分类号：TP212.9　　学校代码：10363
密　级：公开　　学　　号：2220130101

安徽工程大学
Anhui Polytechnic University

硕士学位论文

题目 动态环境下基于冲突搜索的多智能体路径规划算法研究及应用

论文作者　邢曦辰
指导教师　汪步云　教授
学科（专业）　人工智能
研究方向　智能机器人

论文提交日期：2025 年 05 月 20 日

分类号：TP212.9　　单位代码：10363

密　级：公开　　学号：2220130101

动态环境下基于冲突搜索的多智能体路径规划算法研究及应用

RESEARCH AND APPLICATION OF MULTI-AGENT PATH PLANNING ALGORITHM BASED ON CONFLICT SEARCH IN DYNAMIC ENVIRONMENT

学生姓名：邢曦辰

指导教师：汪步云(教授)

专　　业：人工智能

研究方向：智能机器人

论文答辩日期：2025 年 5 月 9 日

安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名： 

日期： 2025 年 5 月 20 日

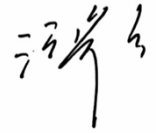
安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保密 □，在 ____ 年解密后适用本版权书。

不保密 □。

学位论文作者签名：  　　指导教师签名： 

日期： 2025 年 5 月 20 日　　日期： 2025 年 5 月 20 日

动态环境下基于冲突搜索的多智能体路径规划算法研究及应用

摘 要

多智能体路径规划作为智能调度与自主导航的核心技术，在仓储物流、无人机编队、自动驾驶、工业机器人调度等场景中发挥着重要作用。然而，在动态环境下，传统的冲突搜索算法依赖于全局路径重规划，计算开销较大，难以满足中高动态场景的实时性需求。除此之外，现有的改进方法通常采用被动调整策略，仅在检测到路径冲突后进行修正，缺乏对未知环境的主动探索能力，导致路径规划的全局最优性受到限制。针对上述问题，本文以提升动态环境下的路径规划效率与适应性为目标，在冲突搜索算法框架的基础上，提出增量式路径调整机制和基于探索约束的路径优化机制，并进一步设计融合优化策略，以增强多智能体系统在复杂环境中的调度能力和路径优化效率。具体研究内容如下：

(1) 针对动态环境下全局路径重规划计算冗余高、响应速度慢的问题，本文在 CBS 算法框架基础上引入增量式路径调整机制，提出增量式冲突搜索算法（ICBS），以提升路径规划的计算效率和实时适应性。 ICBS 通过融合多种传感器信息，实时感知环境变化，精准识别受影响的智能体路径，并对其进行路径评估与分类，将路径划分为可复用路段和失效路段。针对失效路段，ICBS 使用其底层搜索算法进行增量式调整，避免全局重计算，降低计算冗余。调整过程中，系统将动态障碍物转换为相应的约束条件，在路径重规划时施加限制，以确保所有智能体路径符合最新环境约束，从而避免新的冲突产生。

最后，本文在不同动态条件及不同规模环境下开展仿真实验，结果表明 ICBS 能够有效减少计算开销，提高多智能体系统的任务响应能力和整体调度稳定性，使其在复杂动态环境下具备更优的路径优化能力和更高的执行效率。

(2) 传统路径规划方法在动态环境下通常不具备对可行性待定区域进行主动确认的能力，虽然这一策略有助于降低计算开销，但在环境变化较大的情况下，可能导致路径规划陷入局部最优，难以充分利用潜在可行区域。为此，本文提出探索增强型 A*搜索算法，在代价函数中引入探索潜力函数及其权重系数，以引导智能体在路径规划过程中对关键的模糊区域进行评估与确认。同时，将该算法作为底层搜索算法融入 CBS 框架，并在冲突搜索树上补充动态探索约束，以确保探索行为不会影响智能体集群的任务执行稳定性，进而构建动态探索型冲突搜索算法（DE-CBS）。DE-CBS 使智能体在路径计算时不仅依赖当前环境信息进行路径规划，还能结合历史探索数据优化未来路径方案，从而提升路径规划的全局最优性和环境适应性。实验结果表明，该方法通过迭代优化能够有效降低路径冲突率，提高路径质量，使多智能体系统在复杂动态环境中具备更强的适应性和鲁棒性。

(3) 在上述两项工作的基础上，本文融合增量式路径调整机制与动态探索路径优化策略，以兼顾计算效率与环境适应性。在环境变化较小时，优先进行增量式路径调整，对受影响路段进行局部优化，以减少计算负担并提升路径调整的实时性。当路径存在不确定性或环境结构复杂时，算法自适应引入动态探索路径优化策略，主动评估可行性待定区域，扩展路径搜索范围，以优化路径方案并避免局部最优。最终，在大规模 DAO 地图的仿真实验及实验室搭建的仓储环境中，对融合算法的计算效率与环境适应性进行了评估，实验结果表明，该

方法能够有效降低计算开销，提升路径规划的全局优化能力，使多智能体系统在复杂动态环境下能够稳定、高效地执行任务。

关键词：多智能体路径规划，冲突搜索算法，A*算法，动态环境，路径优化

RESEARCH AND APPLICATION OF MULTI-AGENT PATH PLANNING ALGORITHM BASED ON CONFLICT SEARCH IN DYNAMIC ENVIRONMENT

ABSTRACT

Multi-agent path planning, as a core technology in intelligent scheduling and autonomous navigation, plays a crucial role in scenarios such as warehouse logistics, drone formations, autonomous driving, and industrial robot coordination. However, in dynamic environments, traditional conflict-based search (CBS) algorithms rely on global path replanning, which incurs significant computational overhead and struggles to meet real-time requirements in highly dynamic scenarios. Moreover, existing improved methods predominantly adopt passive adjustment strategies, correcting path conflicts only after detection. This lack of proactive exploration in unknown environments limits the global optimality of path planning.

