通过机器人再分配在复杂环境中进行大规模多机器人任务分配

Seabin Lee[a], Joonyeol Sim[a], Changjoo Nam[a,*]

[a]*Dept. of Electronic Engineering, Sogang University, Baekbeom-ro, Mapo-gu, Seoul, 04107, South Korea*

Abstract

我们研究多机器人任务分配（MRTA）问题，目标是在具有密集障碍物和狭窄通道的复杂环境中优化多个机器人到多个任务的分配。在这样的环境中，传统的方法由于机器人之间的冲突会产生额外的成本（例如，避免碰撞、等待），因而在优化总成本时通常是无效的。此外，不考虑实际机器人路径的分配可能导致死锁，这将显著降低机器人的整体表现。我们提出了一种可扩展的 MRTA 方法，该方法考虑了机器人的路径，以避免碰撞和死锁，从而快速完成所有任务（即，最小化完工时间）。为了将机器人路径纳入任务分配，提出的方法使用广义 Voronoi 图构建了一张路线图。该方法将路线图划分为多个部分，以了解如何重新分配机器人，以实现所有任务并降低机器人之间的冲突。在重新分配过程中，机器人根据一种先进先出的推-弹机制被转移到最终目的地。通过大量实验，我们证明了我们的方法可以在密集杂乱环境中处理数百个机器人的实例，而竞争方法则无法在时间限制内计算出解决方案。

Keywords: Multi-robot coordination, multi-robot task allocation, logistics automation

1. 介绍

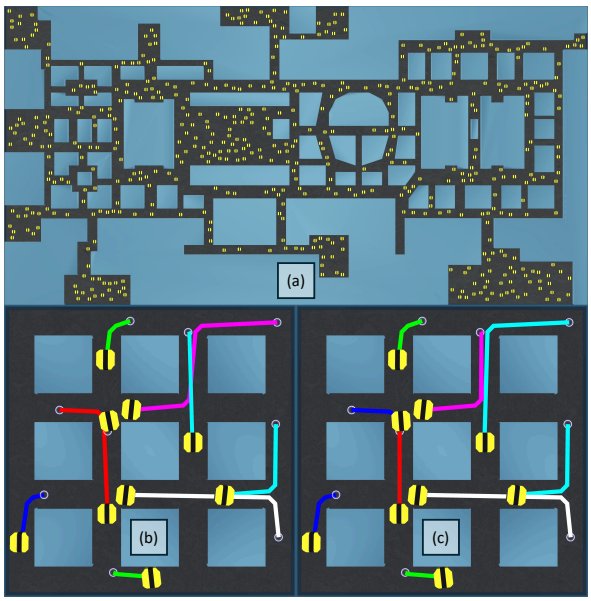


Figure 1: 考虑任务分配中机器人之间冲突影响具有挑战性的环境示例。(a) 在一个有狭窄通道的购物中心里的 500 个机器人，这是我们测试实例之一，频繁发生冲突。(b) 一个未考虑机器人之间冲突的传统方法（匈牙利法）找到的分配。彩色线条表示从机器人到其分配任务的预期路径，未考虑机器人之间的碰撞。带有红线的机器人的路径将被其他路径上的机器人阻挡。(c) 所提方法的结果。机器人不会在走廊中逆向行驶，这有助于防止死锁。

多机器人任务分配（MRTA）考虑优化执行一组任务的机器人团队的整体性能。如图 1 a 所示，在具有狭窄通道和密集障碍物的复杂环境中，机器人分配任务时应考虑机器人在环境中导航时的冲突。否则，由于冲突，机器人可能会遇到长时间的延迟，或者面临一种死锁情况，导致任务未完成，因为一些机器人无法继续前进。

前述问题可以根据优化准则以多种方式解决。在多项式时间内可以找到使成本总和最小化的分配 [1]，然而，最小化执行所有机器人任务的最大成本（即极小极大分配问题）被证明是 $\mathcal{NP}$-hard [2]。虽然极小极大分配问题的标准形式是具有恒定的成本，但我们感兴趣的多机器人任务分配（MRTA）问题可能会受到分配确定后仍可能变化的成本的影响。由于机器人执行任务的路径是基于分配的任务来计算的，因此与路径直接相关的任务成本除非我们列举所有情况下的排列，否则无法准确估算。因此，使用在规划时估算的成本计算出的最优分配在运行时可能是次优的。

作为填补这一最优缺口的努力，冲突基础搜索 (CBS) [3] 已被用于通过将路径规划整合到 MRTA 中来解决 MRTA 问题 (CBS-TA) [4]，以便找到无冲突的分配。此外，用于多智能体取送 (MAPD) 问题的方法 ([5] , [6]) 也一起考虑了 MRTA 和路径规划问题。然而，它们的性能仅限于中等规模案例且机器人数量只有几十个。此外，它们需要对机器人的工作空间和动作空间进行离散化，因此在具有物理机器人的连续环境中可能会过度简化原始问题。我们提出了一种称为 M 双 R 机 T 任务 A 分配通过机器人 R 再分配 M 机制 (MRTA-RM) 的方法，该方法在分配机器人到任务的同时减少机器人之间的冲突。MRTA-RM 利用路线图来估计机器人执行指定任务的实际成本。该方法基于需求供给分析再分配机器人，将机器人送到缺乏机器人的环境部分。再分配过程消除了机器人相对方向的相遇，使得机器人可以减少死锁，缩短总工期（即所有任务的完成时间）。例如，常规方法（例如，匈牙利法 [7] ）在图 1 b 中找到的分配显示了来自相对方向的机器人相遇，这会导致延迟或死锁。另一方面，我方法的分配（如图 1 c 所示）没有机器人在相反方向穿过走廊，这有利于缩短总工

*Corresponding author

Email addresses: 123sebin@naver.com (Seabin Lee), antihero1290@gmail.com (Joonyeol Sim), cjnam@sogang.ac.kr (Changjoo Nam)

期。我们的贡献包括：

- 拥塞感知任务分配：所提出的方法通过构建一个道路图来考虑机器人执行任务的路径，该道路图被划分为几个组件。机器人在这些组件之间重新分配，以完成所有任务，并防止机器人出现反方向的相遇。因此，机器人在朝任务移动的过程中可以避免冲突和死锁，从而导致更短的总完成时间。
- 可扩展性：完整的任务分配过程运行快速，即使在密集杂乱的环境中也能处理数百个机器人和任务。
- 广泛的实验：我们在三种不同的紧凑环境和两种类型的机器人/任务分布中进行了大量实验。总共有六种其他方法被用于比较，以显示我们提出的方法的优势。

2. 相关工作

很少有方法能够考虑任务分配和路径规划，以基于机器人执行任务的实际路径来优化分配。

基于冲突的搜索与最优任务分配 (CBS-TA) [4] 通过找到一个分配和无冲突路径来执行分配的任务，从而产生一个最优解。换句话说，它解决了任务分配与路径搜索 (TAPF) 问题，该问题联合解决了多机器人任务分配 (MRTA) 和多智能体路径搜索 (MAPF)。CBS-TA 在一个搜索森林中操作，每个根节点代表一个不同的任务分配。在任务分配中遇到冲突时，CBS-TA 引入额外的约束来解决冲突，这导致在搜索树中创建新的子节点。这种高级别的搜索策略促进了任务分配和路径组合空间的高效探索。在 [4] 中还提出了增强型 CBS-TA (ECBS-TA)，通过引入一个焦点搜索来提高效率。虽然 CBS-TA 和 ECBS-TA 优于单独解决 MRTA 和 MAPF 的其他方法，但其在大规模或更复杂环境中的表现尚未得到广泛研究。任务冲突基搜索 (TCBS) [8] 是另一种基于 CBS 的算法，用于处理 TAPF 问题。该方法通过具有约束的树搜索来解决问题，同时考虑每个智能体的任务分配和无碰撞路径。然而，TCBS 仅在最多四个机器人中进行了测试，并找到了次优解。

活力驱动的遗传任务分配 [9] 使用遗传算法来优化任务分配，并应用特定策略来避免冲突。通过调整任务顺序解决调度冲突，而路径冲突则通过对较短路径的机器人采用等待策略来解决。同样，目标分配与规划 [10] 用于在图上解决词典顺序瓶颈分配问题 [11] 。任务分配完成后，每个机器人根据其到分配任务的路径被优先排序。随后，计算时间偏移并应用于每个机器人，以优化路径并避免机器人之间的冲突。

另一项工作 [6] 提出了一种用于大规模自主机器人网络中的 TAPF 问题的方法。该方法通过结合环境中预定区域的机器人密度和延误概率来预测机器人的旅行时间。虽然该方法随机器人的数量而扩展，因此可以用于实时应用，但它必须找到环境的预定区域，而这似乎不是自主完成的。例如，每个区域必须具备通道，包括通往相邻区域的出口和入口。此外，每个连接通道必须能够同时容纳至少两个机器人。

前面提到的方法（即，[4] ，[6] ，[8] ，[9] ）存在局限性，因为它们仅在离散空间中工作。然而，我们提出的方法无需手工设计环境结构即可在连续空间中找到解决方案。我们不是明确地寻找无冲突的路径，而是通过消除反方向机器人相遇和移动机器人阻碍的情况，找到一个不太可能有机器人间冲突的分配。通过这样做，我们的方法即使在有数百个机器人和任务以及密集障碍物的环境中，也能快速找到高质量的解决方案。

3. 问题描述

在这项工作中，我们考虑了机器人导航任务，其中分配一个机器人去执行一个任务就足以完成该任务。机器人是可以互换的，因此任何机器人都能完成任何任务。我们还考虑了 MRTA 问题，该问题需要为一组 M 个任务和一组 N 个机器人进行一次性分配。[1] 我们假设地图以及机器人的位置和任务的地点对于计算解决方案的集中式代理来说是已知的。我们不考虑机器人的运动学和动力学约束。此外，两个或两个以上的任务在空间上不能重叠。同样，机器人并不是从同一位置开始的。

我们的问题可以被表述为一个指派问题，其目标是最小化由任何一个机器人执行任何任务所产生的最大成本，这可以表示为 $\mathcal{NP}$-hard [2, 12] 。在机器集 $\mathcal{R}$ 中，机器人 r_i 执行任务集 $\mathcal{T}$ 中的任务 t_j 的成本被表示为 $c_{ij} \in \mathbb{R}_{\geq 0}$ ，其中 $i \in \{1, \cdots, N\}$ 和 $j \in \{1, \cdots, M\}$ 。在我们的工作中，成本指的是机器人在运行时完成任务的实际时间，而不是规划时的估算。因此，该问题是要在最小化完工时间（即完成所有任务的经过时间）的同时找到机器人和任务的分配。在带有导航任务的机器人问题中，这种极小化最大值的指派问题更加困难，因为 c_{ij} 可能会由于机器人执行分配任务时所走的路径不同而变化。

