移动到 v_{dst}。

我们将智能体 i在 t处的当前节点记为 $v^i_{\text{c}}(t)$。智能体 i具有协调节点智能体（或简称协调者）$v^i_{\text{Fcl}}(t) \in V_{\text{main}}$；如果 $v^i_{\text{Fcl}}(t) = v^i_{\text{c}}(t) \in V_{\text{main}}$，则该协调节点与当前节点相同；如果位于树中（即 $\text{c}() \in \text{tree}$），则其协调节点设置为根节点 $v^i_{\text{Fcl}}(t) \in G^k_{\text{tree}} \cap G_{\text{main}}$。如果 $v^i_{\text{c}}(t)$ 和下一个节点 $v^i_{\text{next}} \in p_i$ 不在 V_{main}中， i 无需确认即可移动到 v^i_{next}。否则（即如果 $v^i_{\text{c}}(t) \in V_{\text{main}}$或 $v^i_{\text{next}} \in V_{\text{main}}$），在 $i$at $v^i_{\text{c}}(t)$移动到下一个节点 $v^i_{\text{next}} \in p_i$之前， i会向节点智能体 v^i_{Fcl}发送请求消息，以请求预留 v^i_{next} p_i。随后，它将从 v^i_{Fcl}收到回复。如果回复是接受消息， i会在 $t+1$离开当前节点前往 v^i_{next}，并释放对 $v^i_{\text{c}}(t)$的预留。注意，到达 v^i_{next}可能需要一段时间 $T_{\text{mv}} \geq 1$；然而，我们假设在 i离开之前， $v^i_{\text{c}}(t+1)$一直处于 $= v^i_{\text{next}}$状态。这也意味着，当 $T_{\text{mv}} > 1$时，它们的活动是异步的；也就是说，当一个智能体开始离开当前节点时，其他智能体可能已经位于边的中途。在 i到达下一个节点后， i会根据计划 p_i尝试预留下一个节点 v^i_{next}。

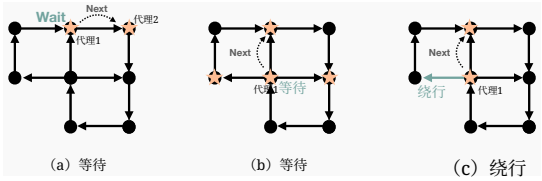


图5：使用 SOM 的拒绝消息示例。

与此同时，如果 i 收到节点智能体 v^i_{Fcl}针对预留 v^i_{next}发来的拒绝消息，该消息会包含移动建议（SOM）：等待（即等待一段时间），或选择与 $t_c(v^i)$相邻的、位于 V_{main} 中的另一个下一个节点进行绕行（即采取绕行路径）。当SOM为绕行时，i会前往指定的下一个节点，并使用传统算法从该节点到目的地生成另一条（最短）路径 p_i，而不考虑其他智能体规划的路径。注意，生成的新路径很可能包含曾被拒绝的 v^i_{next}；然而，G_{main} 中的所有边都是有向的，因此 i 应采取绕行路径返回 v^i_{next}，从而使其周围环境变得不同。

4.3 用于冲突检测的节点代理行为

节点智能体 $v \in V_{主}$要作为协调者，管理相应节点 v 为位于 v 的停留（承运）智能体 i 的预留。它通过与相邻节点通信来确定发生碰撞的可能性，从而确认 i 是否可以移动到下一个节点。这意味着可能有两个以上的智能体 $i,\ j \in I$ 同时尝试预留同一节点，但我们假设节点智能体会从其消息队列中逐一读取这些请求。

当节点智能体 v从位于 $v = v^i_c(t)$的智能体 i接收到移动到相邻节点 $v^i_{下一个} \in V_{主}$的请求消息时，它会通过向节点智能体 $v^i_{下一个}$发送预留消息，询问该节点是否空闲。如果 $v^i_{下一个}$在 $t+1$时未被其他智能体预留，$v^i_{下一个}$就会为i预留其资源，并且 $v^i_{下一个}$发送接受消息 v，该消息随后被转发给 i。如果 $v^i_{下一个}$已在 $t+1$被预留，或者位于 v^i的另一个智能体 j尚未决定在 $t+1$时移动到 $v^j_{下一个}$，则协调者 v会将来自 $v^i_{下一个}$的拒绝消息转发给 i，并附带可能的动作标签；该标签属于以下 SOM 之一：

等待： 节点智能体 v 建议 i 将当前停留时间延长至 $t+1$（因此该延长不一定是T_{mv}）。这始终是可行的，因为 v 只有在 i 已预留下一个节点后，才会接受来自另一智能体的预留请求。**绕行：** 假设 v 有多条向外的直接边。v 会依次向相邻节点发送预留消息，但不包括 $v^i_{下一个}$；如果其中一个 $\tilde{v}^i_{下一个}$ 接受该消息，v 会向 i发送带有绕行信息以及已预留节点 $\tilde{v}^i_{下一个}$ 的拒绝消息。