To address these challenges, this study aims to enhance the efficiency and adaptability of path planning in dynamic environments. Based on the CBS framework, we propose an incremental path adjustment mechanism and an exploration-constrained path optimization mechanism, further integrating an optimization strategy to improve multi-agent system scheduling capabilities and path planning efficiency in complex environments. The key research contributions are as follows:

To address the excessive computational redundancy and slow response time of global path replanning in dynamic environments, this study introduces an incremental path adjustment mechanism into the CBS framework, proposing the Incremental Conflict-Based Search (ICBS) algorithm to improve computational efficiency and real-time adaptability. ICBS integrates multi-sensor information to perceive environmental changes in real time, accurately identifies affected agent paths, and classifies them into reusable and invalid segments. For invalid segments, ICBS employs its underlying search algorithm for incremental adjustments, avoiding global recomputation and reducing computational redundancy. During the adjustment process, dynamic obstacles are transformed into corresponding constraints, ensuring that all agent paths comply with the latest environmental restrictions and preventing new conflicts from arising. Finally, simulation experiments under various dynamic conditions and environment scales demonstrate that ICBS effectively reduces computational overhead, enhances task response capabilities and scheduling stability in multi-agent systems, and improves path optimization and execution efficiency in complex dynamic environments.

Traditional path planning methods in dynamic environments typically lack the ability to actively assess feasibility in uncertain regions. While this strategy reduces computational overhead, it may lead to local optima and suboptimal path utilization in highly dynamic environments. To address this, we propose an Exploration-Enhanced A* search algorithm, introducing an exploration potential function and corresponding weight coefficients into the cost function to guide agents in evaluating and confirming critical ambiguous regions during path planning. This algorithm is then integrated into the CBS framework as an underlying search method, supplemented with dynamic exploration constraints in the

conflict search tree to ensure that exploration behaviors do not compromise task execution stability. The resulting Dynamic Exploration Conflict-Based Search (DE-CBS) algorithm enables agents to leverage both current environmental information and historical exploration data to optimize future path plans, thereby improving the global optimality and environmental adaptability of path planning. Experimental results demonstrate that DE-CBS effectively reduces path conflicts and improves path quality through iterative optimization, equipping multi-agent systems with enhanced adaptability and robustness in complex dynamic environments.

Building on the above two contributions, this study integrates incremental path adjustment and dynamic exploration optimization to balance computational efficiency with environmental adaptability. When environmental changes are minor, the algorithm prioritizes incremental path adjustments, locally optimizing affected segments to reduce computational burden and enhance real-time path modification. In scenarios involving path uncertainty or complex environmental structures, the algorithm adaptively incorporates dynamic exploration optimization to actively evaluate uncertain feasible regions, expanding the search space to optimize path plans and avoid local optima. Finally, large-scale DAO map simulations and warehouse environment experiments validate the computational efficiency and adaptability of the integrated algorithm. Experimental results confirm that this approach effectively reduces computational overhead, enhances the global optimization capability of path planning, and enables multi-agent systems to perform tasks stably and efficiently in complex dynamic environments.

Xing Xichen (Artificial Intelligence)
Supervised by Wang Buyun

KEY WORDS: Multi-Agent Path Finding, Conflict-Based Search, A* Algorithm, Dynamic Environments, Path Optimization

目录

图目录

表目录

第1章 绪论

1.1 研究背景及意义

在人工智能和自动化技术飞速发展的背景下，多智能体系统（Multi-Agent System, MAS）逐渐成为解决复杂任务的一种关键技术。MAS 涉及多个智能体（agents），这些智能体可以相互协作或竞争，完成诸如机器人导航[1] 、物流调度、仓储管理[2] [3] 、无人机编队、自动驾驶车队管理[4] [5] 等任务。每个智能体通常具有一定的自主性和智能，能够感知环境、做出决策并执行相应的动作。在这些应用中，多智能体路径规划（Multi-Agent Path Finding, MAPF）问题成为了至关重要的研究课题，其目标在于在给定的环境中为每个智能体规划一条从起点到达目标的路径，确保在整个规划过程中智能体之间不会发生冲突[6] [7] 。冲突通常指两个或多个智能体同时占据同一个节点（位置）或同时通过同一条边（路径），这些冲突会导致任务的失败或效率的降低。因此，避免冲突的路径规划在 MAS 中至关重要。

MAPF 问题本质上是一个典型的组合优化问题，并且已被证明是 NP 难问题[8] 。也就是说，随着问题规模的增加（即智能体数量和环境复杂度的增加），计算所需的时间和资源将呈指数级增长。这给路径规划带来了巨大的挑战，特别是在动态环境中，问题的复杂性和不确定性进一步加剧[9] [10] 。例如，当环境中的障碍物或其他因素发生变化时，系统必须实时更新路径规划，这对算法的响应速度和计算效率提出了更高要求。再例如，激光雷达（LIDAR）作为一种常见的环境感知工具，虽然能够提供实时的环境信息，但在动态环境中，激光雷达的探测数据容易受到干扰，导致构建的地图出现误差和伪影[11] 。这种不确定性不仅影响智能体的定位精度，还可能导致路径规划出现偏差，从而增加冲突发生的风险。在这种背景下，如何提高路径规划算法在动态环境中的鲁棒性和实时性，成为了研究者关注的重点。

随着技术的进步，MAPF 问题的应用场景日益复杂化。在现代自动化仓储系统中，如 Amazon 的 Kiva 机器人系统，依赖数百台机器人在仓库内移动物品[12] [13] [14] 。机器人需要在狭小且拥挤的仓库环境中高效移动，尽量避免相互干扰。在这种场景中，机器人不仅需要避开固定的障碍物（如货架），还需要实时应对

其他移动机器人带来的潜在冲突。这种环境的动态性和复杂性给路径规划算法提出了极高的要求。

图 1-1 Kiva 仓库作业场景图

MAPF 问题的核心挑战是冲突检测与解决，例如在无人机编队飞行任务中，多个无人机需要在共享的空域中保持队形飞行，同时避免相互之间以及外界障碍物的碰撞[15] 。尤其是在复杂地形中，如森林或城市环境，环境中的障碍物可能会干扰激光雷达的探测，导致路径规划出现误差。此外，突发的天气变化也会增加飞行路径规划的难度。这些因素使得实时路径规划变得尤为重要。例如，在无人机表演或军事编队飞行中，多个无人机需要精准规划路径，以防止发生空中碰撞。在这种动态空域中，环境的不确定性和无人机之间的复杂交互，进一步增加了路径规划的难度。