我们引入一个决策变量 $x_{ij} \in \{0, 1\}$ ，它表示在条件 $x_{ij} = 1$ 下将 r_i 分配给 t_j 。在不失一般性的情况下，我们假设 $N = M$ 以简化问题（如果 $N \neq M$ 不平衡，可以添加虚拟机器人或任务）。给定这种表示法，具有分配结果 $\mathcal{A}$ 的 MRTA 问题可以定义为

$$\text{minimize} \quad \max\{c_{ij} x_{ij}\}, \qquad \forall\{i, j\} \tag{1}$$

，满足条件

$$\sum_{j=1}^{M} x_{ij} = 1, \qquad \forall i, \tag{2}$$

$$[4pt] \sum_{i=1}^{N} x_{ij} = 1, \qquad \forall j, \tag{3}$$

$$[4pt] x_{ij} \in \{0, 1\}, \qquad \forall\{i, j\}. \tag{4}$$

的约束。

给定一个工作空间 $\mathcal{W}$ ，我们有一组自由空间 $\mathcal{W}_f = \mathcal{W} \setminus \mathcal{O}$ ，其中 $\mathcal{O}$ 表示障碍物的集合。在 $\mathcal{W}_f$ 中，我们定义一个拓扑图 $G = (\mathcal{V}, \mathcal{E})$ ，我们称之为路线图，其中节点 $v \in \mathcal{V}$ 位于 $\mathcal{W}_f$ 中，$e \in \mathcal{E}$ 连接 $\mathcal{V}$ 中的一对节点。由于 G 是一个拓扑图，因此 G 的每个节点对应于 $\mathcal{W}_f$ 中的一个点，每条边代表连接 $\mathcal{W}_f$ 和 [13] 中两个点的路径。

因此，路线图在 $\mathcal{W}$ 中编码了可通行的空间。我们的目标是解决（1 – 4），并在导航过程中尽量减少拥堵的同时，

[1] 我们的方法可以通过反复运行所提议的方法，直到所有任务都完成，以扩展到寻找时间扩展的分配。

找到机器人到达其在路线图上分配任务的所有路径。为了实现这一目标，我们希望在机器人执行过程中满足两个约束：

- 在遍历过程中避免边缘冲突：当多个机器人在同一路线上导航时，防止它们在同一边缘上沿相反方向移动是至关重要的。通过确保边缘上的单向流动，机器人可以无碰撞风险地移动。

- 防止在任务目的地发生阻塞：当机器人到达其分配的任务时，它们不应该阻挡仍在前往目标途中的其他机器人。这保证了移动的顺畅，并在执行过程中避免了瓶颈。

4. 方法

我们提出了一种用于任务分配的框架，其中包括机器人再分配，这提供了显著的优势。与直接将机器人分配到任务的传统方法相比，我们的方法可以在机器人执行时的导航过程中减轻机器人间的冲突。

给定一个环境，所提出的方法首先生成一个路线图（即一个拓扑图）来表示环境，并将该路线图划分为几个称为组件的子图。该方法执行需求-供应分析以确定每个组件中有多少机器人是不足或过剩的。然后，重新分配计划确定哪个机器人移动到哪个组件以平衡机器人数量。重新分配计划会考虑机器人转移时所采取的路线以防止机器人之间可能的冲突。一旦重新分配完成，任务就被分配给机器人。

Approach . 为了确保无死锁的任务分配，我们的 MRTA-RM 方法在基于路线图的执行过程中利用了两个关键特性：

我们将路线图 $G = (\mathcal{V}, \mathcal{E})$ 定义为一个拓扑图，其中节点 $v \in \mathcal{V}$ 表示可遍历空间中的一个点，边 $e \in \mathcal{E}$ 表示两个节点之间的路径。给定一个被分配任务 t_j 的机器人 r_i ，其起始位置和目的地分别对应于路线图上的两个节点，记为 (v_s 和 v_d ，其中 v_d 是与分配任务 t_j 的位置匹配的节点。

r_i 的路径表示为节点 $\mathcal{P}_i = (v_\mathrm{s}^i, v_1^i, v_2^i, \ldots, v_\mathrm{d}^i)$ 的序列，每对连续节点之间的连接为路径段。

我们确保在遍历过程中满足两个主要属性：

- 性质 1：假设图 G 中的一条边 e 连接两个节点 v_a 和 v_b ，那么对于任何两个机器人 r_i 和 r_j ，如果 r_i 沿边 e 从 v_a 行驶到 v_b ，则 r_j 在任何时候都不会沿同一条边 e 从 v_b 行驶到 v_a 。因此，机器人可以避免在边上发生碰撞。

- 性质 2：给定 $\mathcal{P}_i$ 和 $\mathcal{P}_j$ ，对于任意 k ，可以有 $v_\mathrm{d}^i = v_k^j$ 。在这种情况下，当 r_i 到达 $v_k^j = v_\mathrm{d}^i$ 时，r_j 不应占据 v_k^j ，并沿着 $\mathcal{P}_j$ 中的以下节点移动。因此，到达的机器人不会阻碍其他机器人。

为了验证这两个性质，我们在第 4.6 节中提供了正式的证明，在那里我们展示了我们的方法如何通过结构化的任务分配本质上强制执行这些行为。

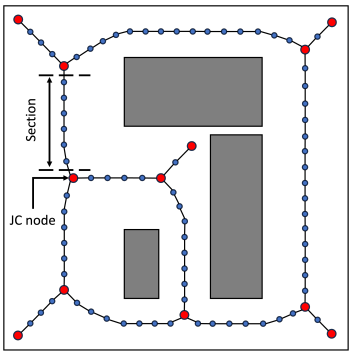


Figure 2: 一个示例路线图。红色较大的节点表示交叉节点（JC 节点）。一个区段由 JC 节点之间的非 JC 节点定义。

4.1. 环境分区

给定一个工作空间 $\mathcal{W}$ ，我们构建一个广义 Voronoi 图 (GVD)。我们采用在 [14] 中提出的方法，该方法使用来自障碍物边界的采样点来生成 GVD。该 GVD 被转化为一个初始拓扑图，该图保留了障碍物之间的拓扑关系 [13] 。在算法 1 的第 2 行中，经过后处理后生成了一张路线图 G ，比如由于障碍物而删除 GVD 中机器人（半径为 r ）无法到达的节点并均匀分布节点，如图 2 所示。

然后，我们从路线图中识别交汇节点（JC 节点）和区段。JC 节点（如图 2 中的红点所示）是与超过两个相邻节点的节点或一个终端节点（即只有一个相邻节点）。区段由两个 JC 节点之间的节点唯一定义，但不包括 JC 节点本身。设 $\mathcal{J}$ 为 JC 节点集合，$\mathcal{S}$ 为所有区段的集合。JC 节点和区段共同构成 G 的组件，其中 $\mathcal{Z} = \mathcal{J} \cup \mathcal{S}$ 。路线图及其组件保持不变，除非出现新的障碍物。即使发生一些障碍物，也只需要更新受影响的组件。

4.2. 需求和供给分析

为了分析机器人和任务之间的供需关系，我们将机器人和任务与路线图关联。每个机器人仅属于一个节点，该节点可以是某一部分中的 JC 节点或非 JC 节点。同样，一个任务也与一个节点相关联。关联是通过寻找与机器人（或任务）最近的节点来确定的，前提是连接它们的直线在 $\mathcal{O}$ 中不与任何障碍物相交。这个限制确保机器人或任务不会与一个在空间上接近但被障碍物隔开的节点（例如墙壁）关联。由于路线图是基于捕捉障碍物之间安全路径的 GVD 构建的，通过障碍物关联将导致在路线图中生成无效或不可达的链接。

在操作环境的每个部分中，节点根据其空间位置按照特定顺序排列。这个有序序列从一个末端节点开始，该节点被随机选中为起点，并被赋予初始索引为一。随后，给相邻节点分配下一个索引，从起始节点开始依次递增。该过程持续进行，直到该部分内所有节点按连续顺序进行索引。

相应地，机器人和任务根据它们与节点的空间关联在各自的区段内被索引。机器人的排序遵循节点的顺序。与区段中第一个节点相关联的机器人（或任务）获得最早的索引，依此类推。在多个机器人或任务与单个节点相连的情况下，它们的排序基于与节点的接近程度进行细化，较近的机器人或任务分别获得较低的索引。如果多个机器人（或任务）位于与节点相同的距离，则通过比较它们与关联节点连接的相邻节点的距离来打破平局。与区段不同，JC 节点不提供线性空间上下文来确定机器人或任务之间的相

对位置。因此，我们按照在路线图遍历期间遇到的顺序对与 JC 节点相关的机器人和任务进行索引。

然后，一个组件 $z_s \in \mathcal{Z}$（即，G 中的一个 JC 节点或部分）可以通过属于 z_s 的机器人和任务的数量来表征，其中 $s = 1, \cdots, |\mathcal{Z}|$ 。数量 D_s 是与 z_s 关联的任务数量减去机器人数量得到的。这一数值表示组件中有多少机器人过多或不足。供应过剩的组件 $z_s^+ \in \mathcal{Z}$ 与 $D_s > 0$ 一起。供应不足的组件 $z_s^- \in \mathcal{Z}$ 与 $D_s < 0$ 一起。其余组件 z_s° 则是平衡的，作为 $D_s = 0$ 。集合 $\mathcal{Z}^+$ 包含供应过剩的组件，而 $\mathcal{Z}^-$ 包括供应不足的组件。

4.3. 机器人重新分配计划

一旦我们完成供需分析，我们就会找到一个计划，以重新分配在所有 z_s^+ 中的过剩机器人到所有供不应求组件 z_s^- 中的过剩任务。我们使用一个加权匹配算法来计算机器人的任务分配 $\mathcal{A}$ 以找到重新分配计划 $\mathcal{X}$ 。匹配算法需要机器人执行任务的成本。通常，许多任务分配方法使用简单的成本度量，例如任务和机器人之间的欧几里得距离。然而，本工作中考虑的环境充满了障碍物。机器人要遵循的实际路径很少是直线。因此，我们使用路网来估算成本（即旅行距离），而不是使用欧几里得距离，以减少最优差距。

