arXiv:2304.14309v1 [cs.RO] 27 Apr 2023

双层多智能体取放货：大型仓库中的多机器人重排

Baiyu Li 和 Hang Ma

摘要——**我们提出一种新的问题建模：双层多智能体取货与配送（DD-MAPD），用于描述自动化仓库中的多机器人货架重排问题。DD-MAPD 扩展了多智能体取货与配送（MAPD）和多智能体路径规划（MAPF），允许智能体从货架下方移动，或抬起货架并将其运送至任意位置，从而改变仓库布局。我们证明，求解 DD-MAPD 是 NP 困难的。为应对 DD-MAPD，我们提出 MAPF-DECOMP 算法框架，将一个 DD-MAPD实例分解为一个用于协调货架轨迹的 MAPF 实例，以及一个后续的、带有任务依赖关系的 MAPD 实例，用于计算智能体的路径。我们还提出一种优化技术，以提升 MAPF-DECOMP 的性能，并展示如何使 MAPF-DECOMP 对良构DD-MAPD实例这一现实子类的 DD-MAPD 实例具有完备性。实验结果表明，MAPF-DECOMP 效率高、效能出色，仅需几分钟运行时间，即可为包含一千多个货架和数百个智能体的大规模实例计算出高质量解。**

索引词—**多机器人系统、多移动机器人或智能体的路径规划、任务规划**

I. 引言

在HE多机器人系统的实际应用中，通常需要多个智能体相互协作，以便在共享环境中重新排列物体。履约中心中的仓库机器人便是一个例子，例如 [1], 中所述的机器人；智能体会被用于搬运库存货架。在这些场景中，智能体必须避免彼此以及与其搬运物体发生碰撞。其他多智能体物体运输问题还包括自动化集装箱搬运和三维自动化仓库履约，在这些问题中，智能体可以操纵和移动物体。在这些情况下，必须防止智能体之间以及智能体与物体之间发生碰撞。

关于自动化仓库中的库存货架多机器人重排，已有大量研究，重点关注简化的多智能体路径规划（MAPF）问题 [2]。在 MAPF 中，每个智能体都必须从预先定义的起始位置快速移动到预先定义的目标位置，

稿件收到日期：2022年11月21日；修订日期：2023年3月2日；录用日期：2023年4月18日。副编辑和审稿人的意见经评估后，编辑M. Ani Hsieh推荐本文发表。本研究获得加拿大自然科学与工程研究委员会（NSERC）项目RGPIN2020-06540、华为技术加拿大公司和CFI JELF奖项的支持。视频：https://youtu.be/6CUdSmQYub8。作者就职于加拿大不列颠哥伦比亚省本拿比市西蒙菲莎大学计算机科学学院，邮编V5A 1B5 {baiyu_li, hangma}@sfu.ca。数字对象标识符（DOI）：见本页顶部。

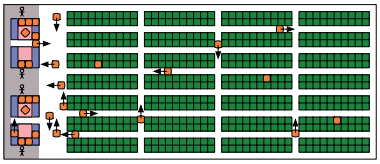

图1：履约中心的典型布局 [1]。

且不得与其他智能体发生碰撞。多智能体取货与配送（MAPD）[3] 将 MAPF 扩展到更贴近现实的场景，其中任务数量多于智能体数量。在 MAPD 中，每项任务都有预先定义的取货位置和交付位置。每个智能体都需要获分配一项任务，并先移动到取货位置，再移动到交付位置，以完成任务。任务分配和路径寻找会反复进行，直至所有任务均已完成。虽然与 MAPF 相比，MAPD 更适用于货架重排，但它假设货架的存储布局固定，且货架只能被取放至指定位置。MAPD 算法不会显式协调货架移动，因此无法解决诸如由单个智能体交换两个货架位置之类的问题。

因此，我们提出了一种新颖的问题建模——双层多智能体取货与配送（DD-MAPD），将现有的多智能体路径规划（MAPF）和 MAPD 技术拓展至实用的大规模真实世界仓库自主化场景。DD-MAPD 通过允许智能体在货架下方移动，或抬起、搬运并将货架放置在新位置，来解决多机器人货架重排问题。“双层”这一名称反映了避免两个层面发生碰撞的要求：高层是智能体搬运货架时货架之间的碰撞，低层则是智能体彼此之间的碰撞。DD-MAPD 的问题是指派 N 个智能体将 M 个货架从指定的取货位置移动到指定的配送位置，从而改变货架的整体布局。

从实践角度来看，DD-MAPD 的问题建模使机器人能够根据不断变化的产品需求，动态调整仓库布局。如图 1 所示，亚马逊配送中心的典型布局包含绿色单元格，这些单元格表示货架指定的存储位置；货架排列在由 4×7 个区块组成的布局中，每个区块包含 10×2 个存储位置。DD-MAPD 能够利用数量不同的智能体，高效地重新排列货架，从而创建相应的解决方案，例如将那些

请求频率较低的货架整理成密集配置，或通过移开其他货架，有选择地从密集排列的区块中“挖出”所需货架。这种方法可以显著减少土地使用量，最终降低自动化仓库的成本。

A. 相关工作

MAPF： MAPF 是 DD-MAPD 的一种特殊情况，其中 M货架由 $N = M$ 智能体搬运。MAPF 在一般图、平面图[6], 甚至二维四邻域网格 [7]上进行最优求解均为 NP 困难 [4], [5] 。近期的 MAPF 求解器包括基于归约的 [8], [9], [10], [11], 基于规则的 [12],和基于搜索的 [13], [14], [15], [16]方法。

MAPD： 现有的多智能体取放任务算法 [3], [17]将一个 MAPD 实例分解为一系列任务分配和 MAPF 实例。当系统发生变化时，例如智能体完成任务或添加新任务，它们会为智能体分配任务并规划路径。针对具有时间约束 [18] 或预定义依赖关系 [19]的任务的 MAPD 问题也已得到研究。然而，这些问题没有将货架建模为占据位置的可移动物体，而这对于货架重排是必要的。

目标排序与可配置环境：近期研究 [20], [21], [22] 探索了多智能体路径规划变体，其中每个智能体必须访问多个目标，并计算访问这些目标的顺序。DD-MAPD也要求对任务进行排序，但任务位置必须动态计算。[23]研究了一种多智能体路径规划变体，其中仓库布局可由黑盒更改；而DD-MAPD要求计算智能体更改仓库布局的计划。

计划执行：多智能体路径规划和MAPD计划可以由真实机器人执行，并使用遵循先后约束的依赖图 [24]。即使存在未建模的运动学约束 [25]或延迟不确定性 [26]，也能保证计划执行过程中不会发生碰撞。与我们提出的用于解决DD-MAPD的框架不同，计算出的计划为每个智能体分配了各自的路径，且智能体不能执行其他路径。