图 1-2 无人机群在森林中协作飞行示意图[15]

同样，自动驾驶技术的发展使得车队行驶管理成为一个新兴的研究领域。在车队行驶中，每辆车都需要根据前方道路状况、其他车辆的位置和速度，实时调

整行驶路径[16][17]。车队车辆之间的相对速度和距离必须严格控制，否则可能引发严重的连环碰撞事故。此外，道路上的动态障碍物（如其他车辆、行人）和天气状况的变化也会对路径规划算法提出挑战。

传统的路径规划算法，如 A*算法和 Dijkstra 算法，尽管在单智能体路径规划中表现优异，但在处理 MAPF 问题时，尤其是在多智能体高密度、动态变化的环境中，表现出明显的不足。具体表现为计算复杂度过高、无法有效处理大规模智能体的路径冲突、对动态变化的适应性差等问题。因此，如何在保障路径规划的最优性与无冲突性的同时，提升算法的计算效率和实时响应能力，已成为当前研究的重要课题。

冲突搜索算法（Conflict-Based Search, CBS）是近年来在 MAPF 领域备受关注的路径规划方法之一[18][19]。CBS 算法以其独特的二层搜索结构为核心，显著提升了在处理多智能体路径冲突问题时的效率。CBS 通过在高层次上构建一个约束树（Constraint Tree, CT），在低层次上为每个智能体进行单独的 A*搜索，从而将复杂的多智能体问题分解为多个相对简单的子问题进行处理[20]。这种方法不仅在理论上具有良好的性能保证，而且在实际应用中也展示了强大的适应性[19]。然而，尽管 CBS 算法在静态环境中表现优异，但其在动态环境中的应用仍面临诸多挑战。尤其是在环境发生变化时，CBS 通常需要重新进行全局路径规划，这不仅增加了计算开销，还可能导致算法效率的显著下降。

动态环境下的路径规划问题具有更高的复杂性和不确定性[21]。例如，在自动化仓储系统中，货架可能随时移动，机器人需要即时调整路径；在无人机飞行过程中，突发的天气变化或其他障碍物可能导致原有路径不再适用，需实时规划新的路径。这些场景要求路径规划算法不仅能够快速响应环境的变化，还要能够在大规模、多目标的复杂环境中高效地处理路径冲突。因此，提升 CBS 算法在动态环境下的性能，特别是在实时决策和多目标优化中的表现，具有重要的研究意义和应用价值。

本研究的主要意义在于对冲突搜索算法（CBS）进行针对性的改进，以提升其在动态环境下的多智能体路径规划效率与稳定性。首先，本研究通过设计增量式冲突搜索算法（ICBS），在环境发生变化时不再需要进行全局路径重新规划，而是通过增量更新的方式减少计算开销，从而显著提高算法的响应速度。其次，

本文提出了一种动态探索型 CBS 算法（DE-CBS），该算法在处理复杂动态环境时，能够有效优化多智能体之间的冲突解决过程，进一步提升路径规划的效率和准确性。通过在典型的动态环境中进行实验验证，本研究的改进算法在路径长度、计算时间等性能指标上表现出了优于传统 CBS 算法的效果，特别是在大规模智能体系统中的扩展性得到了显著提升。研究结果表明，本研究所提出的改进方法能够在动态、多目标优化的复杂场景中实现更加高效、稳定的路径规划。

1.2 国内外研究现状

1.2.1 分布式多车路径规划问题研究现状

分布式多车路径规划是一种无中央控制器的路径规划方法，智能体（如机器人、无人机等）在规划过程中独立决策自己的路径，依赖于自身或周围局部信息来避免与其他智能体的冲突。这意味着每个智能体只能访问其有限的感知或通信范围内的信息，而没有全局环境的全貌。分布式路径规划中的智能体通常通过通信或局部感知与邻近智能体进行协调，以避免冲突并优化路径。

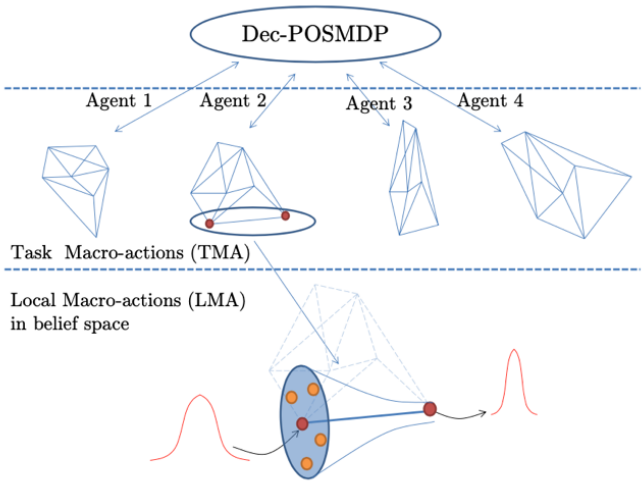


图 1-3 Dec-POSMDP 算法的层级结构[22]

针对分布式多车路径规划问题的部分可观测性、异步决策和通讯受限等特点，马尔可夫决策过程（Markov Decision Process, MDP）能够很好的适用于解决该类问题。其核心思想是系统的未来状态只依赖于当前状态和当前的动作，而与之前的状态和动作无关，然后通过动态规划或强化学习等方法，找到最优策略，使得系统在每个状态下采取的动作能够最大化累积奖励。Shayegan 等人提出了一种

基于去中心化部分可观测半马尔可夫决策过程（Decentralized control of partially observable Markov, Dec-POSMDP）的框架，通过引入宏观动作（Macro-actions），实现异步决策和部分可观测环境中的多机器人协作[22] 。