Algorithm 1 MRTA-RM

Input: $\mathcal{W}, \mathcal{O}, r, \mathcal{R}, \mathcal{T}$
Output: Redistribution plan $\mathcal{X}$, Categories C1, C2, C3, C4
1: $\mathcal{P} = \emptyset$
2: $G, \mathcal{J}, \mathcal{S} \leftarrow$ GenRoadmap($\mathcal{W}, \mathcal{O}, r$) ..... // described in Sec. 4.1
3: $\mathcal{Z} = \mathcal{J} \cup \mathcal{S}$
4: $\mathcal{Z}^+, \mathcal{Z}^- \leftarrow$ Analysis($G, \mathcal{O}, \mathcal{Z}, \mathcal{R}, \mathcal{T}$) ..... // described in Sec. 4.2
5: **for** each component $z_i^+ \in \mathcal{Z}^+$ **do**
6: **for** each component $z_j^- \in \mathcal{Z}^-$ **do**
7: $\mathcal{P}_{i,j} \leftarrow$ ShortestPath(z_i^+ , z_j^-) // set of the nodes
8: $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_{i,j}$
9: $c_{i,j} \leftarrow |\mathcal{P}_{i,j}|$ ..... // construct a $K \times K$ cost matrix
10: **end for**
11: **end for**
12: $\mathcal{A} \leftarrow$ Hungarian($c_{i,j}$) // $c_{i,j}$ is a cost matrix equivalent to a weighted bipartite graph for the Hungarian method
13: $\mathcal{X}_{\text{init}} \leftarrow$ RedistributionPlanning($\mathcal{A}, \mathcal{Z}$)
14: $\mathcal{X} \leftarrow$ Revising($\mathcal{X}_{\text{init}}, \mathcal{P}, \mathcal{Z}$)
15: C1, C2, C3, C4 $\leftarrow$ CategorizeComponent($\mathcal{X}, \mathcal{Z}$)
16: **return** $\mathcal{X}$, C1, C2, C3, C4

可以通过在路线图上寻找路径来获得旅行距离。为了构建代价矩阵，应找到所有盈余机器人与任务对之间的所有路径，这会带来计算开销。我们通过使用组件层次路径来逼近机器人层次路径以减少计算。对于每一对供给过剩组件 z_i^+ 和供给不足组件 z_j^- ，我们找到一条最短路径 $\mathcal{P}_{i,j}$ ，路径由连接一对组件中心的一组节点构成。组件中心被定义为属于该组件的一段中间节点或 JC 节点。我们使用 z_i^+ 与 z_j^- 之间的路径长度作为 z_i^+ 中任意机器人执行 z_j^- 中任意任务的代价。通过该逼近方法，我们可以减少在路线图中的路径规划查询次数。算法 1 的第 5 至 11 行找出供给过剩组件与供给不足组件之间的代价。

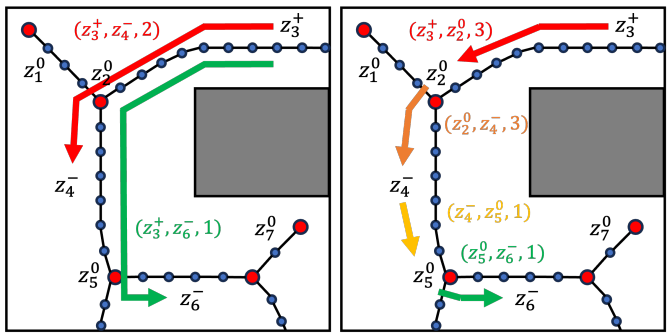
从 z_i^+ 中的剩余机器人与 z_j^- 中的任务之间的分配 $\mathcal{A}$ 出发，针对所有 i 和 j 的组合，我们找到了初始重新分配计划 $\mathcal{X}_{\text{init}}$ 。在算法 1 的第 13 行中，我们搜索所有 $\mathcal{A}$ 中的分配，以找到从 z_i^+ 到 z_j^- 移动的机器人数量，用 N_{ij} 表示。结果，我们得到了 $\mathcal{X}_{\text{init}} = \{(z_i^+, z_j^-, N_{ij}) \mid i, j \text{ are respective component indices}\}$ 。$\mathcal{X}_{\text{init}}$ 中的每个元组称为流动，描述了起始和目标组件以及要在组件之间转移的机器人数量。

Algorithm 2 修订

Input: $\mathcal{X}_{\text{init}}, \mathcal{P}, \mathcal{Z}$
Output: Revised redistribution plan $\mathcal{X}$
1: $\mathcal{X} \leftarrow \emptyset$
2: **for** each flow $(z_i^+, z_j^-, N_{i,j}) \in \mathcal{X}_{\text{init}}$ **do**
3: $\mathcal{Z}_{i,j} \leftarrow$ Decompose($\mathcal{P}_{i,j}, \mathcal{Z}$) // $\mathcal{Z}_{i,j}$ is a array include sequence of components (e.g., $(z_i^+, \cdots, z_j^-)$)
4: **for** $k = 1$ to $|\mathcal{Z}_{i,j}| - 1$ **do**
5: Add $(\mathcal{Z}_{i,j}[k], \mathcal{Z}_{i,j}[k+1], N_{i,j})$ to $\mathcal{X}$
6: **end for**
7: **end for**
8: $\mathcal{X} \leftarrow$ Merge($\mathcal{X}$)
9: **return** $\mathcal{X}$

计划 $\mathcal{X}_{\text{init}}$ 可以通过让机器人沿着在 7 中找到的最短路径 $\mathcal{P}_{i,j}$ 进行来执行。执行该计划后，所有组件的需求和供应可以得到平衡 $D_s = 0$ 。然而，$\mathcal{X}_{\text{init}}$ 在机器人重新分配时不考虑组件之间的交通。为了缓解这一问题，我们修订了 $\mathcal{X}_{\text{init}}$ ，以减少机器人的总行驶距离，从而降低重新分配期间的整体交通。

例如，最初的计划 $\mathcal{X}_{\text{init}} = \{(z_3^+, z_4^-, 2), (z_3^+, z_6^-, 1)\}$ 在图 3a 中表明总共有三个机器人从 z_3^+ 移动到其他组件。其中一个机器人到达 z_6^- ，而另外两个则停在 z_4^- 。如果 z_4^- 中至少存在一个机器人，其中之一可以被发送到 z_6^- ，这样来自 z_3^+ 的三个机器人都可以停在 z_4^- 。所有的流动可以修改为相邻组件的流动。这种修改将机器人所经过的最大组件数从四个减少到一个。如果没有与 z_2° 和 z_4^- 相关的机器人，那么来自 z_3^+ 的三个机器人中的一个将直接被发送到 z_5° 。修改后的计划 $\mathcal{X} = \{(z_3^+, z_2^\circ, 3), (z_2^\circ, z_4^-, 3), (z_4^-, z_5^\circ, 1), (z_5^\circ, z_6^-, 1)\}$ 描述在图 3b 。



(a) 初始再分配计划 $\mathcal{X}_{\text{init}}$ (b) 修订的重新分配计划 $\mathcal{X}$

Figure 3: 修订初始再分配计划为最终再分配计划的示例。修订后，流动被分解为组件流动。

算法 2 中给出的修订过程使用了 $\mathcal{P}_{i,j}$ 、$\mathcal{Z}$ 和 $\mathcal{X}_{\text{init}}$ 。假定按照此计划移动的机器人遵循 $\mathcal{P}_{i,j}$ 。从 $\mathcal{P}_{i,j}$ ，我们可以确定一个由 $\mathcal{Z}_{i,j}$ 表示的组件序列，$\mathcal{P}_{i,j}$ 经过这些组件（见行 3 ）。换句话说，如果在 $\mathcal{X}_{\text{init}}$ 中执行一个流，$\mathcal{Z}_{i,j}$ 代表机器人要通过的组件。在例子中，图 3a 的 $(z_3^+, z_6^-, 1)$ 拥

有 $\mathcal{Z}_{3,6} = (z_3^+, z_2^\circ, z_4^-, z_5^\circ, z_6^-)$，表示从 z_3^+ 到 z_6^- 的路径包含四段，其中一段为从 z_3^+ 到 z_2°，从 z_2° 到 z_4^-，从 z_4^- 到 z_5°，以及另一段从 z_5° 到 z_6^-。基于 $\mathcal{Z}_{3,6}$，$(z_3^+, z_6^-, 1)$ 被分割成 $(z_3^+, z_2^\circ, 1)$、$(z_2^\circ, z_4^-, 1)$、$(z_4^-, z_5^\circ, 1)$ 和 $(z_5^\circ, z_6^-, 1)$（见行 4 – 6）。这种分割对 $\mathcal{X}_{\text{init}}$ 中的所有流重复进行。所有结果流都被插入到 $\mathcal{X}$ 中。

如果有多个流具有相同的起始和结束组件，通过将相同起始和结束的传输机器人数量相加来合并它们。在运行示例中，$\mathcal{X} = ((z_3^+, z_2^\circ, 2)$、$(z_2^\circ, z_4^-, 2)$、$(z_3^+, z_2^\circ, 1)$、$(z_2^\circ, z_4^-, 1)$、$(z_4^-, z_5^\circ, 1)$、$(z_5^\circ, z_6^-, 1))$ 有两个流，从 z_3^+ 发送机器人到 z_2°，从 z_2° 发送到 z_4^-。合并后，我们有 $((z_3^+, z_2^\circ, 3)$、$(z_2^\circ, z_4^-, 3)$、$(z_4^-, z_5^\circ, 1)$、$(z_5^\circ, z_6^-, 1))$（第 8 行）。

4.4. 机器人再分配的优先流执行

$\mathcal{X}$ 中的流并不指定哪个流先执行。此外，它们不确定是哪个机器人执行哪些任务，而仅仅是指明转移的数量。因此，我们进一步处理 $\mathcal{X}$ 以执行再分配计划并发现机器人的任务分配。对于此过程，我们根据组件中机器人流的模式的特征，引入四类组件 C1 到 C4（算法 1 的第 15 行），如表 1 所汇总。

由于属于 C1 的组件没有任何进出的机器人，$\mathcal{X}$ 没有 C1 组件。C2 组件只有向外的机器人，因此它们仅作为 $\mathcal{X}$ 中的起始组件出现。类似地，C3 组件只有向内的机器人，并仅作为 $\mathcal{X}$ 中的结束组件出现。最后，C4 组件同时具有进出机器人，因此它们在 $\mathcal{X}$ 中既作为起始组件又作为结束组件出现。因此，C2 和 C3 组件分别作为源和汇工作。C4 组件管理从源到汇的交通。在运行示例（图 3）中，我们有 $\mathcal{X} = ((z_3^+, z_2^\circ, 3)$、$(z_2^\circ, z_4^-, 3)$、$(z_4^-, z_5^\circ, 1)$、$(z_5^\circ, z_6^-, 1))$，$z_3^+$ 是一个 C2 组件，因为它仅作为起始组件（所以在元组的第一个位置）。另一方面，z_6^- 是一个 C3 组件，而 z_2°、z_4^- 和 z_5° 是一个 C4 组件。其余未出现的组件（即 z_1°, z_7°）属于 C1。

Table 1: 区分组件的类别特征。

Category	C1	C2	C3	C4
Incoming robot?	×	×	✓	✓
Outgoing robot?	×	✓	×	✓

根据 $\mathcal{X}$ 和分类，我们进行规划，为每个机器人找到一个节点序列，这些节点作为路径点来引导机器人沿着移动。然后，到达指定部件的机器人将被分配到各个部件的任务中。

我们首先确定 $\mathcal{X}$ 中流的优先级。然后，我们假想地移动机器人作为一种规划过程，以找到所有机器人的节点序列。我们根据流中起始组件的类别来确定流的优先级。由于 C1 和 C3 组件由于其特性（即两者都不发送任何机器人）从不作为起始组件，我们只考虑从 C2 和 C4 组件开始的流。由于 C4 组件既有进入又有离开的机器人，因此从 C4 组件开始的流在它们接收到来自其他组件的所有机器人后才执行。