节点智能体 v 应根据情况选择 SOM、等待或绕行。两种 SOM 各有优缺点；等待 SOM 可能会阻塞其他智能体，而绕行 SOM 可能会迫使 i 进行绕行。然而，在我们的实验中，节点智能体 v 尝试发送绕行消息，但在无法执行时发送了等待消息，因为确保智能体的流动不受干扰有助于高效执行任务，尤其是在拥挤情况下。示例如图 5 所示；协调者智能体1在图 5a 和 b 中发送包含等待的拒绝消息，因为所有相邻节点都已被预留；而在图 5c 中，协调者发送包含绕行的消息，因为有两个相邻节点未被预留，并随机选择其中一个，尽管智能体1计划的下一个节点已被预留。

4.4 边缘区域中的碰撞检测

首先，假设树结构子图（区域）$G^k_{树}$ 不包含停车节点。如果节点智能体$v \in V_{根节点}$ 是 $G^k_{树}$的根节点，则 v还能够将进入 $V^k_{树} \setminus \{v\}$ 的智能体数量限制为一个。因此，如果 i在时间 t位于 v上（即，$v = v^i_c(t) = v^i_{Fcl}$），试图移动到 $v^i_{下一个} \in V^k_{树}$，则 i 向v发送请求消息。随后，仅当当前没有其他智能体位于 $G^k_{树} \cup G_{边缘区域}$ 中时，v 才向 i 返回接受消息；否则，如果可能，v会发送带有绕行信息的拒绝消息，并将其作为 SOM。此外，如果 v找不到 i 可以移动到的相邻节点，v 会发送带有等待信息的拒绝消息。

同时，当 $v^i_{下一个} = v$和 $v^i_c(t) \in V^k_{树} \cap V_{mar}$时，$i$向其协调者智能体 $v(= v^i_{Fcl} = v^i_{下一个})$发送请求，以预留 $v^i_{下一个}$。如果v能够为 i预留自身 $t+1$，则 v向 i发送接受消息；否则，它会发送一条包含等待指令的拒绝消息。i在 $v^i_c \in G^i_{树} \cap G_{mar}$上，如果 $v^i_{下一个} \notin V_{主}$，则不会发送请求消息。

当 $G^k_{树}$包含多个停车节点时，其根节点 $v \in V_{根}$有两种方法来管理进入其中的智能体数量。一种方法是在 MAPDFS 推进过程中将其设为单向；也就是说，在 MAPDFS 实例开始时，v 将移动方向限制为仅朝向主区域，此后则仅朝向树区域内部。然而，一旦智能体返回停车节点，便无法再回到主区域。另一种方法是通过端点管理进入树区域的智能体数量；v将进入 $V^k_{树} \setminus \{v\}$ 的智能体数量限制为一个；但是，当其到达停车节点时，v 会忽略该限制，因此另一个智能体可以进入此区域。同时，当位于停车节点的智能体 i 试图前往主区域时，会向其协调者 $v^i_{Fcl} = v$ 询问是否可以离开。随后，仅当 $G^k_{树}$中没有其他正在移动的智能体时，v 才接受该请求；否则，v 会向 i发送一条以等待作为 SOM 的拒绝消息。我们在下面的实验中采用了第一种方法。

4.5 主区域中的开放节点数量

最后，我们讨论主区域中智能体数量与节点数量之间的效率和差异。我们将不被任何智能体预留的节点称为开放节点，并且只有节点处于开放状态时，任何智能体才能移动到下一个节点。由于每个智能体都会预留一个不同的节点，如果$|V_{主区域}| = |I|$，那么所有智能体都无法移动。如果$|V_{主区域}| - |I| = 1$且所有智能体都位于 $G_{主区域}$，智能体将被限制在当前双连通分量内，无法移动到相邻的双连通分量。这是因为，当智能体 $i \text{ in } G_k$成功预留下一个开放节点后，当前节点将在下一时刻变为开放节点；因此，开放节点看起来就像是在向后移动。于是，对于智能体 i 进入另一个相邻的双连通分量 $G_{k'} \setminus G_k$，在 $G_{k'} \setminus G_k$中也应存在一个开放节点，并且该节点位于 i前方。然而，如果只有一个开放节点，这种情况就不可能发生。

表1：实验参数值。

参数描述及符号	值
相邻节点的正常所需时间，T_{mv}	3
装载/卸载时间，Tlu	3 或 6
移动延迟概率，ν	0 到 0.2
移动延迟时间，T_{nse}	1 或 2

当 $|V_{\text{主区域}}| - |I| = 2$时，只有当两个开放节点分别位于 $G_k \setminus G_{k'}$ 和 $G_{k'} \setminus G_k$中时，智能体 i 才可能进入相邻的双连通分量 $G_{k'} \setminus G_k$。然而，这种情况虽然可能发生，但大多只是偶然出现。因此，智能体能够到达目的地，但几乎是随机的；智能体可以完成所有任务，但需要花费很长时间。因此，很明显，主区域中的开放节点越多，智能体就越容易移动到目标相邻节点。由此，为了提高效率，建议主区域中超过一半的节点保持开放状态。