B. 贡献

我们提出了一种新颖的问题建模 DD-MAPD，用于对仓库中的多机器人货架重排问题进行建模。从理论上讲，我们的工作将 MAPD 现有的不可近似性和 NP 困难性结果推广到 DD-MAPD，并建立了一组可解性的充分条件，即良构性。我们的主要贡献是一种新的算法框架 MAPF-DECOMP。该框架将一个包含 N 智能体和 M 货架的 DD-MAPD实例分解为一个 M智能体的 MAPF 实例，随后处理一个带有任务依赖关系的 N智能体 MAPD 实例——这是 MAPD 的一种扩展形式，迄今为止受到的关注十分有限。MAPF-DECOMP 首先求解 MAPF 实例，为货架规划无碰撞轨迹。随后，它将计算得到的货架轨迹转换为任务，并求解 N智能体 MAPD 实例，为智能体分配任务并规划路径，以完成所有任务。这种分解具有双重优势。首先，它通过减少规划中的状态空间和智能体数量，提高了处理 N 智能体—货架对问题的可扩展性。其次，它利用现成的 MAPF 求解器的进展来加速 DD-MAPD 的求解。我们还提出了一种优化技术，用于提升 MAPF-DECOMP 的效能，并提出了它的一个变体，该变体可以求解所有良构DD-MAPD实例，即 DD-MAPD 实例中的一个现实子类。实验结果表明，MAPF-DECOMP 能够在几分钟内为多达 1,843 个货架和 400 个智能体计算出高质量的解。

II. 问题定义

一个DD-MAPD实例由 N个智能体 a_1、.....、 a_N、M 个货架 s_1、.....、 s_M以及一个连通无向图$G = (V, E)$组成。图中的顶点 V 表示位置，边E 表示这些位置之间的连接，智能体可以沿这些连接移动。我们只考虑有意义的DD-MAPD实例，其中 $M \geq N$ ；否则，可以使用 $N = M$ 个智能体各自搬运一个独有的货架。

令 $\pi_i(t)$ 表示智能体 a_i 在时间步 t 的位置。每个智能体 a_i 在时间步 0 位于其起始位置，并在每个时间步移动到相邻位置，或在当前位置等待。每个货架 s_j 在时间步 0 位于其取货位置 p_j ，并被指定一个交付位置 d_j。如果货架 s_j 不需要重新定位，则 $p_j = d_j$。当未携带任何货架时，智能体可以在货架下方移动；当智能体与货架位于同一位置时，可以举起货架，此后携带该货架，并在到达另一位置时放置（放下）货架。当携带货架时，智能体处于活动状态；未携带货架时，则处于空闲状态。我们假设执行举起或放置动作所需的时间为 0，但我们的框架可以轻松推广，以适应具有非零时间成本的举起和放置动作。令 $\eta_j(t)$ 表示货架 s_j 在时间步 t 的位置。货架只有在由智能体携带时才能移动。智能体之间（在虚拟低层平台上）以及货架之间（在虚拟高层平台上）均不应发生碰撞。智能体 a_i 与 $a_{i'}$ 之间发生顶点碰撞，当且仅当 $\pi_i(t) = \pi_{i'}(t)$；发生边碰撞，当且仅当 $\pi_i(t) = \pi_{i'}(t+1)$ 且 $\pi_i(t+1) = \pi_{i'}(t)$。类似地，货架 s_j 与 $s_{j'}$ 之间发生顶点碰撞，当且仅当 $\eta_j(t) = \eta_{j'}(t)$；发生边碰撞，当且仅当 $\eta_j(t) = \eta_{j'}(t+1)$ 且 $\eta_j(t+1) = \eta_{j'}(t)$。

DD-MAPD问题旨在为智能体计算无碰撞路径，以便将所有货架从取货位置运送到配送位置。智能体 a_i 的完成时间是指该智能体到达其路径上的最后一个位置并停止移动的最早时间步。我们使用两个指标来衡量DD-MAPD算法的效能：最大完工时间，定义为所有智能体完成时间的最大值；以及流时间，定义为所有智能体完成时间之和。

我们使用图2所示的DD-MAPD实例作为贯穿全文的示例。图2（上）展示了一个最大完工时间为7、流时间为14的解（$= 7 + 7$）。

III. 复杂度与可解性

我们现在将 MAPD 的现有复杂度结果推广到 DD-MAPD，并确定使 DD-MAPD 实例可解的充分条件。

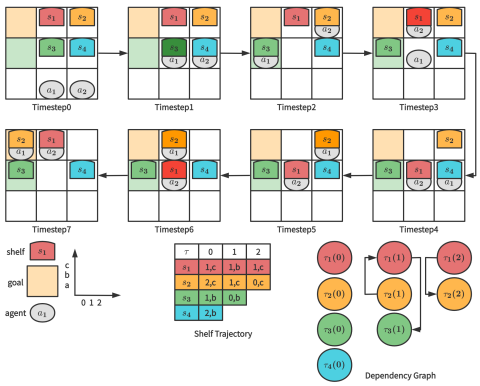

图 2：二维四邻域网格上包含两个智能体和四个货架的 DD-MAPD实例示例。货架 s_2 和 s_3 分别需要重新定位到橙色和绿色单元格。其他货架无需重新定位。上：每个时间步智能体和货架的位置。下：货架轨迹及由此产生的依赖图。

A. 复杂度

首先，我们通过重用从 NP 完全的 ≤3,=3-SAT 问题[28]到 MAPD 的归约 [27]，证明了 DD-MAPD 在最大完工时间最小化方面具有常数因子不可近似性，这与多智能体路径规划的相关结果 [29]类似。

定理 1。 对于任意 $\epsilon > 0$，寻找一个 $(4/3-\epsilon)$-近似解以最小化 *DD-MAPD* 的最大完工时间是 *NP* 困难的。

证明概要。 我们采用 [27] 中定理 4.5 证明所使用的相同归约方法：对于给定的含有 $\mathcal{N}$ 个变量和 $\mathcal{M}$ 个子句的 ≤3,=3-SAT 实例，构造一个包含 $M = N = 2\mathcal{N} + \mathcal{M}$ 个货架和智能体的 DD-MAPD实例。MAPD 的论证仍然适用于 DD-MAPD，因为每个构造的智能体只携带与其对应的构造货架。构造的 DD-MAPD实例存在最大完工时间为三的解，当且仅当 ≤3,=3-SAT 实例是可满足的；即使 ≤3,=3-SAT 实例不可满足，该实例始终存在最大完工时间为四的解。 □

上述证明中构造的 DD-MAPD实例具有这样的性质：每个智能体的完成时间至少为 3。因此，如果最大完工时间为 3，那么每个智能体恰好在 3 个时间步内完成，且流时间为 $3N\ (= 3(2\mathcal{N} + \mathcal{M}))$。此外，如果最大完工时间超过 3，那么流时间就会超过 $3M$，从而得到以下推论：