由于送到 C4 的机器人有可能进一步转移到其他组件，而 C3 则没有这样的可能，因此应该优先将机器人发送到 C4 以减少系统的总完成时间。因此，我们首先执行从 C2 流向 C4 组件的 [2] 流程。我们在 $\mathcal{X}$ 中搜索以找到这些流程，并在每次找到它们时依次顺序执行。

[2]如前所述，该执行假设作为一个规划过程进行。

为了防止机器人实际移动时出现死锁，也必须确定应该发送哪个机器人以及每个接收组件中的机器人的到达顺序。当决定从一个起始组件发送哪个机器人时，我们考虑路线图的配置。如果该组件的起始节点直接连接到目标组件，则选择距离起始节点最近的机器人先移动。如果连接是从起始组件的终端节点到目标组件，那么距离终端节点最近的机器人将首先移动。此外，在所有到达接收组件的机器人中，距离该组件最近的机器人应该先到达。一旦从 C2 到 C4 组件的所有流都被执行后，再搜索并执行从 C2 到 C3 组件的流。

随后，我们搜索 $\mathcal{X}$ 以执行从一个 C4 组件到另一个 C4 的流。在顺序执行这些流的过程中，一些 C4 组件可能仍在接收来自 C4 组件的机器人。它们不会被选择执行，直到完全接收到所有的机器人。完成所有 C4 组件之间的流之后，继续执行从 C4 组件到 C3 组件的流。当一个 C4 组件向其他组件发送机器人时，从一开始就属于该组件的机器人会以与我们选择 C2 组件中要发送的机器人的方式相同的方式被选择。如果所有这些机器人都已发送完毕，但仍需发送更多机器人，则选择其他组件的机器人按照它们在路线图上行驶距离的顺序发送。

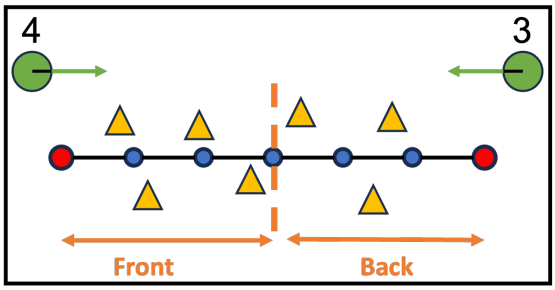
4.5. 组件内的任务分配

在假想执行所有流动到重分配计划 $\mathcal{X}$ 后，我们知道每个机器人的目标组件。因此，根据机器人的到达顺序来确定每个组件中机器人与任务之间的分配，行驶距离较短的机器人被分配到离进入 JC 节点 [3] 较远的任务。这种先到先得的分配方式可以防止机器人阻塞到达后续任务的其他机器人路径。如果有多个机器人同时到达该节点，可以通过随机方式来打破平局。

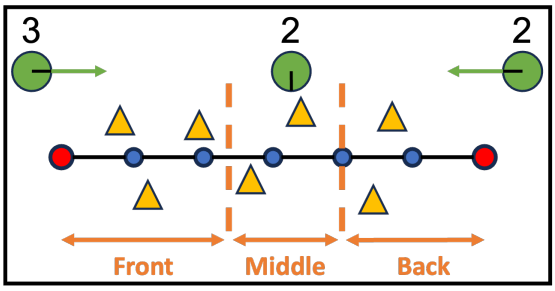
可能会有机器人从两个方向进入一个区域。如果将机器人分配到更接近它们进入方向的任务，来自两个方向的交通就不会混合，从而减少冲突。具体来说，我们根据它们在区域中的索引（即，前组和后组，如图 4a 中橙色部分所示）将该区域的任务分为两组，依据是从每个方向到达的机器人数。因此，从区域起始节点进入的机器人被分配到前任务组，反之亦然。如果区域中有一些机器人没有被转移，任务则分为三组（即，前、中、后任务组，如图 4b 中的橙色部分所示）。因此，从区域起始和结束节点进入的机器人分别被分配到前任务组和后任务组。在区域中停留的机器人被分配到中间任务组，从索引最低的机器人和任务开始，直到最高索引。

当一个部分只接收到来自一侧的机器人并且一些机器人驻留在这个部分时，我们也将任务划分为两组。另一方面，当这个部分只有一个方向接收的机器人或只有驻留在部分中的一个机器人时，我们仅将任务划分为一组。此外，与 JC 节点相关的任务作为一组处理。最后，在我们的方法中，我们为每个机器人提供一条推荐路径，该路径仅由从路线图上的节点序列中派生的 JC 节点组成。此路径用作参考，以引导机器人完成分配的任务，同时使其能够使用自身的本地控制器在连续空间中自主导航。因此，机器人并不严格按照路线图上的所有节点行进，而是使用基于 JC 节点的参考航路点来避免可行轨迹中的障碍物和拥堵。

[3]正如我们假设的那样，没有考虑机器人运动学约束，因此到达时间可以通过行驶距离和速度计算得出。



(a) 机器人从两端进入时的任务分组。根据机器人进入的方向，任务被分为两个组（前部和后部）。



(b) 当同时存在外部机器人和本地机器人时的任务分组。任务被分为三组（前、中、后）以减少交叉流量并实现平稳的任务分配。

Figure 4: 章节中任务分组的示例。黄色三角形表示与该章节相关的任务，绿色圆圈表示机器人。此外，绿色箭头表示机器人进入的方向。橙色虚线表示任务组边界。

4.6. 分析

Theorem 4.1. 对于至少有一个机器人通过的 JC 节点，机器人以一个方向通过该 JC 节点。

Proof 。作为一个基本案例，我们考虑两个相邻的部分 z 和 z'，因此一个 JC 节点连接它们。令 $\mathcal{X}_{\text{init}}$ 为从分配 $\mathcal{A}$ 在 Sec. 4.3 获得的重新分布计划。从 Sec. 4.3 中的 $\mathcal{X}_{\text{init}}$ 的修订过程中，我们得到的 $\mathcal{X}$ 仅在相邻组件之间分配机器人流。

假设重新分配计划要求机器人从相反方向穿过连接 z 和 z' 的 JC 节点。换句话说，$\mathcal{X}$ 有流量通过 JC 节点将机器人从 z 移动到 z'，以及通过同一 JC 节点将机器人从 z' 移动到 z 的其他流量。这些流量导致将机器人从 z 发送到 z'，同时也从 z' 发送到 z。在这种重新分配之后，两部分的任务都可以由来自对方部分的机器人完成。设这种分配的成本为 c。

另一方面，我们可以通过让机器人在同一个组件中执行任务来获得另一种比 c 更便宜的分配。由于 $\mathcal{X}$ 是 $\mathcal{A}$ 最小化任务分配成本的结果，$\mathcal{X}$ 不能有这样的反向流动产生更高的成本。因此，假设机器人从相反方向穿过一个 JC 节点的观点是相悖的。

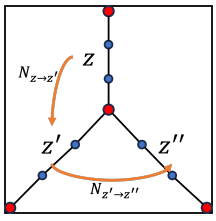


Figure 5: 定理 4.1 的例子环境

让我们考虑另一种情况，其中三个相邻的段 z、z' 和 z'' 连接到一个 JC 节点，如图 5 所示。假设由 $\mathcal{A}$ 计算得出的 $\mathcal{X}$ 有一些流动，经过 JC 节点将 $N_{z\to z'} \in \mathbb{N}$ 个机器人从 z 移动到 z'，另一些流动同样通过 JC 节点将 $N_{z'\to z''} \in \mathbb{N}$ 个机器人从 z' 移动到 z''。在这种情况下，我们可以进行另一种分配，使机器人行驶更短的距离。具体来说，当 $N_{z\to z'} = N_{z'\to z''}$ 时，直接将 N 个机器人从 z 转移到 z'' 可以减少成本。对于 $N_{z\to z'} > N_{z'\to z''}$，当将 $N_{z\to z'} - N_{z'\to z''}$ 个机器人从 z 转移到 z' 和将 $N_{z'\to z''}$ 个机器人从 z 转移到 z'' 时，成本可能更低。相反，如果 $N_{z\to z'} < N_{z'\to z''}$，则将 $N_{z\to z'}$ 个机器人从 z 移动到 z'' 和将 $N_{z'\to z''} - N_{z\to z'}$ 个机器人从 z' 移动到 z'' 是更具成本效益的。由于 $\mathcal{A}$ 使分配的成本最小化，这个假设与可能存在更低成本的分配相矛盾。

总之，如果一个 JC 节点至少向一个区域传递了一台机器人，那么该区域接收到的机器人不得通过该 JC 节点向其他区域发送机器人。因此，没有一对区域的机器人会在相反方向上穿过同一个 JC 节点。 □

Theorem 4.2. 对于路线图中的每条边，没有机器人在相反方向穿过该边。

Proof 。如在 Sec. 4.5 中所述，我们将一个部分中的任务分为一组、两组（即前、后）或三组（即前、中、后）。然后，来自其他组件的机器人以及非来自任何其他组件的本地机器人被分配到相应的任务组。

当机器人仅从一侧进入时，机器人的移动自然是单向的。此外，当机器人从截面的两侧进入时，任务被一分为二，因此机器人向前后任务组的移动也减少为只有单向移动的第一种情况。

对于本地机器人，我们遵循第 4.2 节中描述的一种基于索引的排序策略，其中剩余的机器人被顺序地从最低索引的机器人分配到最低索引的任务。这个索引反映了部分中节点的空间顺序。尽管机器人可能会根据各自的任务位置在该部分内朝不同的方向移动，分配机制确保了它们的路径不会相交或互相阻挡。这是因为机器人和任务是根据其空间索引按顺序分配的，因此每个机器人都被匹配到序列中最近的未分配任务。结果，即使移动方向不同，产生的路径仍然是互不相交的，在包括 JC 节点连接的所有边中保持非干扰性。 □

Theorem 4.3. 当机器人沿着路线图上的参考航点前往其指定任务时，最先到达的机器人不会阻碍其他机器人。

我们假设机器人的到达顺序是根据其行驶距离来计算的，并且遵循这一顺序。在此假设下，同一侧的任意一对机器人不会相互阻挡，因为先到先得的分配机制使得最先到达的机器人移动到相应任务组中最远的任务（即，前方任务组或后方任务组）。

此外，对于任何本地机器人，尽管在该部分中其他本地机器人或任务可能位于机器人和其分配任务之间，但没有分配的机器人-任务对的元素都位于该机器人与其目的地之间。这是因为本地机器人和任务是按照索引顺序分配的，确保没有后分配的对会阻碍之前分配的机器人的路径。因此，本地机器人不会互相阻挡。

此外，对于每个 JC 节点，最先到达的机器人被转移到相邻的区域。剩余的机器人数量与与 JC 节点相关的任务数量相匹配，然后分配给这些任务。因此，每个 JC 节点中的剩余机器人永远不会阻碍其他机器人的移动。 □