5 实验与讨论

5.1 实验设置

我们使用满足所需条件（SC1-SC3）且可能出现在应用场景中的两个环境中的 MAPDFS 实例，对所提出的方法进行了评估（图 1）。这两个环境包含数量较少的任务端点（蓝色圆点），分别对应任务的装载节点和卸载节点。图 1a 所示的第一个环境（环境1）包含 10 个任务端点，这些端点位于树状区域的末端，并满足其他方法（例如 HTE）的 WFI条件。图 1b 所示的第二个环境（环境2）同样包含 10 个任务端点，但这些端点位于主区域内，因此不满足 WFI条件。当智能体在任务端点进行装载或卸载时，可能会在装载或卸载完成前暂时阻挡其他智能体；不过，我们认为这在建筑工地中是常见做法，也是这一过程不可避免的一部分。

作为比较，我们实现了两种现有方法作为基线：*HTE [13]* 和 RHCR [9]。HTE 是一种去中心化方法，智能体参考同步共享内存块中的信息，依次规划其路径；该内存块包含任务集以及所有智能体的路径信息，包括到达和离开时间。RHCR 将一个 MAPD问题分解为一系列窗口化MAPF实例。随后，RHCR 中的智能体寻找并解决下一个 w时间步内发生的冲突，并每隔 h时间步重新规划路径。尽管 RHCR 不像 HTE 那样要求 WFI条件，但 RHCR 中的死锁规避并不完善。请注意，我们将 基于优先级的搜索 [11] 用作 RHCR 的 MAPF求解器，并参考原始实验 [9], 设置 $w = 60$ 和 $h = 15$；在原始实验中，$w = 20$ 和 $h = 5$；因此，我们将其乘以 $T_{mv} = 3$。

在第一个实验（实验1）中，我们将方法的性能与环境1中的两种基线方法进行了比较。环境1满足恒定移动速度下 HTE的WFI条件，因为基线方法无法处理延迟。在第二个实验（实验2）中，我们确认了所提出方法中的智能体是否能够完成环境2中的所有任务。环境2不满足WFI条件，并且存在较小的负向波动（即延迟）。因此，我们仅使用所提出的方法进行了实验2。我们在实验2中设置了 $Tlu = 6$，而在实验1中设置了 $Tlu = 3$。因此，智能体阻挡其他智能体的时间更长。我们使用了三项评估指标：（1）多智能体取放任务实例的完成率；（2）总完成时间（即完成所有任务所需的时间）；以及（3）规划（CPU）时间。

所有智能体均从各自的停车节点（图 1 中的绿色点）出发。任务的装卸节点从任务端点集合（蓝色点）中随机选取，并将初始任务同时分配给所有智能体。智能体完成当前任务后会被分配新任务，重复此过程，直至 T 中的所有任务均完成。如果由于所有任务均已分配而未给智能体分配任务，则该智能体返回其停车节点。

为模拟更真实的机器人运动，在实验2中，当智能体移动到下一个节点时，我们以概率 $\nu (0 \le \nu \le 1)$ 在移动速度T_{nse} 上加入噪声。因此，智能体通常需要$T_{mv} = 3$ 个时间步移动到相邻节点，但以概率 ν 需要$T_{mv} + T_{nse}$个时间步，其中 T_{nse}随机取 1 或 2。我们将智能体数量设置为 2 至 40 个，并将任务数量设置为 $|T| = 100$。其他所有参数值列于表 1 中。所有实验数据均为使用配备 64 GB 内存的 Apple M1 Max CPU 进行 50 次独立试验所得的平均值。

5.2 性能比较

5.2.1 完成率。 我们在 50 次运行中考察了完成率。表2列出了实验1中不同智能体数量下 MAPDFS 实例的完成率。如果运行时间超过时间步上限（10000 个时间步）、规划器无法找到无碰撞路径，或发生碰撞，我们将其视为失败。表2表明，我们的方法能够在实验1中无失败且无碰撞地完成所有实例。由于环境1满足 WFI条件，HTE 也能够完成实验1中的所有实例。然而，随着智能体数量的增加，RHCR 的完成率迅速下降 n，并最终变为零，此时 $n \ge 18$。这是因为智能体经常前往少数几个相同的任务端点作为其目的地，而任务端点附近的区域十分拥挤。因此，在这些情况下，RHCR 中的优先级规划似乎难以避免碰撞。需要注意的是，此处未列出相关数据，但我们还使用基线方法进行了实验2；尽管我们的方法完成了所有实例，这些基线方法却在所有实例中均失败。

5.2.2 完工时间。 图6绘制了实验1中不同智能体数量下的平均总完成时间（以时间步计）。请注意，计算平均值时排除了失败的实例。该图表明，即使智能体数量 n 增加到大约6个，无论采用何种方法，总完成时间都可以缩短。

然而，当 $n \ge 8$时，我们观察到这些方法之间存在性能差异；采用所提出的方法的智能体表现出最佳性能。随着智能体数量增加至 22 个，它们的总完成时间逐渐缩短，但当 $n \ge 24$时，性能略有下降。这种轻微的下降是由于智能体数量增加导致任务端点附近区域过度拥挤所造成的。同时，当 $2 \le n \le 6$时，