推论 2。 对于流时间最小化，寻找 *DD-MAPD* 的最优解是 *NP* 困难的。

B. 良构实例的基线完备算法

我们刻画了一类可求解的 DD-MAPD 实例子类，称为良构DD-MAPD实例，该子类推广了良构 MAPD 实例 [3]，，尽管在实践中我们通常需要求解非良构 DD-MAPD 实例。我们首先定义，针对货架的多智能体路径规划解（将货架视为能够自行移动的智能体）是*1*-鲁棒 [30] 的，当且仅当在任意时间步，如果某个位置当前已被占用，则不允许货架在下一时间步移动到并占据该位置，前提是该位置当前已被占用。我们注意到，在实践中，要求存在一个1-鲁棒多智能体路径规划解并不过于严格，因为其充分条件是多智能体路径寻找图中至少有两个顶点未被占用 [31]。**定义 1**（良构性与安全1-鲁棒性）。ADD-MAPD 实例是良构的，当且仅当所有智能体均从不同位置出发；移除任意 $N-1$ 智能体的起始位置后，图 G 仍保持连通；并且存在一个针对货架的安全1-鲁棒多智能体路径规划解，即该解不会使用任何智能体的起始位置。

随后，我们对一种对于所有良构DD-MAPD实例均完备的基线算法进行如下概述：首先，通过为所有货架计算一个安全的 1-鲁棒多智能体路径寻找解，为所有货架生成轨迹；然后，使用任意一个智能体，以锁步方式执行所有货架轨迹，即只有在为所有货架执行完多智能体路径规划解中的一个步骤后，才继续执行下一步，同时让其他所有智能体在其起始位置等待。显然，所有良构DD-MAPD实例都是可解的，而该基线算法可以解决其中的全部实例。我们的框架 MAPF-DECOMP 与该基线算法相似之处在于，它首先计算货架轨迹；但不同之处在于，它利用多个智能体来执行这些轨迹。

IV. MAPF-DECOMP

算法1展示了 MAPF-DECOMP 的伪代码。该算法首先调用多智能体路径寻找求解器，为所有货架计算从其取货位置到配送位置的无碰撞轨迹 τ [第 1] 行。这些轨迹表示每个货架的预定路径，但每个货架的实际路径由智能体执行。为将轨迹与货架或智能体的实际路径区分开来，我们将其称为 τ 轨迹。[1] 接下来，MAPF-DECOMP 将轨迹转换为依赖图，隐式地将每条轨迹划分为片段 [2]第行。每个片段表示轨迹的一部分，智能体可以通过携带货架并沿该片段行进来执行该部分。最后，MAPF-DECOMP 通过为智能体分配货架（片段）并为其规划路径，求解一个带有任务依赖关系的专用多智能体取送实例，使其在每个时间步完成所有可执行片段；在此过程中，当智能体从活动状态变为空闲状态或反之时，同时遵守片段之间的依赖关系 [第 3-11] 行。与现有的多智能体取放任务算法不同，MAPF-DECOMP 不会为活动智能体规划路径，而是让它们沿所携带货架的规划轨迹行进。因此，活动智能体的路径就是轨迹中尚未执行的部分。

为了解决由依赖图产生的专用 MAPD 实例，MAPF-DECOMP 在变量状态$_i$ 中维护每个智能体的当前状态状态$_i$.类型 为空闲 或活动 ，初始设置为空闲；状态$_i$.货架 是分配给智能体的货架 a_i ，初始设置为$null$ [第 4] 行。此外，每个

[1]本文采用了对“路径”和“轨迹”这两个术语的非标准用法，即反转了它们的传统定义。这样做的目的是与 MAPF 文献中对“路径”的用法保持一致；而在机器人学中，“轨迹”通常指智能体随时间执行的路径。

算法1： MAPF-DECOMP

$\tau \leftarrow$ 通过多智能体路径规划得到的货架轨迹;
$\mathcal{G} = (\mathcal{V}, \mathcal{E}) \leftarrow$ 构建依赖关系();
对智能体 a_i **执行**
  $States_i.type \leftarrow free$, $States_i.shelf \leftarrow null$;
遍历 s $_j$ **do**
  $Steps_j \leftarrow 0$;
当 仍有未完成的货架**时**
  更新($States$, $\mathcal{G}$);
  如果 $States$ 已发生变化**则**
    $FreePaths \leftarrow$分配与规划($States$);
  系统进入下一个时间步;
  移动($States$, τ, $FreePaths$);

算法2： 更新($States, \mathcal{G}$)

$\mathcal{A}_{sc} \leftarrow \emptyset$;
对每个智能体 a_i **执行**
  $s_j \leftarrow States_i.shelf$;
  $numDeps \leftarrow |\tau_j(s_j.step + 1)$的出边，位于 $\mathcal{G}|$;
  $isSoftDep \leftarrow numDeps = 1$？ SoftDep($States_i.shelf$, $\mathcal{G}$)： $false$;
  如果 $States_i.type = free$ **且**智能体 a_i 位于同一位置 $States_i.shelf$ **则**
    如果 $isSoftDep$**则**
      $\mathcal{A}_{sc} \leftarrow \mathcal{A}_{sc} \cup a_i$;
    否则，如果 $numDeps \geq 1$ **则**
      $States_i.shelf \leftarrow null$;
    else
      $States_i.type \leftarrow active$;
  否则如果 $States_i.type = active$ **则**
    如果 $States_i.shelf$ 已完成 **则**
      $States_i.shelf \leftarrow null$;
      $States_i.type \leftarrow free$;
    否则如果 $isSoftDep$ **则**
      $\mathcal{A}_{sc} \leftarrow \mathcal{A}_{sc} \cup a_i$;
    否则如果 $numDeps \geq 1$ **则**
      $States_i.shelf \leftarrow null$;
      $States_i.type \leftarrow free$;
$\mathcal{A}_{noMove} \leftarrow$FindNoMove($\mathcal{A}_{sc}$);
对每个智能体 $a_i \in \mathcal{A}_{sc}$ **执行**
如果 $a_i \in \mathcal{A}_{noMove}$ **则**
  $States_i.shelf \leftarrow null$;
  $States_i.type \leftarrow free$;
else
  $States_i.type \leftarrow active$;

货架 s_j 被分配了一个步数 s_j.步数 $= k$，该步数根据其轨迹表示其当前位置$\tau_j(k)$。步数 s_j.步数 初始设置为 0，对应于取货位置$\tau_j(0) = p_j$ [行 6]。我们回顾一下：如果货架已到达其交付位置并停留在那里，则称该货架已完成。在存在未完成货架的每个时间步，MAPF-DECOMP 会根据依赖图和所有货架的步数更新所有智能体的状态 [行 8]。如果任一智能体改变了其状态，MAPF-DECOMP 会为智能体重新分配货架或安全位置，并调用多智能体路径寻找求解器，为它们计算无碰撞路径，使其到达分配的目的地 [行 10]。随后，算法进入下一个时间步，并将所有智能体及其携带的货架移动到各自的下一个位置 [行 11-12]。