在基于连通 GVD 的路线图中，MRTA-RM 中的流程执行是完整的。

Proof 。我们在第 4.4 节证明流执行的完整性。首先，我们考虑从 C2 中的组件开始的流首先被执行。由于这些流不需要任何条件，因此它们可以始终无限制地进行。

然而，对于从 C4 中的组件开始的流，所有机器人必须在流可以执行之前收到。因此，为了证明完整性，我们展示了系统无法达到这样一种状态，即剩余的流中没有一个能够满足其执行条件。

假设在流程执行期间，系统达到了一种状态，在这种状态下，剩余的流程都无法满足其执行条件。设此时未执行的流程集合为 $\mathcal{F}_l$ ，相应的起始组件集合为 $\mathcal{Z}_l$ 。在这种情况下，$\mathcal{Z}_l$ 中的每个组件尚未完全接收机器人。

在这种情况下，由于所有来自 C2 组件的流都已执行，至少有一个流未包含在 $\mathcal{F}_l$ 中，并正流向 $\mathcal{Z}_l$ 中的某个组件。这意味着至少有一个 C4 中的组件未包含在 $\mathcal{Z}_l$ 中，并未达到执行条件。这与初始状态相矛盾。因此，我们得出结论，不能存在这样的状态，并且所有流最终将被执行，确保流执行的完整性。 □

根据定理 4.1–3，假如机器人能够准确地沿路线图的路径行进，那么第 3 节中的性质 1 和 2 便得以满足。换句话说，机器人不会朝相反方向移动，也不会阻挡其他移动的机器人。因此，机器人可以无冲突地到达分配给它们的任务。然而，如果机器人偏离路线图中的路径，那么无法保证无冲突的路径。如果机器人选择了不与计划的流一致的替代路线，这种偏离可能会发生，并可能使得在执行流和任务分配期间使用的假设失效，比如行进距离和进入方向。为此，要么应引导机器人沿计算出的参考航点移动，要么可以使用如 MAPF 求解器的冲突感知运动规划器来计算保留无死锁性质的路径。

Theorem 4.5. MRTA-RM 在多项式时间内执行。

我们分析了 MRTA-RM 的计算复杂性，涉及机器人数量 N 、组件数量 $|\mathcal{Z}|$ 、以及路线图 $G = (\mathcal{V}, \mathcal{E})$ 中节点数量 $|\mathcal{V}|$ 。

在第 4.2 节（Alg. 1 中的第 4 行）的需求和供应分析具有 $O(N|\mathcal{V}|)$ 的时间复杂度，因为它涉及为每个机器人和任务寻找 G 中的最近节点。

在机器人再分配计划中（第 4.3 节），为了构建算法 1 中第 5 到第 11 行的成本矩阵，采用了 Dijkstra 算法来查找供应过剩组件和供应不足组件之间的最短路径，其时间复杂度为 $O(|\mathcal{V}|^2)$ 。在这部分中，由于章节包含多个顶点，$|\mathcal{Z}|$ 小于 $|\mathcal{V}|$ 。因此，我们将 G 压缩为按组件划分的图，将时间复杂度降低到 $O(|\mathcal{Z}|^2)$ 。

对于一个分配 $\mathcal{A}$ ，我们采用匈牙利方法，该方法在 $O(N^3)$ 中计算。为了从 $\mathcal{A}$ 制定初始机器人重新分配计划 $\mathcal{X}_{\text{init}}$ ，我们确定在每个分配对中，过量供应的组件中的机器人被分配到供给不足的组件。因此，算法 1 中的行 13 的复杂度为 $O(N)$ 。算法 2 中的修订过程，将 $\mathcal{X}_{\text{init}}$ 转化为 $\mathcal{X}$ ，涉及分解和合并序列。分解 $\mathcal{X}_{\text{init}}$ 中的每个流的复杂度为 $O(N|\mathcal{Z}|^2)$ ，其中 $|\mathcal{Z}|^2/2$ 代表组件之间的潜在连接数量。同样，合并序列也需要 $O(N|\mathcal{Z}|^2)$ 时间来处理最多 $O(N|\mathcal{Z}|^2)$ 个分解后的重新分配计划片段。在行 15 中，我们搜索每个组件以确定是否有来自于或发向它的流量在 $\mathcal{X}$ 。因此，这个过程的时间复杂度为 $O(|\mathcal{Z}|^3)$ 。

最后，在流执行步骤（第 4.4 节）中，来自 C2 的流是无条件执行的，而来自 C4 的流则需要满足一个条件。然而，由于总是至少有一个可行的流（定理 4.4），因此可以在 $O(|\mathcal{Z}|^3)$ 时间内完成，因为在最多 $|\mathcal{Z}|^2/2$ 个流中经过 $|\mathcal{Z}|$ 次迭代即可完成。

总之，MRTA-RM 的计算复杂度为 $O(N|\mathcal{V}| + \max(N^3, |\mathcal{Z}|^3))$ 。 □

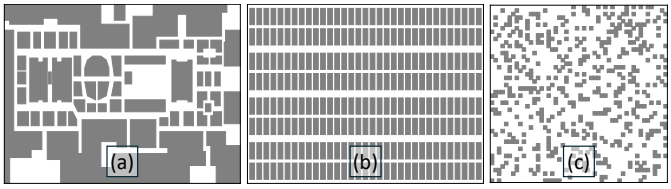


Figure 6: 用于评估的环境：(a) 购物中心（1760×900 单位），(b) 仓库（2200×880 单位），(c) 杂乱环境（1000×1000 单位）

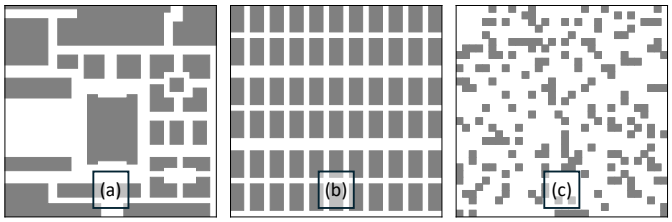


Figure 7: 用于小规模实验比较基于 CBS 的方法的环境的一部分。所有环境的大小为 640×640 单位。

5. 实验

我们有三种不同的环境，其配置如图 6 所示。第一个环境是商场，它来自百货商店的平面图，具有大厅和走廊。它有许多没有出口的长走廊，可能导致拥堵和死锁。仓库环境有许多交叉路口，机器人可能会频繁相遇。障碍环境中随机分布着大小相同的方形障碍物。这些环境的大小分别为 1760×900 、2200×880 和 1000×1000 单位。机器人的形状是圆形的，半径为六个单位。在这些环境中，根据机器人与任务的分布，我们有两种场景。随机场景中，机器人和任务随机分布在自由空间。分隔场景中，机器人在环境的左侧，而任务在右侧。后一种场景由于机器人形成大块而导致更多的机器人之间的冲突。

我们有两组不同的测试。第一组（见第 5.1 节）使用 CBS 来解决机器人之间的冲突，并将我们提出的方法与其他方法进行比较。基于搜索的方法显式地解决冲突，这需要额外的计算，因此它们通常无法随着机器人数量的增加而扩展。因此，我们最多使用 30 个机器人来运行这些方法。为了保持机器人密度高，我们仅使用图 7 所示的三个环境的一小部分。第二组（见第 5.2 节）将我们的方法与在任务分配中未考虑冲突的方法进行比较。由于这些方法简单快速，我们使用图 6 所示的三个环境的完整地图将问题实例扩展到最多 500 个机器人，这些地图我们称之为完整环境。这与图 7 中小规模测试使用的局部或裁剪区域形成对比。

为了在测试中考虑机器人间的冲突，我们使用一个动态机器人模拟器。我们使用 Unity 引擎，其中机器人的导航由 Unity NavMesh 完成，这使机器人能够找到路径并沿路径导航。被分配到任务的机器人在其动态（例如，加速度、减速度）被适当建模的环境中导航。每个机器人运行一个内置的局部碰撞规避算法，该算法使用互补速度障碍（RVO）算法 [15] 。虽然 RVO 可以处理局部碰撞，但在狭窄通道中无法解决僵局情况。

测试的性能指标包括计算成功率、计算时间、成功率、总完工时间以及成本总和（SoC）。计算成功率统计在 5 分

钟时间限制内解决的实例数量。计算时间表示解决成功实例的平均耗时。因此，如果成功实例较少，计算时间可能会有偏差。成功率统计在任务分配中成功执行且无死锁的次数。换句话说，计算成功率衡量方法的计算效率，而成功率衡量分配质量以防止死锁。死锁被定义为任何机器人停滞五秒钟的情况。总完工时间衡量所有机器人完成所有任务的总耗时。这是我们需要优化的主要目标值。我们还测量 SoC，即所有机器人的行程时间总和。

所有实验均在配备 AMD 5800X 3.8GHz CPU、32GB RAM 和 Python 3.11 的系统上完成。

5.1. 与明确解决冲突的方法的比较

我们将我们的方法与使用 CBS 的任务分配算法进行比较，例如在 Sec. 2 中回顾的 CBS-TA 和 ECBS-TA。CBS-TA 是当前 MRTA 中考虑导航拥堵的最先进（SOTA）方法之一。加上其增强版本 ECBS-TA，CBS-TA 因其在离散环境中的有效性而得到广泛认可。由于它们仅在离散空间中工作，我们将图 7 中显示的环境离散化为网格，其中每个单元格为 20 × 20 单位。我们还开发了 TA-CCBS 和 TA-CECBS，它们分别解决任务分配和路径规划。这两种后续方法使用 MRTA-RM 计算分配，然后运行由 [16] 提供的 CBS 和 ECBS 的连续变体，即连续 CBS（CCBS）和连续 ECBS（CECBS）。虽然也可以使用其他方法，但由于 MRTA-RM 在我们的实验中表现出最佳性能，因此用于任务分配，使我们能够在 CCBS 和 CECBS 的路径规划能力之前最大化任务分配过程的有效性。在 TA-CCBS 和 TA-CECBS 中，CBS 和 ECBS 在概率路网（PRM）上运行以在连续空间中寻找路径。我们选择 PRM 而不是基于 GVD 的路网是因为 GVD 通常在宽阔的走廊中产生单一代表路径，这可能限制 CBS 风格方法中解决冲突所需的灵活性。PRM 提供了更丰富的替代路径集，允许在 CBS 执行过程中更有效的重新规划。这些方法首先使用 MRTA-RM 计算任务分配，然后生成无碰撞的路径来执行分配。我们将机器人的数量和任务 N 从 5 增加到 30，间隔为 5。我们为每个环境、场景和 N 的组合测试 50 个随机实例。

对于每个环境，应构建一个路线图。这个路线图对于所有情境和 N 的组合都是相同的，因此对于每个环境只需要构建一次。例如，我们只需构建一次路线图就可以测试购物中心的 600 个实例（两个情境、六个 N 的值，以及每个组合的 50 个实例）。因此，我们将这视为一个初步步骤，所以不包括在计算时间的测量中。购物中心、仓库和混乱环境的这一步骤分别需要 0.39 秒、1.08 秒和 3.08 秒。[4]