表2：MAPDFS实例的完成率和规划时间。

		智能体数量																
	Alg.	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	35	40
完成率	所提出的方法 $v = 0$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	HTE	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	RHCR	1.0	0.92	0.86	0.78	0.52	0.28	0.18	0.02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
规划时间	所提出的方法 $v = 0$	0.09	0.15	0.22	0.29	0.36	0.46	0.54	0.63	0.71	0.84	0.96	1.09	1.20	1.36	1.56	2.09	2.78
	HTE	11.2	11.6	12.2	13.6	15.0	15.1	14.9	15.0	15.1	15.1	15.1	15.5	15.4	15.1	15.3	15.5	15.5
	RHCR	90.8	128.1	167.5	209.1	270.4	358.7	421.9	640.4	-	-	-	-	-	-	-	-	-

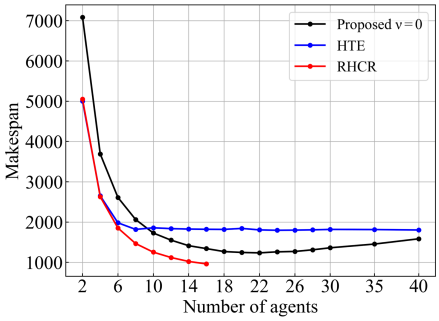


图6：实验1中的完工时间。

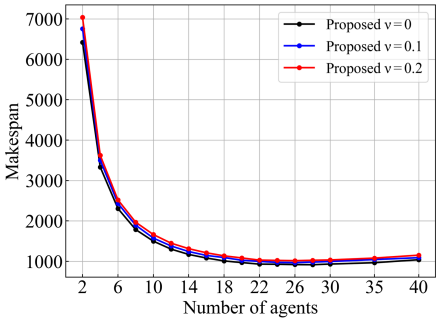


图7：实验2中的完工时间。

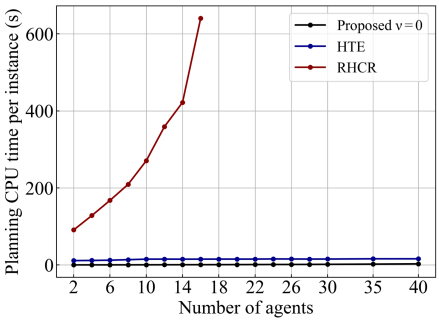


图8：实验1中每个实例的规划时间（秒）。

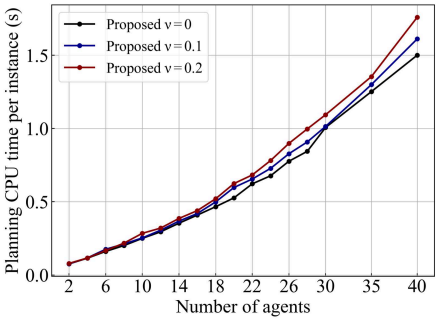


图9：实验2中每个实例的规划时间（秒）。

基线方法的表现优于所提出的方法。即使任务端点的数量并不显著，基线方法也能使智能体并行移动通过环境。然而，采用所提出的方法的智能体有时会因遵循边的方向而被迫进行更长的绕行。

相反，采用 HTE 的智能体的性能几乎保持不变，当 $n \geq 8$。尽管环境1满足 WFI 条件，但其任务端点数量较少，少于智能体数量。因此，HTE 只能为智能体分配有限数量的任务，因为智能体无法选择或被分配装载节点或卸载节点已被预留为其他正在执行任务的任务端点的任务。

尽管 RHCR 在实验1中一直明显优于其他方法，直到 $n \leq 16$，但当数量超过 16 后，我们无法计算总完成时间，因为由于之前讨论过的拥塞，RHCR 无法完成多智能体取放任务问题的任何实例。即使 $n = 12$，表2显示采用 RHCR 的智能体的完成率为 0.28；由于完成率较低，在我们的目标应用中使用 RHCR 并不现实。

我们在图7中绘制了实验2的平均总完成时间，以研究移动速度波动对MAPDFS中总完成时间的影响。该图显示，随着移动延迟概率的增加，性能逐渐下降 v。然而，两者之间的差异并不显著。因此，结果表明，所提出的方法对移动速度的波动具有鲁棒性。我们还通过设置 $Tlu = 3$进行了实验；但是，差异非常小。我们认为，这种影响是由主

区域的朝向造成的；尽管某个智能体必须等待其他智能体进行装卸，但它们会根据朝向沿同一方向开始移动。因此，智能体无需担心正面碰撞，并且可以在装卸位置旁边等待。

5.2.3 规划（*CPU*）时间。图8展示了实验1中不同智能体数量下，每个实例所有智能体的平均规划时间。我们还在表2中列出了规划总时间的详细信息。显然，无论智能体数量如何，所提出的方法的规划时间都远小于其他方法。这是因为，与HTE不同，采用所提出的方法的智能体无需时间信息，也无需考虑其他智能体的路径，即可生成路径。

从图中可以清楚地看出，当智能体数量为 $n \geq 10$ 时，HTE的规划时间几乎相同，因为并行移动的智能体数量受到限制，只有处于活动状态的智能体会生成规划。相反，在所提出的方法中，随着 n的增加，规划时间略有增加，因为所有智能体都并行移动，并且都需要时间进行规划。然而，即使 $n = 40$，由于采用了简单的分布式规划，所提出的方法的规划总时间也仅为2.78秒。与此同时，RHCR需要相当长的规划时间（图8），因为所有智能体都必须在 $h = 15$ 个时间步内至少重新规划一次，通过交替进行规划与执行来实现。