A. 构建依赖关系图

给定调用多智能体路径寻找求解器的结果 τ 轨迹，MAPF-DECOMP 调用函数 BuildDep() 来构建依赖图。轨迹 τ 为其条目$\tau_j(k)$定义了一个全序，而依赖图将其放宽为偏序。得到的依赖图$\mathcal{G} = (\mathcal{V}, \mathcal{E})$是一个有向图，其顶点 $\mathcal{V}$ 为轨迹条目$\tau_j(k)$。对于所有 j、j'、k、k'，若 $j \neq j'$、$k > k'$，且位置$\tau_j(k) = \tau_{j'}(k')$，则存在一条边 $\langle \tau_j(k)\tau_{j'}(k'+1)\rangle$，表示货架 s_j 应处于步骤 k，即由$\tau_j(k)$指定的位置（该位置紧随$\tau_j(k)$之后，而k指定的位置与$\tau_{j'}(k')$相同），不得早于货架 $s_{j'}$ 处于步骤$k'+1$，即由$\tau_{j'}(k'+1)$指定的位置。也就是说，智能体是否可以执行从步骤 k 开始的片段，取决于货架 $s_{j'}$ 是否已经处于步骤 $k'+1$。根据构造方式，环只能存在于具有相同步骤 k的条目$\tau_j(k)$之间。上述依赖图的构建方式类似于先前工作 [25], [26] 中采用的方法，并保证货架的无碰撞轨迹在执行过程中也不会发生碰撞。不同之处在于，该方法不在同一轨迹的条目之间构建边。图2（底部）展示了运行示例的依赖图。

B. 更新智能体状态

在每个时间步，MAPF-DECOMP 调用函数 Update() 来更新智能体的状态。Update() 有三个作用：(1) 检查位于其分配的货架位置上的空闲智能体是否可以执行该货架的轨迹。(2) 验证活动智能体是否可以继续执行其分配货架的轨迹，或该货架的轨迹是否已完成。(3) 如果货架轨迹不是 1-鲁棒的，该函数会检查多个智能体的分配货架的下一条轨迹条目是否构成环或路径，以及这些货架是否可以同时执行。

我们将 $\mathcal{G}$ 上的（简单）路径 定义为一个顶点序列：序列中的每个顶点都有一条出边指向序列中的后继顶点，且顶点和边均不重复。我们将 $\mathcal{G}$ 上的（简单）环 定义为（简单）路径，不同之处在于第一个顶点和最后一个顶点相同。

算法2展示了Update()的伪代码。 $\mathcal{A}_{sc}$ 是这样一组可能的智能体：其分配的货架具有可能在下一步同时解除的依赖关系。 $\mathcal{A}_{sc}$ 初始设置为空 [第 1]行。对于每个智能体a_i，numDeps存储其分配货架的下一步所具有的依赖关系数量 s_j [第 4]行。如果numDeps $= 1$，则唯一的依赖关系为 $\langle \tau_j$（s_j.步 $+ 1$）$\tau_{j'}$（$k' + 1$）$\rangle$（其中τ_j（s_j.步 $+ 1$）$= \tau_{j'}$（k'）且步为 $+ 1 > k'$）。在这种情况下，函数SoftDep()当且仅当步骤$_{j'} = k'$（即，货架 $s_{j'}$ 位于步骤 k' 中，该步骤指定的位置与货架 s_j的下一步相同）时返回true，并将结果存储在布尔变量isSoftDep中。在这种情况下，我们称货架 s_j 受到软约束，或具有软依赖，因为货架s_j 和 $s_{j'}$ 均可同时向前移动一步，其中包括（τ_j（s_j.步 $+ 1$）和$\tau_{j'}$（$k' + 1$）处于环或路径中的情况。对于每个空闲智能体 a_i ，如果其与分配的货架 [位于同一位置 6],，则如果该货架具有软依赖，智能体 a_i 将被添加到 $\mathcal{A}_{sc}$ [第7-8]行。否则，如果该货架具有依赖关系（因而是硬依赖），则将其从智能体 a_i [中取消分配 10]第9-10]行。否则，该货架没有依赖关系，智能体 a_i 从

空闲变为活动，并开始执行货架的轨迹[第11-12]行。所有其他空闲智能体均不改变其状态（保持空闲）。对于每个活动智能体 a_i，如果其分配的货架已完成，则将该货架从该智能体中取消分配，并将其从活动变为空闲 [第13-16]行。否则，如果该货架具有软依赖，则将智能体 a_i 添加到 $\mathcal{A}_{sc}$ [第17-18]行。否则，如果该货架具有依赖关系（因而是硬依赖），则将其从智能体 a_i中取消分配，并将该智能体从活动变为空闲 [第19-21]行。所有其他活动智能体均不改变其状态，并继续执行所携带货架的轨迹。

函数 Update() 随后调用过程 FindNoMove()，以识别集合 $\mathcal{A}_{noMove}$ 中所有无法移动到其所分配货架的轨迹下一步所指定位置的智能体。回顾一下，每个这样的货架 s_j 都受到软约束，也就是说，它的下一条轨迹记录取决于另一个货架 $s_{j'}$ 的下一条轨迹记录（该货架不一定分配给集合 $\mathcal{A}_{sc}$ 中的智能体），因此它只有在货架 $s_{j'}$ 移动一步后才能移动一步。Find NoMove() 会考察由分配给集合 $\mathcal{A}_{sc}$ 中所有智能体的货架的下一条轨迹记录所诱导的子图 $\mathcal{G}_{sc}$ （属于 $\mathcal{G}$ ）。每条这样的轨迹记录都位于一个环或一条路径上 $\mathcal{G}_{sc}$，因为除以下情况外，每个顶点都有一条出边 $\mathcal{G}_{sc}$：它是某条路径上的最后一个顶点，且其唯一的出边 $\mathcal{G}$ 指向某个未分配给集合 $\mathcal{A}_{sc}$ 中任何智能体的货架的轨迹记录 $s_{j'}$ （因此，这条边不属于 $\mathcal{G}_{sc}$）。在这种情况下，如果货架 $s_{j'}$ 分配给一个空闲智能体（不在集合 $\mathcal{A}_{sc}$ 中），或根本没有分配给任何智能体，那么这条出边（依赖关系）就不会解除，因为没有任何智能体搬运货架 $s_{j'}$。这条路径上具有下一条轨迹记录的货架无法移动，因此，分配给它们的智能体也会被加入集合 $\mathcal{A}_{noMove}$。