在这个实验中，我们将 ECBS-TA 和 TA-CECBS 的次优界设置为 1.1。为了构造 CCBS 和 CECBS 的 PRM，我们采样 7000 个点，并将每个点的邻居数量设置为 15。

如图 8 和表 2 所示，我们提出的 MRTA-RM 在每个实例中都在 0.2 秒内成功找到了解决方案。相比之下，当机器人数增加时，CBS-TA 和 ECBS-TA 需要显著更多的计算时间。例如，虽然 CBS-TA 和 ECBS-TA 在随机场景中有 30 个机器人时至少达到了 60 % 的成功率，但它们在分隔场景中的所有实例中无法在 5 分钟的时间限制内找到任何解决方案。这种差异可能源于在分隔场景中由多个机器人共享的路径引发的更频繁的冲突。

[4]即使包括了这段生成路线图的时间，我们的方法在大多数情况下仍然明显优于比较的方法。

对于计算时间来说，在随机场景中，MRTA-RM 相比 CBS-TA 减少了至少 19.32 % 的计算时间，较 ECBS-TA 减少了 29.43 % 的计算时间。涉及 TA-CCBS 和 TA-CECBS 在连续空间的实验显示，成功率较低于在栅格空间中使用的 CBS-TA 和 ECBS-TA，因为 PRM 中的搜索空间更大。此外，TA-CCBS 和 TA-CECBS 的计算时间比 MRTA-RM 更高。在购物中心环境中的分离场景中，TA-CCBS 平均需要 16.88 秒处理五个代理，92.40 秒处理十个代理，而 TA-CECBS 需要 20.29 秒处理五个代理，78.70 秒处理十个代理。相比之下，MRTA-RM 只需要 0.01 到 0.05 秒，表明比 TA-CCBS 和 TA-CECBS 的计算时间减少了超过 99 %。

在动态模拟中，由于基于 CBS 的方法旨在寻找无碰撞路径，它们的解决方案可以在没有任何冲突或死锁的情况下执行。因此，只要它们在时间限制内找到解决方案，成功率就为 100 %。然而，我们的方法无法实现无死锁，因为我们的研究目的是找到一种可能防止死锁的任务分配，但并不直接着手寻找无碰撞路径。尽管如此，我们的方法在随机场景中实现了超过 96 % 的高成功率。此外，在分离场景中，成功率在 58 到 100 % 之间，而所有基于 CBS 的方法在时间限制内几乎都无法计算出任何解决方案，因为它们的可扩展性有限。

结果还表明，与基于 CBS 的方法相比，MRTA-RM 实现了更小的完工时间和 SoC。具体来说，MRTA-RM 相比 ECBS-TA 最多减少了 32 % 的完工时间并改善了 36 % 的 SoC。与 CBS-TA 相比，其中一个五个机器人参与的仓库环境实例中，CBS-TA 的完工时间比 MRTA-RM 小 1 %。然而，在其他所有场景中，MRTA-RM 始终实现了更小的完工时间，差异最多为 24 %。在 SoC 方面，MRTA-RM 相比 CBS-TA 达到了最多 21 % 的改善。

这些差异在涉及 TA-CCBS 和 TA-CECBS 的实验中变得更加明显，这些实验在连续空间中使用 PRM 进行测试。与 TA-CCBS 相比，MRTA-RM 至少减少了 24 % 的完工时间和 30 % 的 SoC。与 TA-CECBS 相比，MRTA-RM 在完工时间上至少减少了 22 %，并在 SoC 上减少了 32 %。

基于 CBS 的方法的主要缺点是计算时间，因为它们必须在找到任务分配后计算所有机器人的路径。此外，由于 CBS 和 ECBS 的设计，它们在完工时间和 SoC 等其他指标上性能不如我们的。在 CBS 和 ECBS 的连续变体中，PRM 被用来代替网格单元，以更灵活地表示环境。在网格空间中，每个机器人在每个时间步都有相同的移动距离。然而，这种统一的移动距离在拓扑图（例如，PRM）中并不成立，因为拓扑图可以具有不同的边长。因此，在每个时间步中，一些移动较短边的机器人必须等待其他机器人完成在该时间步的移动，这导致了低效。即使在这些已知的 CBS 方法的限制下，我们的方法仍然在完工时间和 SoC 上表现出色，显示出其在每个测试环境中的稳健性和效率。

尽管 CBS-TA 在理论上对 TAPF 问题是最优的，但它的最优性仅在离散网格环境中得到保证。相比之下，我们的方法在连续空间中操作，通过分配任务并推荐由 JC 节点组成的参考航点，机器人使用局部控制器遵循这些航点。这些控制器允许更平滑的导航和灵活的运动调整，通常导致更低的实际执行成本，如完工时间和总成本，尤其是在密集场景中。

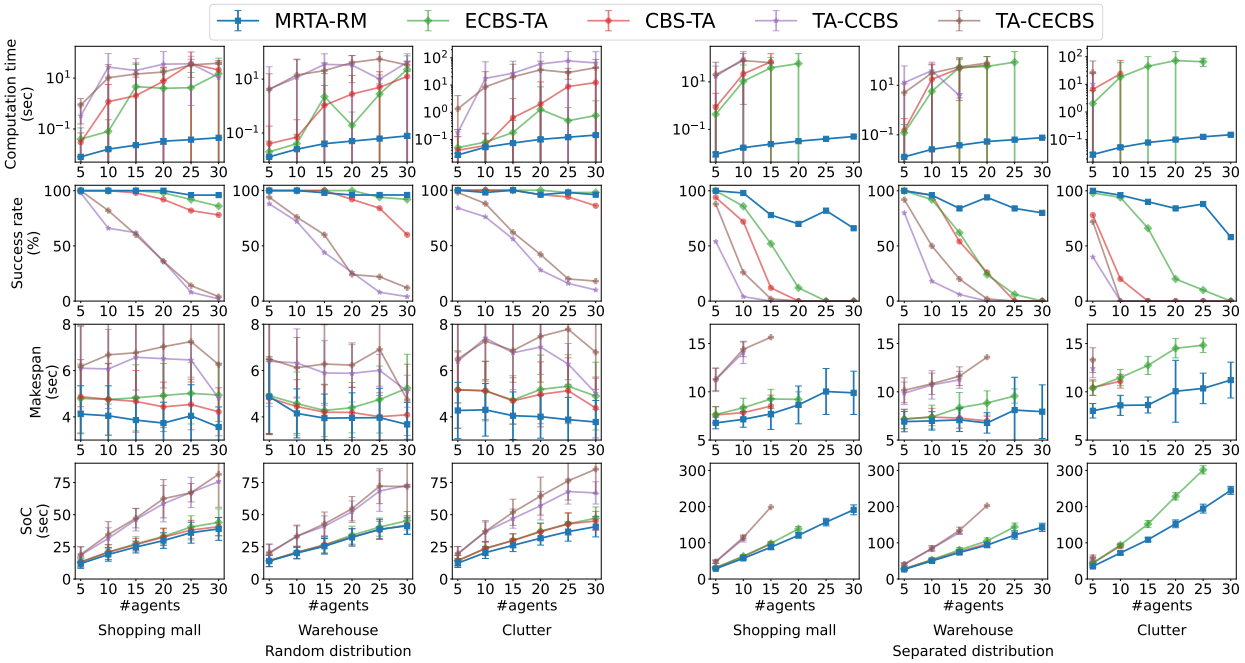


Figure 8: 与那些显式解决冲突的方法的比较结果。这里的成功率指的是在计算时间限制内，该方法成功计算出解决方案并在动态模拟中完成分配任务而未遇到任何死锁的实例百分比。我们提出的方法在所有指标上均优于所有比较的方法。

Table 2: 与那些明确解决冲突的方法的比较结果。由于篇幅限制，标准差未被列出。

Distribution		Random														
Method		CBS-TA			ECBS-TA			TA-CCBS			TA-CECBS			MRTA-RM		
# Robots		10	20	30	10	20	30	10	20	30	10	20	30	10	20	30
Comp. Success rate (%)	Shopping mall	100	92	78	100	98	86	66	36	2	82	36	4	100	100	100
	Warehouse	100	92	60	100	100	92	72	26	4	76	24	12	100	100	100
	Clutter	100	96	86	100	100	98	76	28	10	88	42	18	100	100	100
Comp. Time (sec)	Shopping mall	1.18	7.62	20.88	0.08	3.97	14.91	27.07	35.36	10.55	10.31	17.33	38.69	0.02	0.03	0.04
	Warehouse	0.07	2.79	12.64	0.04	0.19	22.92	11.52	33.35	41.95	14.25	41.11	33.38	0.03	0.05	0.08
	Clutter	0.06	2.01	12.37	0.08	1.27	0.76	17.54	59.71	64.84	8.57	36.30	43.41	0.05	0.10	0.15
Success rate (%)	Shopping mall	100	100	100	100	100	100	100	100	100	100	100	100	100	100	96
	Warehouse	100	100	100	100	100	100	100	100	100	100	100	100	100	96	96
	Clutter	100	100	100	100	100	100	100	100	100	100	100	100	98	96	96
Makespan (sec)	Shopping mall	4.76	4.43	4.22	4.75	4.92	4.94	6.07	6.51	4.81	6.68	7.03	6.26	4.05	3.73	3.56
	Warehouse	4.45	4.19	4.09	4.57	4.40	5.24	6.33	5.88	5.11	6.12	6.23	4.74	4.15	3.96	3.67
	Clutter	5.13	4.97	4.38	5.12	5.19	4.90	7.41	7.02	5.02	7.28	7.48	6.79	4.31	4.01	3.78
SoC (sec)	Shopping mall	20.84	32.55	40.65	20.91	33.07	44.03	31.58	58.55	75.71	34.57	62.42	81.38	19.07	29.86	38.97
	Warehouse	20.68	32.77	42.06	20.76	34.12	45.33	33.33	52.29	72.78	33.10	54.62	71.83	20.18	32.33	41.33
	Clutter	23.77	36.72	45.11	23.76	37.17	47.24	36.68	56.95	66.91	36.97	64.44	85.35	20.50	31.69	40.70
Distribution		Separate														
Method		CBS-TA			ECBS-TA			TA-CCBS			TA-CECBS			MRTA-RM		
# Robots		10	20	30	10	20	30	10	20	30	10	20	30	10	20	30
Comp. Success rate (%)	Shopping mall	72	0	0	86	12	0	4	0	0	26	0	0	100	100	100
	Warehouse	96	26	0	92	24	0	18	0	0	50	2	0	100	100	100
	Clutter	20	0	0	94	20	0	0	0	0	0	0	0	100	100	100
Comp. Time (sec)	Shopping mall	21.61	-	-	10.62	58.94	-	92.40	-	-	78.70	-	-	0.02	0.03	0.05
	Warehouse	17.30	65.62	-	5.69	57.16	-	38.79	-	-	30.98	78.21	-	0.03	0.05	0.08
	Clutter	24.06	-	-	18.39	72.32	-	-	-	-	-	-	-	0.05	0.10	0.15
Success rate (%)	Shopping mall	100	-	-	100	100	-	100	-	-	100	-	-	98	70	66
	Warehouse	100	100	-	100	100	-	100	-	-	100	100	-	96	94	80
	Clutter	100	-	-	100	100	-	-	-	-	-	-	-	96	84	58
Makespan (sec)	Shopping mall	7.84	-	-	8.34	9.21	-	14.09	-	-	14.40	-	-	7.15	8.64	9.89
	Warehouse	7.38	6.99	-	7.39	8.84	-	10.72	-	-	10.83	13.59	-	6.98	6.77	7.94
	Clutter	11.10	-	-	11.48	14.51	-	-	-	-	-	-	-	8.60	10.05	11.22
SoC (sec)	Shopping mall	61.67	-	-	63.12	138.69	-	110.78	-	-	114.79	-	-	56.99	120.72	191.13
	Warehouse	52.86	97.33	-	53.11	105.11	-	84.07	-	-	84.51	202.98	-	49.97	93.54	142.86
	Clutter	90.99	-	-	94.14	228.42	-	-	-	-	-	-	-	72.09	152.51	244.98