图9展示了实验2中移动延迟概率为 $\nu = 0$，0。1，0。2 时每个实例的平均规划时间。与实验1不同，由于其他智能体造成的延迟以及其他智能体的装卸操作，智能体往往不得不在主区域的同一节点停留更长时间。因此，智能体可能会更频繁地收到拒绝消息。然而，从图中可以看出，移动速度波动对规划时间的影响非常小。

5.3 讨论

在满足所需条件（例如对速度波动具有鲁棒性）的环境中，所提出的方法通过采用异步执行的分布式规划，在没有发生碰撞或进入死锁状态的情况下完成了所有任务。这是因为无论速度变化造成的延迟如何，节点智能体都会阻止载体智能体移动到已被其他智能体预留和/或占据的下一个节点，而直接边则防止智能体沿相反方向穿越同一条边。

此外，当智能体数量超过 10 个时，在任务端点数量较少的环境中，我们的方法优于基线方法（HTE 和 RHCR）。我们的实验结果表明，所提出的方法能够提高任务执行并发度，并减轻拥挤区域导致的性能下降。由于任务端点数量少于智能体数量，HTE 无法提高并发任务执行数量。此外，RHCR 要求所有智能体在同步规划和执行的情况下反复重新规划，这增加了计算成本；由于拥塞导致许多活动状态和死锁情况，RHCR 也无法在合理时间内完成所有任务。

在所提出的方法中，智能体生成通往目的地的路径后，通过局部通信移动到下一个节点以检查可用性，并在必要时修改路径。因此，该方法被认为类似于窗口大小受限的局部修复算法，例如传统的局部修复 A^* [30], 及其扩展算法，以及 RHCR。这类算法的缺点是，当许多智能体聚集在少量节点时，可能会导致碰撞概率较高、出现许多活锁状态，并且由于拥塞而需要进行大量代价高昂的修复/重新规划。然而，由于所提出的方法在图中引入了方向性，因此可以防止例如某个智能体被来自左右两侧的其他智能体夹在中间。此外，由于智能体沿着其能够移动的方向进行导航，即使智能体的目的地集中且拥挤，也可以将智能体暂时移动到周围区域，以保持其移动能力。

最后，我们还需要进一步讨论主区域中的开放节点数量和（搬运）智能体数量。如第 4.5 节所述，当 $|V_{\text{主区域}}| - |I| = 2$ 时，只有两个智能体可以同时开始移动，但移动几乎是随机的。对于假设同步移动 [12, 13, 16, 25], 的去中心化方法而言，这一限制有很大不同；在这些方法中，所有智能体都会同步移动，因此即使 $|V_{\text{主区域}}| = |I|$，智能体也可以移动。然而，我们假设采用异步移动，因此无法进行这种移动。此外，由于分布式环境中的异步性，移动方式会受到活动时机的影响，例如消息到达的时间和顺序。因此，性能会受到随机性的部分影响。不过，在使用我们的方法进行的大量实验中，智能体都完成了所有任务。

6 结论

我们提出了一种采用异步执行的分布式规划方法，为现实环境提供了高效且稳健的解决方案。我们的方法简单易行，同时适用于任务端点数量少于智能体数量的环境，以及包含智能体波动移动速度的环境。根据实验结果，所提出的方法在MAPD问题上优于基线方法；即使在由于速度可变且端点位置灵活而不适用于基线方法的环境中，所提出的方法也表现良好。我们的方法在此类环境中高效地完成了所有任务，且没有发生碰撞和死锁。

未来，我们计划扩展该方法；例如，放宽环境图的条件，提出适当的图定向方法以提高有效性和效率，并处理复杂任务（例如，一项任务可以由多个智能体执行）。

参考文献

[1] Mikhail J. Atallah。1984。《无向图的并行强定向》。*Inform. Process. Lett.* 18，1（1984），37–39。https://doi.org/ 10 1016 0020 0190 84 90072 3。/ -（）- Max Barer、Guni Sharon、[2] Roni Stern 和 Ariel Felner。2014。《多智能体路径规划问题的基于冲突的搜索算法的次优变体》。载于第七届组合搜索年度研讨会。Citeseer。[3] Eli Boyarski、Ariel Felner、Roni Stern、Guni Sharon、David Tolpin、Oded Betzalel 和 Eyal Shimony。2015。《ICBS：改进的多智能体路径规划基于冲突的搜索算法》。载于第*24*届国际人工智能会议论文集（阿根廷布宜诺斯艾利斯）（*IJCAI'15*）。AAAI Press，740–746。[4] Liron Cohen、Tansel Uras 和 Sven Koenig。2015。《可行性研究：利用高速路进行有界次优多智能体路径规划》。载于第八届组合搜索年度研讨会。2–8。