C. 货架分配和路径规划

如果 Up- date() 将智能体的类型从空闲更改为活动，或从活动更改为空闲，并取消分配其分配的货架，MAPF-DECOMP 就会调用 AssignAndPlan()函数。随后，AssignAnd- Plan() 将可执行的货架，即在当前步不受约束的货架，分配给空闲智能体并规划其路径，除非这些货架已经由活动智能体携带。该函数分多个轮次运行，旨在让每个后续轮次中有更多货架变得可执行，因为上一轮规划的路径会解除约束。为此，AssignAndPlan()过程在每个轮次中执行以下步骤：(1) 构建未分配且可执行货架的候选集。(2) 如果不存在此类货架，函数就模拟 Move函数，让所有智能体（包括在前几轮中获得货架分配的空闲智能体和所有活动智能体）沿其路径移动。在模拟过程中，每个货架都与其分配的活动智能体一起移动，直到其到达下一条轨迹条目中的硬依赖。随后，AssignAndPlan() 识别出任何新变得可执行的未分配货架，并将其添加到候选集中。(3) 如果模拟没有使任何此类货架变得可执行，函数就识别出所有下一条轨迹条目构成环的未分配货架 $\mathcal{G}$，并将其添加到候选集中，因为这些货架可以被执行

算法3： 移动($States$, τ, $FreePaths$)

```
遍历每个智能体 a_i 执行
    如果 States_i.type = active 则
        s_j ← States_i.shelf;
        根据 τ_j 更新 a_i 和 s_j 的位置; s_j.step ← s_j.step + 1;
        从 G 中移除 τ_j(s_j.step) 的入边;
    else
        根据 FreePaths_i 更新 a_i 的位置;
```

。一旦候选集包含一个或多个货架，AssignAndPlan() 就会调用匈牙利算法 [32]，为货架寻找最小成本分配并将其分配给空闲智能体，同时规划其路径。将货架分配给智能体的成本，计算为该智能体与货架之间的最短路径距离，以及货架首次变得可执行的时间步两者中的最大值。随后，AssignAndPlan() 调用多智能体路径寻找求解器，为这些智能体计算从其当前位置到所分配货架当前位置的路径，同时避免与活动智能体的路径以及前几轮中已经规划的空闲智能体路径发生碰撞。当所有可执行货架都已完成分配后，对于仍未分配货架的每个空闲智能体，函数都会为其分配一个距离最近且不位于任何活动智能体路径上的唯一位置。然后，函数调用多智能体路径寻找求解器，为这些未分配货架的智能体计算路径，同时避免与所有其他智能体的路径发生碰撞。

D. 移动智能体和货架

系统进入下一个时间步时，多智能体路径规划-分解会调用 Move() 函数，让智能体和货架向前移动一步。算法3展示了 Move() 的伪代码。每个活动智能体 a_i 及其携带的货架都会根据货架 [的轨迹 τ_j 移动一步，第 4-5]，行会因此释放 τ_j 对应条目的依赖关系（从 $\mathcal{G}$ 中移除该条目的所有入边） [第 6] 行。每个空闲智能体都会根据空闲路径$_i$ [移动一步，第 8] 行。

E. 运行示例

图2（上）展示了运行示例中货架轨迹的执行过程。为便于说明，我们只指出最有启发性的细节。在时间步 0，a_1 和 a_2 被分配了 s_3 和 s_1，因为 s_2 在第1轮不可执行。在时间步 1，a_1 从空闲变为活动。在时间步 2，a_1 完成 s_3 并变为空闲。在第1轮中，只有 s_1 可执行，并被分配给 a_2。在第2轮中，s_2 被分配给 a_1。在时间步 4 和 5，尽管 a_2 与分配给它的货架 s_1 位于同一位置，Update() 仍将 a_2 设为空闲，因为 $\langle \tau_1(2)\ \tau_2(2) \rangle$ 是硬约束。在时间步 6，Update() 只将 a_2 添加到 $\mathcal{A}_{sc}$ 并将其设为活动，因为 $\langle \tau_1(2)\ \tau_2(2) \rangle$ 是软约束。

F. 优化：涉及未来（IVF）

现在，我们提出一种称为涉及未来（IVF）的优化技术，旨在提升多智能体路径规划分解（MAPF-DECOMP）中货架分配过程的效能。IVF 改进

分配与规划（AssignAndPlan()），在进行货架分配时，不仅考虑当前空闲的智能体，还考虑将在 K 个时间步后变为空闲的智能体。这样可以扩大参与货架分配的智能体范围，从而有可能得到更优的整体分配结果。为此，IVF 对更新（Update()）和移动（Move()）分别进行 K 次迭代模拟，并将在模拟过程中变为空闲的活动智能体纳入货架分配和路径规划过程。在计算将货架分配给智能体的成本时，记录每个此类智能体变为空闲时所经过的时间步数，并将其加到该智能体与任意货架之间的最短路径距离上。

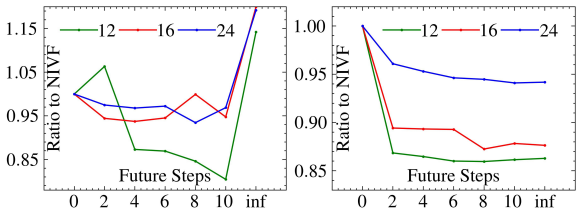

图3：在不同 K下，IVF相对于NIVF的智能体时间（左）和最大完工时间（右）比率。图例显示了网格大小。

V. 优先级规划（PP）的完备性

由于以下两个原因，MAPF-DECOMP无法保证解决所有良构DD-MAPD实例：(1) 如果货架轨迹不是1-鲁棒的，具有软依赖关系的多个货架可能形成一个环（详见第IV-C节）；当智能体总数少于该环中的货架数量时，该环无法被消解。(2) AssignAndPlan()函数为处于空闲状态的智能体规划前往不同于其他智能体起始位置的位置的路径，因此无法保证存在从这些位置出发的无碰撞路径。

因此，我们提出了MAPF-DECOMP的一种变体MAPF-DECOMP(PP)，该变体能够完备地解决所有良构DD-MAPD实例。该变体与原始版本的区别如下：(1) 它使用多智能体路径寻找求解器来计算安全的1-鲁棒货架轨迹。(2) 在每个分配轮次中，AssignAndPlan()仅通过选择分配成本最小的配对，将候选集中的一个货架分配给一个智能体，然后使用多标签A*搜索 [33] 计算该智能体的时间最优路径，同时避免与其他智能体的路径发生任何碰撞。该路径首先将智能体从其当前位置移动到货架的当前位置，然后沿货架轨迹移动，期间不等待，直至货架受到约束，最后将智能体移动到其起始位置。当所有可执行货架均已完成分配后，任何仍未被分配的空闲智能体都会保留其当前路径，该路径终止于其起始位置。因此，每个智能体始终保持其路径终止于起始位置这一不变量；这一做法受MAPD中的“保留虚拟路径”死锁规避技术启发 [17]。