宏观动作将连续空间的复杂问题简化为一系列离散的高层次任务决策，使得每个机器人可以独立选择并执行这些任务，避免了同步决策的需求。文章还设计了基于动态规划和蒙特卡罗搜索的算法，用于在构建的宏观动作图中高效规划各个机器人的策略，并计算每个动作的成功概率、完成时间和价值。Aurelie 等人关注时间和资源约束，引入机会成本的概念，确保机器人能够在不通信的情况下，通过局部的 MDP 计算协调各自的任务，避免因任务执行延迟影响其他机器人的任务进度[23] 。Christopher 总结了动态规划、启发式搜索、近似算法等几类针对 Dec-POMDP 的求解方法，解决了在多个智能体之间进行分布式决策时的复杂性问题，特别适用于通信受限的去中心化场景[24] 。Han 提出了一种称为 IPOMDP-Net 的神经网络架构[25] 。IPOMDP-Net 基于交互式部分可观测马尔可夫决策过程（Interactive POMDP, I-POMDP），结合了 QMDP 规划算法，并将其嵌入到一个可微分的递归神经网络（RNN）中，用以解决 MAPF 问题。

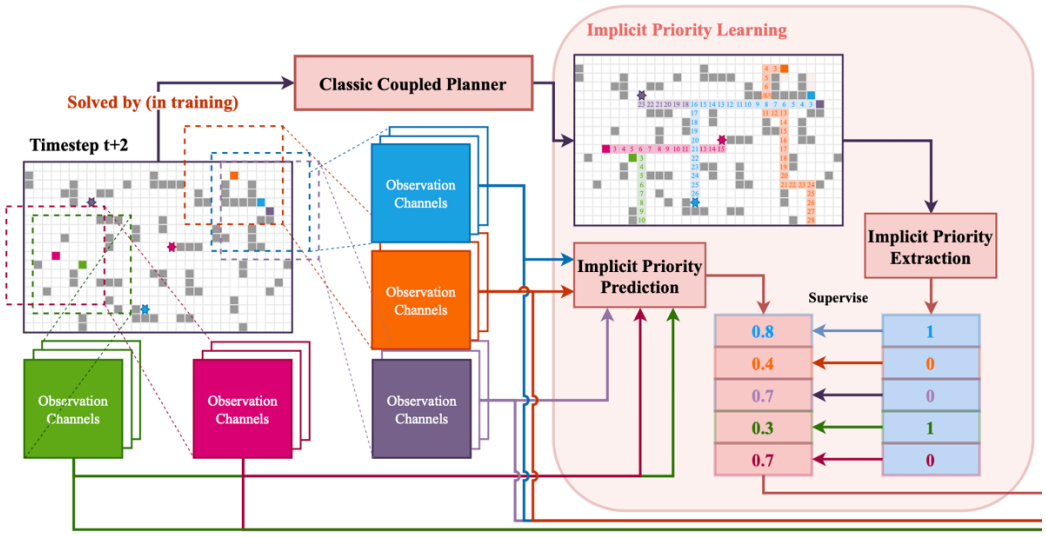


图 1-4 文献[27] 中 PICO 算法概述

优先级规则在分布式的多智能体路径规划问题中是一个非常常见的策略。例如，有研究学者为了优化多智能体路径规划中的优先级排序，提出了两种基于机器学习的模型 ML-T 和 ML-P。其中 ML-T 模型通过学习全局优先级排序，而 ML-P 模型学习部分优先级排序，除此之外还引入了随机排序和随机重启的策略，以

提升解决问题的成功率和解决质量[26]。类似的也有学者提出了一种名为 PICO（PrIoritized COmmunication learning）的方法，通过引入通信增强的多智能体强化学习框架，在实现去中心化路径规划的同时，利用隐式优先级来动态调整通信拓扑，避免碰撞[27]。该方法分两个阶段实现：隐式优先级学习阶段模仿经典的耦合规划器来预测局部优先级，而优先级通信学习阶段基于这些优先级构建时间变化的通信拓扑，并通过权重调整实现通信集群的动态更新。优先级规划方法具有实现简单、计算成本低的优点，特别是在分布式系统中，能够有效避免集中式方法带来的高计算开销。然而，随着智能体数量和环境复杂度的增加，优先级规划方法在大规模、高密度、多变的动态环境中表现逐渐受限，容易产生次优解。

近年来，机器学习作为一种数据驱动的方法，也在逐渐被应用于分布式多车路径规划问题中[28][29][30][31]。如强化学习（Reinforcement Learning）中的多智能体深度强化学习（Multi-Agent Deep Reinforcement Learning, MADRL），智能体可以通过学习环境及其他智能体的行为，逐渐自我调整以优化路径选择。在分布式系统中，多智能体之间可能是合作关系也可能是竞争关系，如何设计奖励机制使得智能体能够协调行动、避免冲突，成为了 MARL 中的关键挑战之一。有学者就引入了策略均衡概念，如纳什均衡，将博弈论与强化学习结合，以便智能体在共享环境中能有效协调和竞争[32]。具体而言，该文提出了一种框架，允许多个智能体在共享环境中同时学习并适应其他智能体的策略变化。在这个框架中，智能体既可以独立 Q 学习（基于自身观察的奖励进行学习），也可以使用联合动作学习（基于观察到的其他智能体的动作和奖励进行学习）、纳什 Q 学习和稀疏协同 Q 学习等，解决了多智能体环境中策略依赖、动态变化等问题。