[5] Edsger W. Dijkstra。1976。《编程的规范》。Prentice-Hall。I–XVII，1–217页。[6] Ariel Felner、Jiaoyang Li、Eli Boyarski、Hang Ma、Liron Cohen、T. K. Satish Kumar 和 Sven Koenig。2018。《为基于冲突的搜索添加启发式方法以解决多智能体路径规划问题》。国际自动规划与调度会议论文集28，1（2018年6月），83–87。[7] Taoan Huang、Bistra Dilkina 和 Sven Koenig。2021。《在用于多智能体路径规划的有界次优基于冲突的搜索中学习节点选择策略》。载于《第20届国际自主智能体与多智能体系统会议论文集》。611–619。[8] J. S. Jennings、G. Whelan 和 W. F. Evans。1997。《移动机器人团队的协作式搜索与救援》。载于《1997年第八届先进机器人技术国际会议论文集》，ICAR'97。193–200。[9] Jiaoyang Li、Andrew Tinka、Scott Kiesel、Joseph W. Durham、T. K. Satish Kumar 和 Sven Koenig。2021。《大型仓库中的终身多智能体路径规划》。AAAI人工智能会议论文集35，13（2021年5月），11272–11281。[10] Minghua Liu、Hang Ma、Jiaoyang Li 和 Sven Koenig。2019。《多智能体取货与配送的任务与路径规划》。载于《第18届国际自主智能体与多智能体系统会议论文集》。国际自主智能体与多智能体系统基金会，1152–1160。[11] Hang Ma、Daniel Harabor、Peter J. Stuckey、Jiaoyang Li 和 Sven Koenig。2019。《使用一致性优先级排序进行多智能体路径规划》。AAAI人工智能会议论文集33，01（2019年7月），7643–7650。[12] Hang Ma、Wolfgang Hönig、TK Satish Kumar、Nora Ayanian 和 Sven Koenig。2019。《具有运动学约束的多智能体取货与配送终身路径规划》。载于AAAI人工智能会议论文集，第33卷。7651–7658。[13] Hang Ma、Jiaoyang Li、T.K. Satish Kumar 和 Sven Koenig。2017。《面向在线取货与配送任务的终身多智能体路径规划》。载于《第16届自主智能体与多智能体系统会议论文集》（巴西圣保罗）（AAMAS '17）。国际自主智能体与多智能体系统基金会，南卡罗来纳州里奇兰，837–845。[14] Yuki Miyashita、Tomoki Yamauchi 和 Toshiharu Sugawara。2022。《通过短视冲突解决实现多智能体系统的分布式异步规划与执行》。载于《2022年IEEE第46届计算机、软件与应用会议》（COMPSAC）。14–23。https://doi.org/10.1109/COMPSAC54236.2022.00012[15] Van Nguyen、Philipp Obermeier、Tran Cao Son、Torsten Schaub 和 William Yeoh。2017。《使用答案集编程进行广义目标分配与路径寻找》。载于《第26届国际人工智能联合会议论文集》（澳大利亚墨尔本）（IJCAI'17）。AAAI Press，1216–1223。[16] Keisuke Okumura、Manao Machida、Xavier Défago 和 Yasumasa Tamura。2019。《用于迭代多智能体路径规划的带回溯的优先级继承》。载于《第二十八届国际人工智能联合会议论文集》（IJCAI'

19 535 542 10 24963 2019 76）。–。

[17] https://doi.org/。/ijcai。/ Michal Pěchouček、David Šišlák、Dušan Pavlíček 和 Miroslav Uller。2006。《用于空中交通冲突消解的自主智能体》。载于《第五届国际自主智能体与多智能体系统联合会议论文集》（日本函馆）（AAMAS '06）。美国计算机协会，美国纽约州纽约，1498 1505 10 1145 1160633 1160925–。

[18] https://doi.org/。/。Herbert E. Robbins。1939。《图论定理及其在交通控制问题中的应用》。Amer. Math. Monthly 46（1939），281。[19] Oren Salzman 和 Roni Stern。2020。《多智能体路径规划与多智能体取货与配送问题中的研究挑战与机遇》。载于《第19届国际自主智能体与多智能体系统会议论文集》。1711–1715。[20] Guni Sharon、Roni Stern、Ariel Felner 和 Nathan R Sturtevant。2015。《用于最优多智能体路径规划的基于冲突的搜索》。Artificial Intelligence 219（2015），40–66。[21] Jiří Švancara、Marek Vlk、Roni Stern、Dor Atzmon 和 Roman Barták。2019。《在线多智能体路径寻找》。载于AAAI人工智能会议论文集，第33卷。7732–7739。[22] Robert Tarjan。1972。《深度优先搜索与线性图算法》。SIAM J. Comput。

1, 2 1972 146 160 10 1137 0201010（），–。

[23] https://doi.org/。/ Michal Čáp、Jiří Vokřínek 和 Alexander Kleiner。2015。《结构良好基础设施中的完整去中心化在线多机器人轨迹规划方法》。载于《第二十五届国际会议——国际自动规划与调度会议论文集》（以色列耶路撒冷）（ICAPS'15）。AAAI Press，324–332。[24] Qian Wan、Chonglin Gu、Sankui Sun、Mengxia Chen、Hejiao Huang 和 Xiaohua Jia。2018。《动态环境中的终身多智能体路径规划》。载于《2018年第15届控制、自动化、机器人与视觉国际会议》（ICARCV）。