定理3。 *MAPF-DECOMP(PP)*可解决所有良构*DD-MAPD*实例。

证明。 由于 1-鲁棒轨迹的依赖图是无环的，AssignAndPlan()函数至少会将一个尚未分配、已经可执行或（在模拟中）将变得可执行的货架添加到候选集，并将其分配给一个空闲智能体。多标签A*搜索保证能够找到一条供该智能体执行货架轨迹片段的无碰撞路径，因为这样的路径始终存在。例如，该智能体可以先沿用其旧路径，并在其起始位置等待，直到所有其他智能体都已完成各自的路径并无限期地停留在其起始位置。在不经过其他智能体起始位置的情况下，该智能体随后可以移动到其分配货架的当前位置，沿着该货架的轨迹移动，期间无需等待，也不会造成货架碰撞，直到该货架受到约束，最后返回其起始位置。因此，所有货架轨迹最终都会被执行，因为每次调用AssignAndPlan()时，至少都会分配并执行一个额外的轨迹片段。 □

VI. 实验

我们在一台3.1GHz英特尔酷睿 i5笔记本电脑上进行实验，该电脑配备16GB内存。我们实现了MAPF-DECOMP的三个变体：不采用IVF 优化的MAPF-DECOMP、采用IVF的MAPF-DECOMP，以及MAPF-DECOMP(PP)（分别标记为**`NIVF`**、**`IVF`**和**`PP`**）。它们均使用EECBS[15] 计算货架的轨迹和智能体的路径，并使用C++实现。我们对EECBS进行调整，使其能够为PP计算安全的1-鲁棒货架轨迹。我们将使用EECBS返回的1-鲁棒货架轨迹的IVF执行标记为**`IVF-R`**。在不同设置下，EECBS的次优因子 ω 范围为1.2至1.8，以平衡效能与效率。我们还实现了两种基线算法：一种使用单个智能体在锁定时间步中执行EECBS返回的1-鲁棒货架轨迹（标记为**`BASE`**）；另一种使用单个智能体执行Push and Swap [31]返回的1-鲁棒货架轨迹（标记为**`PAS`**）。与EECBS返回的轨迹不同，Push and Swap返回的连续1-鲁棒货架轨迹通常包含长度超过一步的片段，而PAS将这些轨迹解释为片段上的一个全序。

A. 使用 *IVF* 的 *MAPF-DECOMP*

我们通过随机采样由 $n \times n$ 个方格组成的不同规模的二维四邻域网格来构建 DD-MAPD 实例（标记为**规模** n）：随机选取由 2×2 个单元格组成的区块作为货架的取货位置，直至取货位置达到特定密度（占所有单元格的百分比，标记为**密度**）。我们从所有货架中随机抽取 $0.1 \cdot n^2$ 个货架，作为唯一需要重新定位的货架，并从非取货单元格中为其抽取配送位置。智能体的起始位置也通过随机抽样确定。每种设置下，我们评估 50 个随机实例，并报告所有已求解实例的平均值。

优化与未来时间步数量。图 3 展示了 IVF 在使用不同 K（未来时间步数量）取值时的结果，其中 $K = \inf$ 表示货架分配中包含所有活动智能体。对于较大的 K，最大完工时间往往更小。 $K = \inf$ 不一定会产生最小的最大完工时间，因为 EECBS 的最后一次调用会为少数智能体返回路径，这些路径比其他路径长得多，因为 EECBS 仅优化

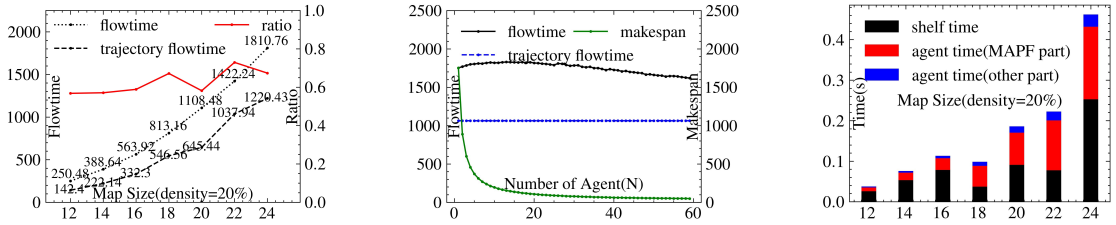

图 4：IVF 的结果（密度=20%）。左：4 个智能体在不同规模下的流动时间比与轨迹流动时间对比。中：24 种规模下不同智能体数量对应的效能。右：8 个智能体在不同规模下的时间分解。

规模	密度	M	N	最大完工时间	流时间	总计 time (s)	智能体 time (ms)	ω	成功
8	40%	25	4	31.10	109.32	1.81	2.82	1.2	
10		40	4	55.02	204.10	3.41	5.83	1.4	
			8	31.83	209.26	3.39	8.97		
12		57	4	89.18	339.53	0.30	11.81	1.6	
			8	50.27	347.39	0.31	17.52		
16		102	8	105.68	772.60	0.92	55.83	1.8	
16	20%	51	4	140.98	534.94	0.10	22.51	1.2	
			8	74.34	530.60	0.11	34.82		
24		115	4	452.69	1,775.31	0.38	134.82	1.2	
			8	236.65	1,786.88	0.46	209.49		
32		204	4	1,072.18	4,247.16	0.67	487.65	1.4	
			8	546.39	4,249.12	0.92	737.61		
40		320	8	1,072.35	8,444.20	2.62	1,119.00	1.6	
大规模实例，智能体时间以秒为单位报告							(s)		
48	20%	460	8	1,850.22	14,629.29	6.24	5.04	1.6	98%
			32	487.78	14 ,578.45	13.09	11.83		98%
64		819	8	4,451.22	35,425.22	32.02	25.68		98%
			32	1,148.41	35 ,476.73	106.00	99.26		98%
			100	397.68	33 ,648.55	88.95	82.57		94%
			400	153.31	26,969.08	182.46	176.72		96%
96		1,843	32	3,909.00	123,383.24	302.92	270.47	1.8	92%
			100	1 ,264.05	118 ,965.28	1 ,000.16	1 ,019.93		80%

大规模实例失败原因

规模	密度	M	N	实例 (a) 超时	(b) 小规模 N	(c) 不完整	
48	20%	460	8	1/50	0	1	0
			32	1/50	0	1	0
64		819	8	1/50	0	1	0
			32	1/50	0	1	0
			100	3/50	0	3	0
			400	2/50	0	0	2
96		1,843	32	4/50	1	3	0
			100	10/50	8	2	0
sum				23/400	9	8	2