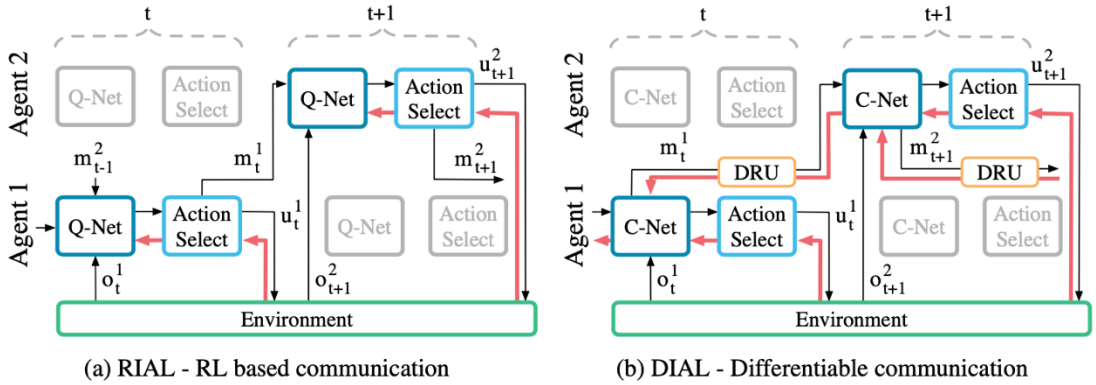


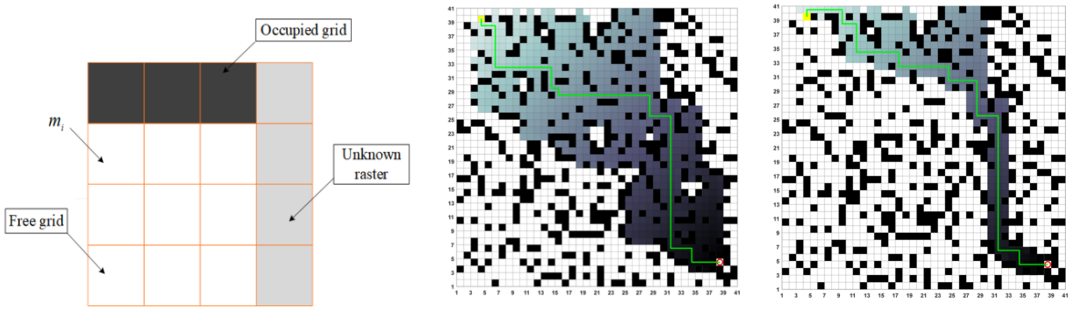
图 1-5 文献[35] 中 RIAL 和 DIAL 方法的工作流程

Deng 等人针对机器人在危险或不可接近的环境中代替或协助人类进行任务的需求，将问题建模为一个马尔可夫决策过程（MDP），提出了一种多机器人协同探索方法[33] 。在该方法中，机器人仅基于其局部观测进行自主决策，并通过多智能体深度确定性策略梯度（Multi-Agent Deep Deterministic Policy Gradient, MADDPG）算法来训练机器人。在此过程中，利用集中训练、分布式执行的架构，提升了多个机器人在协作探索中的表现。使得多个机器人可以在未知环境中自主探索并协同构建地图的同时，有效减少探索中的重复和冗余，提升探索效率。类似的 Saifullah 等人基于 MADRL，结合集中训练与分布式执行架构，解决了交通基础设施管理中的大规模优化问题[34] 。Jakob 等人提出了强化智能体间学习（RIAL）和可微智能体间学习（DIAL）两种方法，用于多智能体在部分可观测环境下学习通信协议[35] 。DIAL 通过允许智能体间在学习阶段传递梯度，提升了通信学习的效率和表现，使得智能体可以在有限的通信带宽下进行更有效的协作。

1.2.2 集中式多车路径规划问题研究现状

在集中式多车路径规划中，所有智能体的路径规划由一个中央控制器或全局控制器统一协调和管理。控制器可以获取整个系统的全局信息，包括所有智能体的状态（位置、速度等）以及整个环境的完整信息，然后根据这些信息规划出所有智能体的路径。每个智能体的路径由中央控制器独立规划和分配，保证全局的最优性或近似最优性。

基于图搜索的路径规划方式，是早期研究和解决 MAPF 问题的基础技术之一。其中最具代表性的图搜索算法，例如 A*算法[36] ，它在单智能体路径规划问题中表现出色，但在多智能体路径规划中遇到了计算复杂度的挑战。为了适应 MAPF 问题，诸多研究人员对 A*算法进行了各种改进。例如，Han 等人首先在 A*算法的启发式函数中加入了自适应指数系数，以动态调整启发式函数的权重，然后将改进后的 A*算法与动态窗口算法相结合[37] 。改进的 A*算法用于全局路径规划，DWA*算法则负责局部动态障碍物的规避。当局部路径偏离全局方向时，DWA*算算法会触发全局路径的动态重新规划。



(a) DWA*网格利用率示意图　　(b) A*规划效果　　(c) DWA*规划效果

图 1-6 DWA*与 A*算法规划效果对比

Lucas 等人在 Transformer 框架下训练模仿 A*算法的搜索过程，提出了一种名为 Searchformer 的模型[38] 。该模型将 A*算法的搜索过程表示为标记序列进行训练，并通过专家迭代进一步优化，在减少搜索步骤的同时生成最优解，实现对 A*算法的改进。Cao 提出了一种混合 A*算法，将双向搜索与跳点搜索（JPS）算法相结合，引入安全权重矩阵，优化 A*算法在路径搜索中的效率与安全性[39] 。Grenouilleau 提出了多标签 A*算法（MLA*），在一次搜索中同时考虑拾取和交付目标以减少不必要的约束[40] ，并结合基于 h 值的启发式任务分配方法，通过逐步计算智能体和任务之间的 h 值来分配任务。然而，A*算法在处理多智能体路径规划时存在显著的局限性，主要体现在计算复杂度高和冲突处理能力不足。随着智能体数量的增加，A*算法的计算复杂度呈指数级增长，难以在大规模 MAPF 问题中保持高效[41] 。此外，A*算法无法有效处理智能体之间的冲突，容易导致路径规划失败或产生次优解。

冲突搜索算法是由 Sharon 等人在 2015 年提出的一种专门用于 MAPF 问题的集中式方法[42] 。该算法采用了双层搜索结构，在高层次构建了一个冲突树，每个节点表示一组针对智能体运动的约束条件。如果智能体之间发生冲突（如两个智能体在同一时间占据同一位置），算法会在冲突树上进行分叉，添加新的约束以消除冲突。低层次则是针对每个智能体进行单智能体路径搜索，确保其运动满足高层次给出的约束。以这种方式，通过分离冲突和路径规划的过程，CBS 能够减少状态空间的搜索量，逐步逼近最优解。Wen 等人在冲突搜索算法的基础上对其进行改进，提出了带有运动学和时空约束的类车冲突搜索算法[43] 。该算法针对类车智能体无法原地转动、具有最小转弯半径且外表通常为矩形等特点，设立了考虑智能体之间因外形导致的形体冲突。在低层次上，使用该文提出的一种