5.2. 与没有冲突解决方法的比较

为了评估我们的方法与其他不显式解决冲突的方法的可扩展性，我们包括了 Greedy-TA 和 Hungarian-TA。它们基于匈牙利方法（Hungarian-TA）和贪心方法（Greedy-TA），这些方法根据与我们相同路线图中的成本分配任务。这些方法效率高，并且能够处理大量机器人，但它们在任务执行期间不考虑冲突，使得它们与我们提出的方法从根本上不同。Greedy-TA 根据最低成本配对分配任务给机器

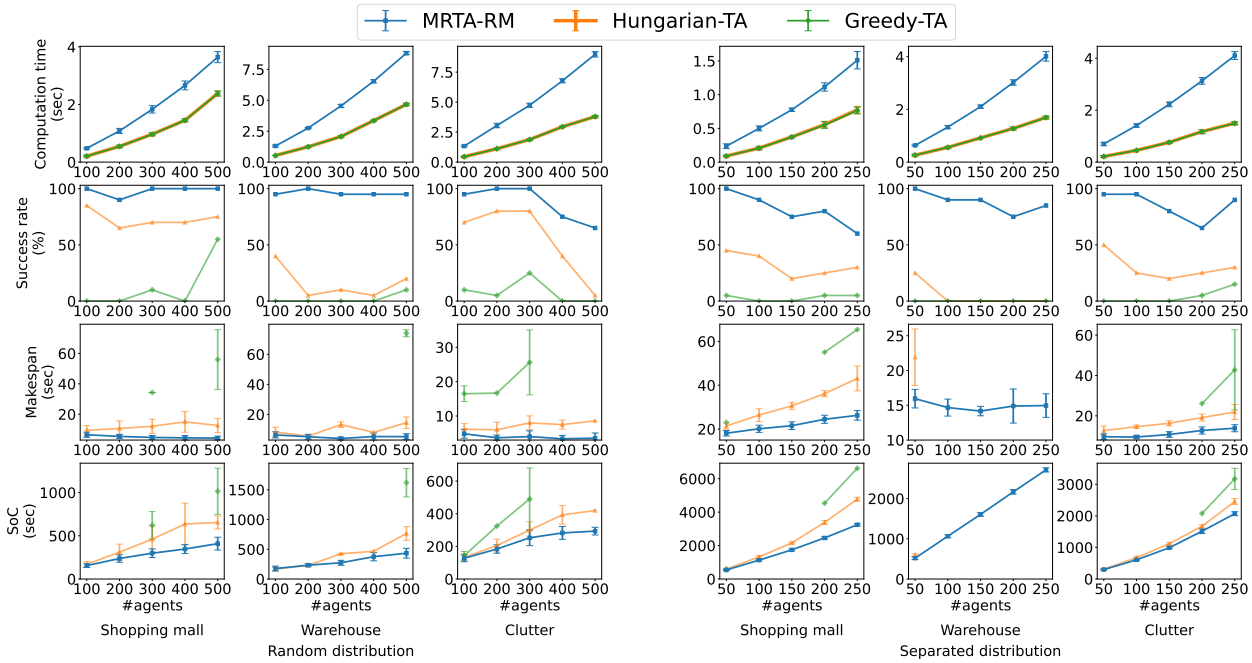


Figure 9: 与那些未明确解决冲突的方法的比较结果。我们提出的方法略慢于其他方法，但在导航中显示出显著更高的成功率。

Table 3: 与不明确解决冲突的方法的比较结果。由于空间限制，标准差被省略。

Distribution		Random									Separate								
Method		Greedy-TA			Hungarian-TA			MRTA-RM			Greedy-TA			Hungarian-TA			MRTA-RM		
# Robots		100	300	500	100	300	500	100	300	500	50	150	250	50	150	250	50	150	250
Comp. Success rate (%)	Shopping mall	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Warehouse	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Clutter	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Comp. Time (sec)	Shopping mall	0.21	0.97	2.38	0.21	0.97	2.38	0.48	1.83	3.64	0.09	0.37	0.77	0.09	0.37	0.76	0.24	0.78	1.51
	Warehouse	0.54	2.07	4.67	0.54	2.07	4.67	1.30	4.55	8.82	0.26	0.92	1.69	0.26	0.92	1.69	0.63	2.10	4.01
	Clutter	0.45	1.88	3.79	0.45	1.88	3.79	1.34	4.74	8.99	0.22	0.76	1.49	0.22	0.76	1.50	0.70	2.23	4.10
Success rate (%)	Shopping mall	0	10	55	85	70	75	100	100	100	5	0	5	45	20	30	100	75	60
	Warehouse	0	0	10	40	10	20	95	95	95	0	0	0	25	0	0	100	90	85
	Clutter	10	25	0	70	80	5	95	100	65	0	0	15	50	20	30	95	80	90
Makespan (sec)	Shopping mall	-	34.28	55.96	9.41	12.08	12.59	6.49	4.72	4.30	23.03	-	65.39	21.41	30.59	43.09	18.10	21.54	26.26
	Warehouse	-	-	74.02	8.27	13.25	14.52	6.43	4.02	5.26	-	-	-	21.91	-	-	15.95	14.17	14.96
	Clutter	16.49	25.62	-	6.14	8.00	8.59	4.83	4.01	3.53	-	-	42.78	12.68	16.30	21.89	9.65	10.75	13.85
SoC (sec)	Shopping mall	-	622.13	1015.00	170.61	460.67	654.16	155.35	298.40	409.12	599.40	-	6621.28	589.91	2157.87	4771.25	540.44	1743.76	3246.06
	Warehouse	-	-	1619.99	166.47	424.36	765.64	175.60	273.07	433.80	-	-	-	600.23	-	-	518.42	1606.22	2713.80
	Clutter	142.58	490.57	-	132.35	299.98	419.90	127.16	252.63	293.42	-	-	3171.46	311.01	1119.40	2455.43	298.73	989.88	2068.23
Total time (sec)	Shopping mall	-	35.25	58.34	9.62	13.05	14.97	6.97	6.55	7.94	23.12	-	66.16	21.50	30.96	43.85	18.34	22.32	27.77
	Warehouse	-	-	78.69	8.81	15.32	19.19	7.73	8.57	14.08	-	-	-	22.17	-	-	16.58	16.27	18.97
	Clutter	16.94	27.50	-	6.59	9.88	12.38	6.17	8.75	12.52	-	-	44.27	12.90	17.06	23.39	10.35	12.98	17.95

人，而 Hungarian-TA 则通过匈牙利方法使用一种最优分配策略。选择 Greedy-TA 和 Hungarian-TA 作为基准的动机在于它们能够扩展到极大的实例，这是我们评估的一个关键方面。虽然其他方法如 CBS-TA 和 ECBS-TA 可以同时考虑任务分配和路径规划，但它们需要显著的计算开销，对于涉及数百个机器人的实例并不适用。据我们所知，除这些传统的、简单的方法外，目前不存在能够在明确解决冲突的同时解决如此大规模 MRTA 问题的方法。因此，包括 Hungarian-TA 和 Greedy-TA 进行比较，以突出我们的方法在这些具有挑战性的场景中的可扩展性和有效性。

在随机场景中，N 从 100 增长到 500，增量为 100。另一方面，在更困难的分离场景中，N 的范围从 50 到 250，间隔为 50。对于每种场景、方法和 N 的组合，测试了 20 个实例。一次性路线图构建在商场、仓库和杂乱环境中分别需要 2.44、17.09 和 19.16 秒。所有比较的方法包括我们的方法都需要路线图。同样，路线图生成时间不包括在计算时间的测量中。

实验结果如图 9 和表 3 所示。平均而言，与 Greedy-TA 和 Hungarian-TA 相比，我们的方法在寻找分配时需要更长的时间。具体而言，在包含 500 个代理的复杂环境中，我们的方法平均需要 8.99 秒，而 Hungarian-TA 仅需 3.79 秒，增加了大约 137 %。在分离的场景中，差异更加明显，在某些情况下，MRTA-RM 比 Hungarian-TA 慢多达 218 %。然而，平均计算时间的差异最多为 5.20 秒，花费几秒钟来减少导航中的巨大延迟对于车队操作是非常有利的，正如以下实验所示。

在动态仿真中，由于缺乏冲突解决机制，Greedy-TA 和 Hungarian-TA 表现出显著的局限性，经常导致死锁。因此，它们的成功率相当低。例如，在随机场景的所有实验环境中，Greedy-TA 的成功率高达 25 %，但在一个实验环境中除外。这种低成功率是由于贪婪方法的本质，其分配的机器人和任务对距离较近，阻挡了其他机器人的路径。然而，在一个购物中心环境中，有 500 个机器人和任务的随机场景下，Greedy-TA 达到 55 % 的成功率，这是因为机器人和任务的密度增加，减少了机器人和任务之间的平均距离，从而减少了机器人相互阻挡路径的次数。Hungarian-TA 方法的成功率比 Greedy-TA 高，在随机场景中高达 85 %，在分开的场景中高达 50 %。这些结果是

Table 4: 基于我们的任务分配以及两种传统的任务分配方法（匈牙利方法和贪心方法），使用 ST-RRT * 在连续空间中进行路径规划的计算时间。表格详细列出了在不同机器人数量（100 和 200）下，在仓库和杂乱环境中进行路径规划所需的计算时间，每个单元格显示均值和标准差值，以均值/标准差表示。