875 882 10 1109 2018 8581181–。https://doi.org/。/ICARCV。。

[25] A. Wiktor、D. Scobee、S. Messenger 和 C. Clark。2014。《受限空间中的去中心化完整多机器人运动规划》。载于《2014年IEEE/RSJ智能机器人与系统国际会议》。1168–1175。

[26] Christopher Wilt 和 Adi Botea。2014 年。空间分布式多智能体路径规划。载于 第二十四届国际自动规划与调度会议论文集（美国新罕布什尔州朴次茅斯）（*ICAPS'14*）。AAAI Press，332–340。[27] Tomoki Yamauchi、Yuki Miyashita 和 Toshiharu Sugawara。2022 年。多智能体取送任务的基于待机的死锁规避方法。载于 第 *21* 届国际自主智能体与多智能体系统会议论文集（新西兰虚拟会议）（*AAMAS '22*）。国际自主智能体与多智能体系统基金会，南卡罗来纳州里奇兰，1427–1435。[28] Zhi Yan、Nicolas Jouandeau 和 Arab Ali Cherif。2013 年。多机器人协同综述与分析。《国际先进机器人系统杂志》10，12（2013），399，10，5772，57313，10，5772（），。https://doi.org/。/ arXiv:https://doi.org/。/

57313

[29] 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 第二十七届 *AAAI*人工智能会议论文集（华盛顿州贝尔维尤）（*AAAI'13*）。AAAI Press，1443–1449。[30] A. Zelinsky。1992 年。移动机器人探索算法。*IEEE*机器人学与自动化汇刊8，6（1992），707–717。https://doi.org/10.1109/70.182671

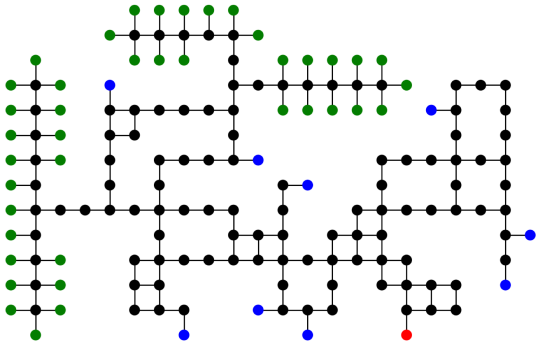


图10：示例环境（环境3）。

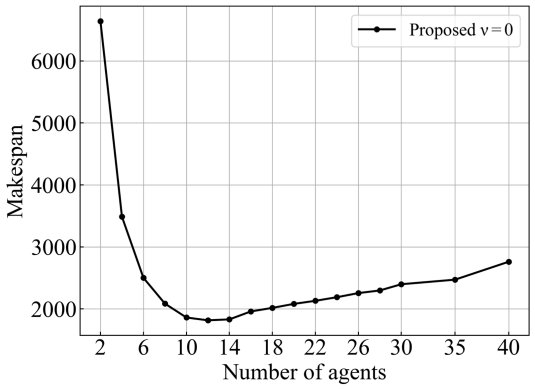


图11：实验3中的总完成时间。

附录
命题 1 的详细证明

尽管命题 1 在直觉上显而易见，我们仍将给出详细证明。

命题 1。 假设 $G_{main} = G_1 \cup \cdots \cup G_K$ 是连通图，且 G_k 是双连通分量。如果 $V_k \cap V_{k'} \neq \emptyset$，则 $V_k \cap V_{k'}$ 是单例。

证明。如果 $V_k \cap V_{k'}$ 包含两个节点 w 和 w'，我们可以证明 $V_k \cup V_{k'}$ 是双连通的。因此，我们证明：对于任意节点 $v \in V_k$ 和 $v' \in V_{k'}$，在 G_k 中，v 与 w 之间以及 v 与 w' 之间分别存在两条路径，且它们的公共节点仅为 v。由此，我们还可以在 $G_{k'}$ 中生成 v' 与 w 之间以及 v' 与 w' 之间的两条路径，且它们的公共节点仅为 v'。然后，通过经由 w 连接 v 与 v'，以及经由 w' 连接 v 与 v'，我们可以证明 $V_k \cup V_{k'}$ 是双连通的。首先，由于 G_k 是双连通的，我们可以在 G_k 中生成连接 v 与 w 的至少两条路径，使得除 v 和 w 外，这些路径没有公共节点。我们用节点序列表示这些路径：$p^0_{v\to w} = (v^0_0, v^0_1, \ldots, v^0_{h_1})$ 和 $p^1_{v\to w} = (v^1_0, \ldots, v^1_{h_2})$，其中 $v^0_0 = v^1_0 = v$、$v^0_{h_1} = v^1_{h_2} = w$、$v^0_l$ 和 $v^1_{l'} \in V_k$。类似地，我们可以生成 v 与 w' 之间的两条路径，其公共节点仅为末端节点。这些路径表示为 $p^0_{v\to w'} = (v'00, \ldots, v'^0_{h'_1})$ 和 $p^1_{v\to w'} = (v'10, \ldots, v'^1_{h'_2})$，其中 $v'^0_0 = v'^1_0 = v$、$v'^0_{h'_1} = v'^1_{h'_2} = w$，以及 v'^0_l 和 $v'^1_{l'} \in V_k$。如果 $\exists m \in \{0, 1\}$，使得 $p^m_{v\to w}$ 与 $p^1_{v\to w'}$ 和 $p^2_{v\to w'}$ 中的至多一个相交（端节点处除外），我们可以选择不与 $p^m_{v\to w}$ 相交的 $p^{m'}_{v\to w'}$。如果不存在这样的路径，则 $\forall m \in \{0, 1\}$，使得 $p^m_{v\to w}$ 同时与 $p^0_{v\to w'}$ 和 $p^1_{v\to w'}$ 相交。令 v^m_l 为 $(v^m_0, v^m_1, \ldots, v^m_{h_m})$ 中与 $p^0_{v\to w'}$ 或 $p^1_{v\to w'}$ 相交的最后一个节点。假设 e v^m_l 从 $p^m_{v\to w'}$ 中选取（e$m \in \{0, 1\}$），若考虑路径 $p_{v\to w} = (v' \text{e}^m_0, \ldots, v' \text{e}^m_{l'} = v^m_l, \ldots, v^m_h, v^m_{h_m})$ 和 $p^{1-\text{e}}_{v\to w'}$，则除 v 外它们没有交集。□