新的“时空混合状态 A*”算法，作为单智能体路径规划器，来生成符合运动学和时空约束的路径。在高层次上，在原本的冲突树上引入智能体之间的形体冲突，确保它们在整个路径上不会发生碰撞。

Barer 等人为了提高运行效率，放宽了 CBS 算法的最优性条件，提出了几种无界次优变体和有界次优变体[44] 。例如，贪心 CBS 使用多种冲突启发式来减少冲突节点的扩展，通过对 CBS 的高层和低层搜索进行简化，优先处理那些似乎更接近目标节点的节点，从而在不追求最优性的前提下快速找到解；有界 CBS 通过引入焦点列表机制，确保返回的解在一定的次优性范围内；增强 CBS 通过共享高层和低层的次优性约束来提高效率，确保最终解的代价在给定的范围内。在 ECBS 的基础上，Li 提出了 EECBS（Explicit Estimation CBS）算法，它使用了一种在线学习技术来估算每个高层节点的解的成本，结合显式估算搜索（EES）策略来选择扩展的节点，优化 CBS 算法的次优解搜索能力[45] 。还有允许智能体以不同的速度移动，并使用多种启发式方法优化运行效率的无限速度冲突搜索（Infinite Speed CBS, IS-CBS）算法[46] ；用于解决多目标 MAPF 问题的多目标冲突搜索（Multi-objective CBS, MO-CBS）算法[47] ；引入动态障碍预判机制以及灵活的等待和绕行策略，应对未知动态环境的 CBS-D*（Conflict-Based Search with D* Lite）算法[48] ；以及将 MAPF 问题分解为更小的子问题，使用并行化和动态调度等方式，优化求解速度和内存效率的动态并行层次化组合冲突搜索（DPHC-CBS）算法[49] 等等。

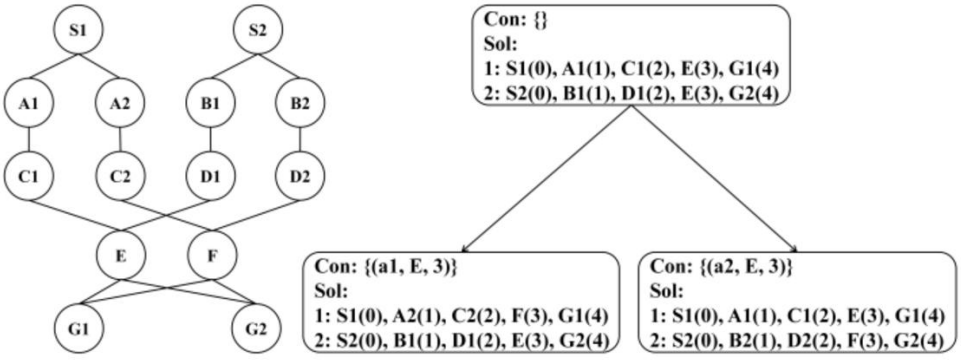


图 1-7 IS-CBS 算法处理不同类型冲突

MAPF 问题还可以转化为一个整数线性规划（Integer Linear Programming, ILP）问题，通过定义目标函数和约束条件来构建全局最优解的规划问题。例如 Yu 等人将 ILP 和启发式技术相结合，把 MAPF 问题转化为一种特殊的多流网络问题

（Multi-flow Network Problem）[50]。每一个智能体相当于网络中的一个商品，从源点移动到目的点的同时，使用 ILP 求解器最小化优化目标。Wang 研究了带有任务截止时间的 MAPF 问题，除了新增任务的截止时间还要求在避免机器人碰撞的前提下最小化三个不同的目标：最大延迟时间、总拖延时间以及总单位罚款[51]。通过构建时间扩展网络，将问题转化为多流网络问题，并使用整数线性规划求解每个时间步的最优解，实现了全局最优的无碰撞路径规划。

常见应用于 MAPF 问题的基于全局优化的方法还有遗传算法（Genetic Algorithm, GA）、粒子群优化算法（Particle Swarm Optimization, PSO）、模拟退火（Simulated Annealing, SA）等。比如 Mao 提出了一种改进的自适应探索模拟退火 Q-Learning 算法（AE-SAQL）[52]，在 SA 中引入 Q-Learning 的策略选择过程，通过动态调整探索率来平衡探索与利用。同时引入一种自适应的温度调整机制，在算法初期，给智能体提供较高的探索率以避免陷入局部最优，随着算法迭代，逐步降低温度，减少随机行为确保智能体最终收敛到全局最优解。Lamini 通过优化遗传算法的迭代优化和适应度评估机制，设计有效的编码、适应度函数、交叉和变异操作，使得遗传算法能够在多约束环境中找到全局最优路径[53]。Neumann 拓展了早期的基于量子玻尔兹曼机（Quantum Boltzmann Machines, QBM）的单智能体研究，使用量子力学的方式模拟受限玻尔兹曼机中的退火过程，优化权重找到状态-动作对的最优组合[54]。Tao 提出改进的粒子群优化（Improved Particle Swarm Optimization, IPSO）算法，模拟鸟群觅食行为[55]。通过设计特殊的整数编码方法、交叉变异操作，避免了连续解空间导致的无效搜索，当算法检测到全局最优值在多次迭代中没有更新时，使其能够跳出局部最优。

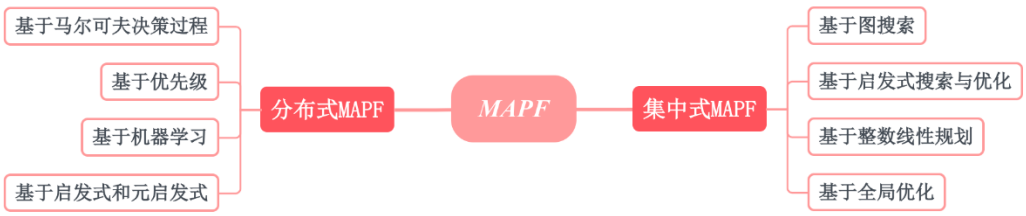


图 1-8 MAPF 问题的研究方法分类

综上当前主流的多智能体路径规划方法多集中于静态环境或全局重计算策略，面对动态任务调度、高频障碍变化等应用需求，存在路径反复重算、规划延

迟高、冲突解决滞后等问题。为此，本文选用集中式图搜索算法，在传统 CBS 算法基础上，提出增量式路径调整机制与主动探索优化策略，构建适应复杂动态环境的路径规划算法框架，以期为智能仓储、机器人调度等场景中的智能设备提供高效、稳定的路径决策支持，进一步提升其环境适应性与调度效率。