表 I：IVF 在不同设置下的结果。

流时间。随着 K 增大，不包括货架轨迹计算的运行时间（标记为 **智能体时间**）会（1）首先下降，因为最大完工时间也会下降，并且对 EECBS 的调用次数减少；（2）随后可能上升，因为每次调用 EECBS 都涉及更多智能体。在接下来的所有实验中，我们对 IVF 使用 $K = 8$，以平衡运行时间与效能。由于涉及更多智能体能够获得更好的货架分配，IVF（使用 $K > 0$）始终比 NIVF 更有效。

网格大小、智能体数量和货架密度。表 I（上）显示，对于智能体数量较少的情况，IVF 能在数秒内解决所有实例；将智能体数量翻倍有时会增加流时间（即所有智能体完成时间的总和），但总能显著缩短最大完工时间。由于 AssignAndPlan() 调用 EECBS 的次数很多，最大完工时间较长时，智能体时间也会很长。表 I（中）显示，对于数百个智能体和一千多个货架，IVF 仍能取得较高的成功率（标记为 **成功**）。尽管在最大完工时间达到数千个时间步时，任务分配和路径规划函数必须执行多次，总运行时间仍只有几分钟。表 I（下）对实例失败的原因进行了分类：（a）EECBS 在计算货架轨迹时超时（1 分钟）（通常对应非常大的 M）；（b）智能体数量不足，无法同时移动软依赖循环中的所有货架（通常对应较大的 M和较小的 N）。对此，可以为货架计算一个 1-鲁棒 MAPF 解，或将 N 设为最大循环中的货架数量，以增加智能体数量；（c）AssignAndPlan() 中针对空闲智能体的路径规划不完整（通常对应非常大的 N）。如果给定的 DD-MAPD实例格式良好，可以采用避免死锁技术来解决这一问题。

效能。图 4（左）显示，货架轨迹的流时间（所有轨迹长度之和）是流时间的一个（显然的）下界，而 MAPF-DECOMP 并未对其进行优化；在不同实例规模下，该指标始终占流时间的很大部分（≥60%），这表明我们框架中的分解和轨迹执行均具有良好的效能。图 4（中）显示，对于相同的轨迹，随着智能体数量增加，当智能体数量较多时，流时间会下降，而最大完工时间在所有情况下都会下降。这是因为当智能体较多时，货架通常会被分配给距离较近的智能体。

运行时间分解。图 4（右）证实，用于计算货架轨迹的运行时间（标记为**货架时间**）占总运行时间的很大部分。这也表明，提高多智能体路径寻找求解器的效率将直接提升我们框架的效能，因为大部分运行时间都由多智能体路径寻找求解器占用。

B. 良构实例上的算法比较

我们采用类似的流程来构造随机 DD-MAPD 实例，但不在网格外围放置货架，而是从外围（不包括角落）的格子中采样智能体的起始位置，以保证良构性。表 II 显示，对于单个智能体，IVF-R 比两个基线方法更有效；增加智能体数量则能进一步提升效能。例如，在规模为 96 的实例中，100 个智能体对应的最大完工时间仅为单个智能体时的 1.1%。IVF-R 和 IVF 往往更有效，但效率不如 PP，因为 PP 会一次为一个智能体分配任务并规划路径。正如预期，PP 能解决所有良构实例。IVF-R 的成功率高于 IVF，因为它不会因先前实验中观察到的原因 (b) 而失败，但由于 AssignAndPlan() 不完整，它总共仍有三个实例未能解决。所有

size	M	N	算法。	最大完工时间	流时间	总计 time (s)	智能体 time (s)	ω	succ
48	460	1	BASE	190,243.52	-	1.20	-		100%
			PAS	14,707.44	-	2.31	-		100%
			IVF-R	13,439.26	-	2.55	1.35	100%	
		8	PP	1,757.33	13,910.77	2.11	0.93	100%	
			IVF-R	1,750.96	13,862.56	6.35	5.20	100%	
			IVF	1,745.44	13,812.10	6.86	5.80		96%
		32	PP	463.71	13,836.77	2.94	1.58		100%
			IVF-R	460.75	13,705.56	10.68	9.48		100%
			IVF	461.35	13,740.19	10.05	9.13		96%
64	819	1	BASE	605,647.28	-	4.34	-		100%
			PAS	34,642.68	-	9.17	-		100%
			IVF-R	32,560.80	-	9.06	4.72		100%
		8	PP	4,258.04	33,877.76	7.66	3.18	100%	
			IVF-R	4,254.92	33,839.10	25.64	21.01	100%	
			IVF	4,256.45	33,865.43	24.43	20.98	98%	
		32	PP	1,105.00	34,090.16	10.69	5.38		100%
			IVF-R	1,099.69	33,943.73	41.29	36.64		100%
			IVF	1,096.71	33,835.06	40.50	37.01		98%
		100	PP	384.81	32,695.28	17.78	11.00		100%
			IVF-R	381.58	32,419.40	63.19	58.75		98%
			IVF	384.00	32,530.45	66.11	62.59		98%
96	1,843	1	BASE	3,027,935.92	-	37.13	-		100%
			PAS	112,282.98	-	71.02	-	100%	
			IVF-R	111,258.48	-	72.10	34.97	100%	
		32	PP	3,817.24	120,368.60	79.49	33.06	100%	
			IVF-R	3,817.67	120,410.38	301.82	264.46		98%
			IVF	3,807.28	120,134.72	294.06	271.03		92%
		100	PP	1,235.37	115,719.63	123.86	64.49		100%
			IVF-R	1,228.53	115,274.80	623.54	585.69		98%
			IVF	1,229.35	115,037.24	708.07	684.56		92%

表 II：大规模良构实例上的结果。

算法在每次调用多智能体路径规划求解器时均设定一分钟的运行时间上限，且都不会超时。总体而言，如果能够保证良构性，PP 在实践中似乎是最优的算法选择，因为它在效率与效能之间取得了良好的平衡。

C. 仓库重排演示

我们在规模为 27×27 的二维四邻域网格上构建了 50 个良构实例，其中 32 个智能体沿周边起始，其布局基于一个由 8×4 个 5×2 货架组成的履约中心。320 个货架的配送位置采用对角对称配置，并进行随机打乱。使用带有 $\omega = 1.8$ 的 EECBS，IVF-R 成功解决了所有实例，平均总时间为 30.68 秒，平均智能体时间为 4.85 秒。展示其中一个实例执行过程的演示视频可访问：https://youtu.be/WFPl3wKDXXY。

VII. 结论与未来工作

我们提出了一种用于求解 DD-MAPD 的新算法框架，使多智能体路径规划和 MAPD 能够应用于大规模仓库中的多机器人货架重排问题。在本文中，我们专注于提高框架的效率，同时不过多牺牲其效能。