Distribution		Random					
Method		MRTA-RM + ST-RRT *		Hungarian-TA + ST-RRT *		Greedy-TA + ST-RRT *	
# Robots		100	200	100	200	100	200
Comp. time (sec) (mean / sd)	Warehouse	26.38/18.12	76.05/54.08	30.30/12.96	113.68/46.55	53.08/37.40	225.10/58.90
	Clutter	24.00/7.34	45.91/16.70	28.07/8.64	63.35/25.69	46.00/38.48	106.58/64.08
Distribution		Separate					
Method		MRTA-RM + ST-RRT *		Hungarian-TA + ST-RRT *		Greedy-TA + ST-RRT *	
# Robots		100	200	100	200	100	200
Comp. time (sec) (mean / sd)	Warehouse	133.72/17.48	-	213.49/38.54	-	-	-
	Clutter	175.45/17.88	-	198.10/24.24	-	190.29/32.02	-

Table 5: ST-RRT * 结果的性能指标。本表展示了在两种不同的机器人分布（随机和分开）下，使用 ST-RRT * 计算的路径的完工时间和 SoC，跨越两种环境类型（仓库和杂乱）。

Distribution		Random							
Method		MRTA-RM		MRTA-RM + ST-RRT *		Hungarian-TA + ST-RRT *		Greedy-TA + ST-RRT *	
# Robots		100	200	100	200	100	200	100	200
Success rate (%)	Warehouse	95	100	95	50	80	15	90	25
	Clutter	95	100	95	95	60	0	85	70
Makespan (sec)	Warehouse	6.43	5.19	14.95	16.24	21.30	15.02	86.15	147.05
	Clutter	4.83	3.66	9.97	9.07	10.36	9.36	55.34	40.40
SoC (sec)	Warehouse	175.60	232.69	288.12	420.46	316.07	449.89	496.97	956.38
	Clutter	127.16	183.94	158.54	210.87	171.43	225.79	283.11	355.66
Distribution		Separate							
Method		MRTA-RM		MRTA-RM + ST-RRT *		Hungarian-TA + ST-RRT *		Greedy-TA + ST-RRT *	
# Robots		100	200	100	200	100	200	100	200
Success rate (%)	Warehouse	90	75	70	0	50	0	0	0
	Clutter	95	65	90	0	60	0	65	0
Makespan (sec)	Warehouse	14.67	14.89	55.34	-	140.25	-	-	-
	Clutter	9.49	12.74	36.93	-	49.48	-	109.21	-
SoC (sec)	Warehouse	1064.30	2166.83	1805.00	-	2422.61	-	-	-
	Clutter	607.21	1513.74	959.10	-	1165.14	-	1321.85	-

因为 Hungarian-TA 像我们提出的 MRTA-RM 一样，满足 Sec. 3 的性质 1，这避免了机器人之间的正碰。另一方面，MRTA-RM 在随机场景中表现出至少 65 % 的更高成功率，最高可达 100 %，在分开的场景中至少 60 %，最高可达 100 %。特别是在一个有超过 100 个机器人和任务分别定位的仓库环境中，MRTA-RM 表现出至少 75 % 的成功率，而 Greedy-TA 和 Hungarian-TA 方法甚至一次都没有成功过。

对于所有成功的实例，MRTA-RM 显示出比 Hungarian-TA 和 Greedy-TA 更小的完工时间，相对于 Greedy-TA 有高达 92 % 的改善，相对于 Hungarian-TA 有高达 70 % 的改善。这些结果也表明，MRTA-RM 在总时间方面具有优势，总时间是计算时间和完工时间之和，相比传统方法更具优势。

在具有至少两个成功实例（用于计算统计数据）的环境中，Hungarian-TA 在仓库场景中测得平均 SoC 比 MRTA-RM 低 5 %，并且在独立的机器人和任务情境中也是如此。然而，在所有其他情况下，MRTA-RM 的 SoC 都比其他对比方法低，特别是至少比 Greedy-TA 低 11 %，比 Hungarian-TA 低至少 4 %。

MRTA-RM 相比于其他方法能实现较低的完工时间和 Soc 的原因在于其实施了一种重新分配机器人的策略来平衡组件供应。具体来说，MRTA-RM 利用来自组件之间的机器人，这些组件处于供过于求和供不应求之间，从而有效地重新分配它们。

总体而言，我们提出的方法表现出高度的可扩展性，能够高效计算数百个机器人的有效解决方案，同时减少死锁发生的可能性，而不依赖于 MAPF 求解器。

5.3. 与 MAPF 求解器的集成

在本实验中，我们评估了在第 5.2 节中讨论的任务分配方法，并使用一个 MAPF 求解器来比较任务分配的执行结果。为了在连续空间中进行大规模实验，我们使用 ST-RRT * [17] 。ST-RRT * 是由 [18] 提供并用 C++17 实现的。将 ST-RRT * 与任务分配方法相结合的实验结果详细列于表 4 和 5 ，这些表格展示了不同机器分布和环境下的计算时间、成功率、完成时间和通信代价。我们为 ST-RRT * 算法设置了 5 分钟的时间限制以寻找解决方案，这与以前的实验一致。

在涉及 200 个机器人的随机场景中，MRTA-RM + ST-RRT * 方法在复杂环境中保持 95 % 的成功率，而 Hungarian-TA + ST-RRT * 方法的成功率为 0 %，Greedy-TA + ST-RRT * 方法的成功率为 70 %。

在分离场景中，MRTA-RM + ST-RRT * 方法始终优于 Hungarian-TA + ST-RRT * 和 Greedy-TA + ST-RRT * 方法。在拥有 100 个机器人和任务的仓库环境中，MRTA-RM 达到了 70 % 的成功率，远高于 Hungarian 方法的 50 % 的成功率，并且显著优于 Greedy 方法实现的 0 % 。在同样数量机器人的复杂环境中，MRTA-RM 保持了 90 % 的成功率，而 Hungarian 方法达到了 60 % ，Greedy 方法则为 65 % 。在有 200 个机器人的情况下，ST-RRT * 方法无法计算每个实例。

在使用基于不同任务分配策略的 ST-RRT * 算法进行路径规划的计算时间比较中，我们提出的 MRTA-RM 方法始终表现出优越的性能。具体来说，该方法的计算时间比匈牙利方法快至少 11.43 %，比贪心方法快 7.80 %。这种计算时间的大幅减少突显了 MRTA-RM 方法在较快处理任务方面相比于传统方法的效率。

在 MAPF 解决方案的质量方面，MRTA-RM 任务分配策略在所有情况下显示出比其他任务分配方法更高的成功率。虽然匈牙利方法在随机分布的 200 个机器人和任务的仓库环境中比 MRTA-RM 实现了 8.14 % 更好的完工时间，但 MRTA-RM 在其他所有情况下至少超过匈牙利和其他传统任务分配策略 3.10 %，最多达 88.95 %。此外，

MRTA-RM 的 SoC 测量在所有场景中始终至少低 6.54 %。

请注意，如 [19] 中所述，在我们的场景中实现 ST-RRT * 可能会由于在无界状态空间中采样所产生的挑战而增加完工时间和 SoC。这可能导致需要更密集的样本生成，随后使完工时间和 SoC 的数值更高。

这些结果突出了我们提出的 MRTA-RM 任务分配策略的效率和效能，尤其是在处理复杂场景时，能够提高计算速度并提供更高质量的解决方案。

在本文中，我们提出了一种任务分配方法，该方法在机器人执行分配任务时考虑机器人之间的冲突。通过基于路线图的重新分配策略和考虑冲突的任务分配过程，我们在环境的不同组件之间实现了机器人的供需平衡，从而让机器人导航和任务执行时的冲突和死锁减少。我们的动态模拟实验表明，即使在具有挑战性的环境中有数百个机器人，MRTA-RM 也是有效的。未来，我们将通过实现一个使机器人能够沿路线图路径行走的控制器来达到 100 % 的成功率。

此外，为了将我们的方法扩展到具有不同规模或能力的异构机器人团队，我们计划研究针对每种机器人类型生成不同的基于 GVD 的路线图。

6.

致谢 本工作得到了韩国政府（MOTIE）资助的韩国工业技术计划与评估研究所（KEIT）资助（编号 RS-2024-00444344）以及韩国政府（MSIT）资助的韩国国家研究基金会（NRF）资助（编号 2022R1C1C1008476）。

References

[1] B. P. Gerkey, M. J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, The International Journal of Robotics Research 23 (9) (2004) 939–954.

[2] J. K. Lenstra, D. B. Shmoys, É. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical Programming 46 (1990) 259–271.

[3] G. Sharon, R. Stern, A. Felner, N. R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artificial Intelligence 219 (2015) 40–66.

[4] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, N. Ayanian, Conflict-based search with optimal task assignment, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), IFAAMAS, 2018, pp. 757–765.

[5] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, P. J. Stuckey, Integrated task assignment and path planning for capacitated multi-agent pickup and delivery, IEEE Robotics and Automation Letters 6 (3) (2021) 5816–5823.

[6] Z. Liu, H. Wei, H. Wang, H. Li, H. Wang, Integrated task allocation and path coordination for large-scale robot networks with uncertainties, IEEE Transactions on Automation Science and Engineering 19 (4) (2021) 2750–2761.

[7] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1-2) (1955) 83–97.

[8] C. Henkel, J. Abbenseth, M. Toussaint, An optimal algorithm to solve the combined task allocation and path finding problem, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2019, pp. 4140–4146.

[9] H. Zhang, H. Luo, Z. Wang, Y. Liu, Y. Liu, Multi-robot cooperative task allocation with definite path-conflict-free handling, IEEE Access 7 (2019) 138495–138511.

[10] M. Turpin, K. Mohta, N. Michael, V. Kumar, Goal assignment and trajectory planning for large teams of interchangeable robots, Autonomous Robots 37 (4) (2014) 401–415.

[11] R. E. Burkard, F. Rendl, Lexicographic bottleneck problems, Operations Research Letters 10 (5) (1991) 303–308.

[12] D. Chakrabarty, S. Khanna, S. Li, On (1,ϵ)-restricted assignment makespan minimization, in: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2014, pp. 1087–1101.

[13] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.

[14] H. Choset, Incremental construction of the generalized voronoi diagram, the generalized voronoi graph, and the hierarchical generalized voronoi graph, in: Proceedings of the CGC Workshop on Computational Geometry, 1997.

[15] J. Van den Berg, M. Lin, D. Manocha, Reciprocal velocity obstacles for real-time multi-agent navigation, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2008, pp. 1928–1935.

[16] C. Yu, Q. Li, S. Gao, A. Prorok, Accelerating multi-agent planning using graph transformers with bounded suboptimality, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2023, pp. 3432–3439.

[17] F. Grothe, V. N. Hartmann, A. Orthey, M. Toussaint, St-rrt*: Asymptotically-optimal bidirectional motion planning through space-time, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2022, pp. 3314–3320.

[18] I. A. Sucan, M. Moll, L. E. Kavraki, The open motion planning library, IEEE Robotics & Automation Magazine 19 (4) (2012) 72–82.

[19] A. Orthey, C. Chamzas, L. E. Kavraki, Sampling-based motion planning: A comparative review, Annual Review of Control, Robotics, and Autonomous Systems 7 (2023).