1.3 本文结构安排

本文围绕动态环境下多智能体路径规划的关键问题，提出基于冲突搜索的优化算法，并通过理论分析、仿真实验及实际应用验证算法的有效性。论文的章节安排如下：

第 1 章绪论：介绍研究背景及意义，综述国内外多智能体路径规划的研究现状，并概述本文的创新点及结构安排。

第 2 章多智能体路径规划问题的建模与分析：对 MAPF 问题的基本定义、约束条件及优化目标进行建模，并介绍冲突搜索算法的原理、实现及局限性，为后续改进算法提供理论基础。

第 3 章基于动态环境的增量式冲突搜索算法研究：针对 CBS 算法在动态环境中的计算冗余问题，提出增量式路径调整机制，包括环境监测模块、增量更新模块和冲突解构模块，并通过实验评估算法的计算效率和路径质量。

第 4 章动态探索型冲突搜索算法的设计与分析：进一步优化动态环境下的路径调整策略，提出动态探索型冲突搜索算法，结合探索增强型 A*算法和探索依赖图，提升路径优化的全局适应性，并通过实验验证其优越性。

第 5 章动态环境下的算法融合与实验：在前述研究基础上，提出融合增量式路径调整与动态探索优化的混合算法，并在 DAO 标准地图环境与实际仓储场景中进行测试，评估算法的计算效率、任务执行稳定性及应用价值。

第 6 章总结与展望：总结本文的研究成果，分析算法的适用性与局限性，并探讨未来在多智能体路径规划领域的进一步研究方向。

第2章 多智能体路径规划问题的建模与分析

2.1 引言

MAPF 问题是人工智能和多智能体系统中的经典难题。其核心目标是在一个给定的空间中为多个智能体规划各自的无冲突路径，使它们能够从起始位置到达目标位置。由于多个智能体同时移动，路径规划必须避免冲突，确保每个智能体在指定时间和空间内的路径是无冲突的，从而保证系统的整体运行效率。

2.2 多智能体路径规划问题的提出

MAPF 问题通常被建模为一个无向图 $G = (V, E)$和一组包含 k 个智能体数量的智能体集合 $A = \{a_i, a_2, ..., a_k\}$。其中 V 为节点集合，表示智能体可以占据的位置，E 为边集合，表示智能体可移动的路径。对于集合 A 中的每一个智能体 a_i 都有独特的起始节点 $s_i \in V$ 和目标节点 $g_i \in V$。MAPF 问题的目标就是为这 k 个智能体找到一组从起始节点到目标节点互不冲突的路径集合$\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$，并且满足一定的优化目标。为了达成这样的条件，我们一般将 MAPF 问题下的时间离散成时间步，而不是连续的。如此一来，每个智能体 a_i 的路径π_i 就成为了一个节点序列，形如$\pi_i = \{v_i^0, v_i^1, ..., v_i^T\}$。其中 $v_i^t \in V$ 表示在时间步 t 时智能体 a_i 所在的节点，完整路径时间步长度为 T，对应路径的终点位置为 $v_i^T = g_i$。

图 2-1 提供了 MAPF 构建模型的初始流程，如图所示构建模型的第一步是将连续或复杂的环境进行离散化，生成有限的节点集合。对于图中的 5x5 的网格环境，每个网格单元(x,y)都视为一个节点 $v_{x,y}$ 纳入节点集合 $V = \{v_{0,0}, v_{0,1}, \ldots, v_{5,5}\}$。

随后将所有允许在相邻节点之间移动的地方记为一条边 $e = (v_i, v_j) \in E$。例如图 2-1 中的节点 $v_{x,y}$ 与其相邻的节点 $v_{x+1,y}$、$v_{x-1,y}$、$v_{x,y+1}$ 和 $v_{x,y-1}$ 之间各有一条边。

时间被离散化后，MAPF 问题中不同智能体之间的所产生的冲突也就能在离散的时间轴上进行表示了。具体来说冲突主要分为顶点冲突和边冲突这两种路径冲突。

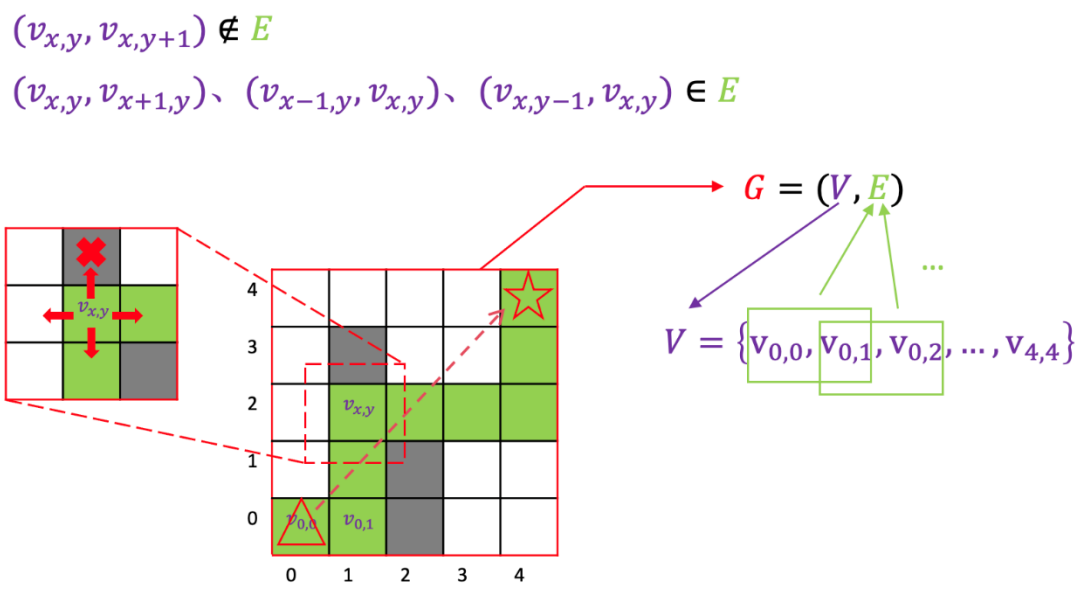


图 2-1 图模型部分构建过程示意图

顶点冲突指的是两个或多个智能体在同一时间步 t 试图占据相同的位置 v，如图 2-2 所示。这种冲突在仓储环境中较为常见，特别是在机器人需要同时通过同一个狭窄通道时。