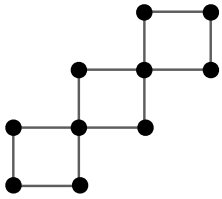


图12：示例环境（环境4）。

（补充）实验中智能体移动的视频

我们在本附录中添加了实验中智能体移动的视频。我们还进行了两项补充实验，以确认在经常出现过度拥挤的罕见环境中，使用所提出的方法的智能体是否能够在不发生碰撞的情况下完成所有任务。下面将简要介绍这些实验。

在第一次附加实验（实验3）中，环境的结构与实验1相同，但装载节点仅从红色节点中选择，卸载节点则从蓝色节点中选择，如图10所示。这假设材料存放在特定位置（红色节点），并被运送到各个节点；因此，红色节点周围很可能出现过度拥挤情况。

图11绘制了实验3中不同智能体数量下的平均总完成时间 n。需要注意的是，与其他实验一样，我们进行了五十次实验运行。实验结果表明，无论智能体数量（最大为40）如何，使用所提出的方法的智能体也能够完成实验3中的所有实例。使用所提出的方法的智能体能够随着智能体数量增加而减少总完成时间，直至 $n = 12$。此后，随着智能体数量增加 $n \geq 14$，性能逐渐下降，这可能是由于单个装载节点（红色节点）周围出现过度拥挤情况。此外，通过比较图6、图7和图11，我们可以看到，实验3中的总完成时间明显更长。这同样是因为只有一个用于装载的端点，因此所有智能体都可能聚集到该端点。我们想强调的是，使用所提出的方法的智能体仍能执行多智能体取放任务

实例而不会出现死锁情况，如下面的视频所示，这表明我们所提出的方法即使在这种过度拥挤情形下也具有鲁棒性。

在第二个补充实验（Exp.4）中，我们测试了主区域中的节点数量与智能体数量之间的相关性，这一点在主论文中有所提及。如图 12 所示，在由三个双连通分量组成的区域中，节点总数为 10 个，有 8 个智能体运行多智能体取放任务实例；因此，$|V_{\text{主区域}}| - |I| = 2$。我们还假设所有节点都是终点，因此智能体可以在任意节点装载和卸载。所有智能体最初随机放置在各个节点上，并开始执行取放任务。我们认为这种情况是病态的。然而，我们确认，在 Exp.4 中，智能体能够完成一个 MAPD 实例，尽管智能体几乎是随机地到达目的地，因此花费了很长时间。

以下是使用所提出的方法进行的四个实验中，用于展示智能体运动情况的示例视频列表。请注意，由于提交文件大小的限制，我们制作了视频的短片段。请注意，在这些视频中，边的方向用长三角形表示，而无向边用粗线表示。

实验1（实验1）。
智能体采用了所提出的方法，智能体数量为22，环境为环境1，示例视频可通过以下网址观看 https://youtu.be/l4xnnsy5TJs。我们可以确认，所有智能体都能够在该区域内移动且未发生碰撞，但由于多个智能体设定了相同的目的地，某些位置偶尔会出现轻微拥塞。

实验2（实验2）。
智能体采用所提出的方法，智能体数量为40，环境为Env.2。示例视频可通过网址 https://youtu.be/do3pa22yKps 查看。尽管Env. 2不满足WFI条件且存在较小的负向波动，但它们仍能在其中顺畅移动，并完成多智能体取放任务实例。

实验3（实验3）。
采用所提出的方法的40个智能体在环境Env. 3中完成了多智能体取放任务实例。示例视频可在 https://youtu.be/uXfgFJjgLIA 查看。正如上文所述，在如此拥挤的情况下，采用所提出的方法的智能体仍能完成所有实例。

实验4（Exp.4）。
采用所提出的方法的八个智能体在仅有十个节点的环境 Env. 3 中完成了多智能体取放任务实例。TSam-ple 视频可在 https://youtu.be/0ap6Vq9JbBw 查看。如上文所述，当 $|V_{\text{主要}}| - |I| = 2$ 时，使用所提出的方法的智能体能够完成整个实例。