我们提出了框架的两项未来扩展方向：(1) 我们计划让其用于货架轨迹的多智能体路径规划求解过程考虑后续的分解与规划，从而提高框架的效能。(2) 我们计划为活动智能体规划路径，而不是让它们沿着货架轨迹移动。

参考文献

[1] P. R. Wurman、R. D'Andrea 和 M. Mountz，《在仓库中协调数百辆协作型自主车辆》，《*AI Magazine*》，第 29 卷第 1 期，第 9–20 页，2008 年。

[2] R. Stern、N. Sturtevant、A. Felner、S. Koenig、H. Ma、T. Walker、J. Li、D. Atzmon、L. Cohen、T. K. S. Kumar、E. Boyarski 和 R. Bartak，《多智能体路径寻找：定义、变体与基准测试》，载于 *SoCS*，2019 年，第 151–159 页。

[3] H. Ma、J. Li、T. K. S. Kumar 和 S. Koenig，《面向在线取放任务的终身多智能体路径寻找》，载于 AAMAS，2017 年，第 837–845 页。[4] J. Yu 和 S. LaValle，《图上最优多机器人路径规划的结构与难解性》，载于 AAAI，2013 年，第 1444–1449 页。[5] P. Surynek，《多机器人路径规划的一个优化变体是不可解的》，载于 AAAI，2010 年，第 1261–1263 页。[6] J. Yu，《平面图上最优多机器人路径规划的不可解性》，IEEE RA-L，第 1 卷第 1 期，第 33–40 页，2016 年。[7] J. Banfi、N. Basilico 和 F. Amigoni，《带孔二维网格图上的时间最优多机器人路径规划的不可解性》，IEEE RA-L，第 2 卷第 4 期，第 1941–1947 页，2017 年。[8] E. Lam、P. Le Bodic、D. Harabor 和 P. Stuckey，《多智能体路径寻找的分支定界与割平面价格法》，载于 IJCAI，2019 年，第 1289–1296 页。[9] P. Surynek，《通过理论可满足性统一基于搜索和基于编译的多智能体路径寻找方法》，载于 IJCAI，2019 年，第 1177–1183 页。[10] G. Gange、D. Harabor 和 P. Stuckey，《惰性CBS：使用惰性子句生成的隐式基于冲突的搜索》，载于 ICAPS，2019 年，第 155–162 页。[11] R. N. G´omez、C. Hern´andez 和 J. A. Baier，《使用答案集编程求解代价总和多智能体路径寻找问题》，载于 AAAI，2020 年，第 9867–9874 页。[12] K. Okumura、M. Machida、X. D´efago 和 Y. Tamura，《用于迭代式多智能体路径寻找的带回溯优先级继承》，载于 IJCAI，2019 年，第 535–542 页。[13] G. Sharon、R. Stern、A. Felner 和 N. Sturtevant，《用于最优多智能体路径寻找的基于冲突的搜索》，Artificial Intelligence，第 219 卷，第 40–66 页，2015 年。[14] G. Wagner 和 H. Choset，《用于多机器人路径规划的子维度扩展》，Artificial Intelligence，第 219 卷，第 1–24 页，2015 年。[15] J. Li、W. Ruml 和 S. Koenig，《EECBS：面向多智能体路径寻找的有界次优搜索》，载于 AAAI，2021 年，第 12 353–12 362 页。[16] J. Li、D. Harabor、P. Stuckey、H. Ma、G. Gange 和 S. Koenig，《用于多智能体路径寻找搜索的成对对称性推理》，Artifical Intelligence，第 301 卷，第 103574 页，2021 年。[17] M. Liu、H. Ma、J. Li 和 S. Koenig，《多智能体取放任务的任务与路径规划》，载于 AAMAS，2019 年，第 2253–2255 页。[18] V. Nguyen、P. Obermeier、T. C. Son、T. Schaub 和 W. Yeoh，《使用答案集编程进行广义目标分配与路径寻找》，载于 IJCAI，2017 年，第 1216–1223 页。[19] K. Brown、O. Peltzer、M. A. Sehr、M. Schwager 和 M. J. Kochenderfer，《多智能体机器人装配规划的最优序列任务分配与路径寻找》，载于 ICRA，2020 年，第 441–447 页。[20] P. Surynek，《通过解耦式与集成式目标顶点排序进行多目标多智能体路径寻找》，载于 AAAI，2021 年，第 12 409–12 417 页。[21] H. Zhang、J. Chen、J. Li、B. Williams 和 S. Koenig，《面向前置约束目标序列的多智能体路径寻找》，载于 AAMAS，2022 年，第 1464–1472 页。[22] Z. Ren、S. Rathinam 和 H. Choset，《用于多智能体组合路径寻找的基于冲突的 Steiner 搜索》，载于 RSS，2022 年。[23] M. Bellusci、N. Basilico 和 F. Amigoni，《可配置环境中的多智能体路径寻找》，载于 AAMAS，2020 年，第 159–167 页。[24] W. H¨onig、S. Kiesel、A. Tinka、J. W. Durham 和 N. Ayanian，《仓库中 MAPF 调度方案的持久且鲁棒的执行》，IEEE RA-L，第 4 卷第 2 期，第 1125–1131 页，2019 年。[25] W. H¨onig、T. K. S. Kumar、L. Cohen、H. Ma、H. Xu、N. Ayanian 和 S. Koenig，《带运动学约束的多智能体路径寻找》，载于 ICAPS，2016 年，第 477–485 页。[26] H. Ma、T. K. S. Kumar 和 S. Koenig，《带延迟概率的多智能体路径寻找》，载于 AAAI，2017 年，第 3605–3612 页。[27] H. Ma，《智能体团队导航任务的目标分配与路径规划》，博士学位论文，USC，2020 年。[28] C. Tovey，《一个简化的 NP 完全可满足性问题》，Discrete Applied Mathematics，第 8 卷，第 85–90 页，1984 年。[29] H. Ma、C. Tovey、G. Sharon、T. K. S. Kumar 和 S. Koenig，《带载荷转移的多智能体路径寻找与包裹交换机器人路由问题》，载于 AAAI，2016 年，第 3166–3173 页。[30] D. Atzmon、R. Stern、A. Felner、G. Wagner、R. Bart´ak 和 N.-F. Zhou，《鲁棒多智能体路径寻找与执行》，Journal of Artificial Intelligence Research，第 67 卷，第 549–579 页，2020 年。[31] R. Luna 和 K. E. Bekris，《Push and Swap：具有完备性保证的快速协作路径寻找》，载于 IJCAI，2011 年，第 294–300 页。[32] H. W. Kuhn 和 B. Yaw，《分配问题的匈牙利方法》，Naval Research Logistics Quarterly，第 83–97 页，1955 年。[33] F. Grenouilleau、W.-J. van Hoeve 和 J. N. Hooker，《用于多智能体路径寻找的多标签A*算法》，载于 ICAPS，2019 年，第 181–185 页。