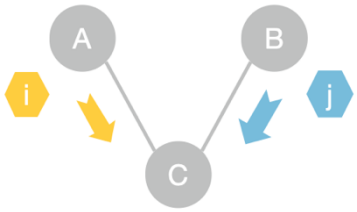


图 2-2 顶点冲突示意图

形如公式 2-1 所示，顶点冲突可以表示为不同智能体 a_i 和 a_j 的路径π_i、π_j 在时间步 t 上出现在了相同的节点位置。

$$\pi_i(t) = \pi_j(t), \quad \forall i \neq j \tag{2-1}$$

另一种冲突边冲突，则分为两种情况：第一种情况是指两个智能体在相邻一个或多个时间步内交换位置，如图 2-3 所示。假设若智能体 a_i 在时间步 t 至 $t+1$ 内从节点 v_i^t 移动到节点 v_i^{t+1}，而智能体 a_j 在同一时间步从节点 v_j^t 移动到节点 v_j^{t+1}，但是 $v_j^t = v_i^{t+1}$ 且 $v_i^t = v_j^{t+1}$。此时两个智能体会因为所需要的节点资源相同而产生冲突导致路径规划方案失效。

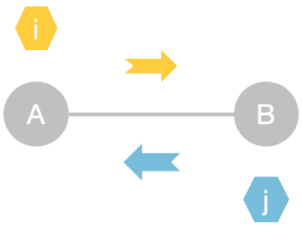


图 2-3 第一种边冲突示意图

边冲突的第二种情况较为常见，指的是两个智能体在相邻一个或多个时间步内向同一方向出发，但需要占用同一条边（如图 2-4 所示）。具体的，假设智能体 a_i 在时间步 t 至 $t+1$ 内从节点 v_1 移动到节点 v_2，而智能体 a_j 在同一时间步内同样需要从节点 v_1 移动到节点 v_2。

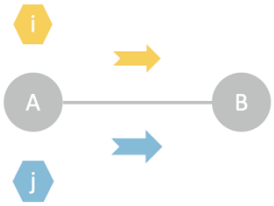


图 2-4 第二种边冲突示意图

第一种边冲突可以使用公式 2-2 来表示，第二种边冲突则如公式 2-3 所示。

$$\pi_i(t) = \pi_j(t+1) \quad \text{and} \quad \pi_j(t) = \pi_i(t+1) \tag{2-2}$$

$$\pi_i(t) = \pi_j(t) \quad \text{and} \quad \pi_j(t+1) = \pi_i(t+1) \tag{2-3}$$

如前文所述，MAPF 问题的目标是在一个给定的空间中为多个智能体规划各自的无冲突路径，并且满足一定的优化目标。不同解决 MAPF 问题的算法有不同的应对冲突的策略，但对于优化目标，各种算法具备一定的共通性，通常分为最小化路径总成本和最小化最大完成工时（Makespan）这两种。

路径总长度是指所有智能体从起点到目标位置的路径长度之和，即总移动距离最小化，该目标能够有效减少系统中智能体的总运行成本，提高整体资源利用率。对于任务路径集合 $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$，单个智能体 a_i 的路径长度为 $|\pi_i|$，最小化路径总长度的目标函数可以定义为：

$$\Pi^* = arg\min_{\Pi} \sum_{i=1}^{k} |\pi_i| \tag{2-4}$$

式中 Π^* 为使目标函数达到最优的最优路径集合。最大完成工时指的是完成所有智能体任务所需的最短时间，即最后一个到达目标的智能体的路径长度。形式化如公式 2-5 所示：

$$\Pi^* = arg\min_{\Pi} \max_{i\in\{1,\dots,k\}} |\pi_i| \tag{2-5}$$

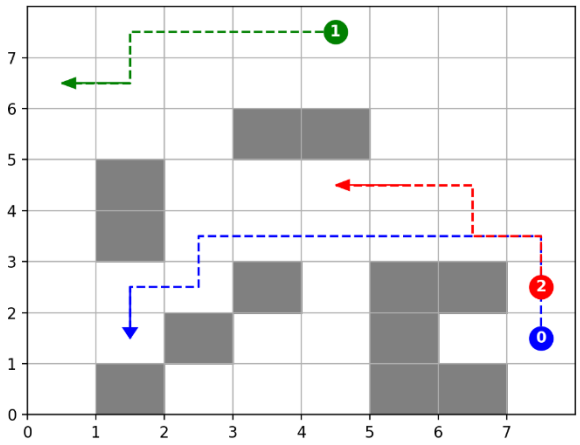


图 2-5 MAPF 示例图

实际应用中，MAPF 问题的需求往往多样化，需要在多个任务目标间进行权衡。例如，同时最小化路径总长度和完成时间，即在追求资源节省的同时也要确保任务尽早完成。多目标优化可以将多个目标结合，通过加权或其他方法将其统一在一个优化框架中，从而获得更平衡的路径规划方案。多目标优化通常使用加权和的形式，将各个目标结合为一个整体目标函数。例如，给定路径总长度权重 w_1 和 Makespan 权重 w_2，多目标优化可以表示为：

$$\Pi^* = arg\min_{\Pi}\left(w_1\sum_{i=1}^{k}|\pi_i| + w_2 \max_{i\in\{1,\dots,k\}}|\pi_i|\right) \tag{2-6}$$

通过调整权重 w_1 和 w_2，可以在总路径长度和完成时间之间找到最佳平衡。这种多目标优化广泛应用于需要兼顾多种性能指标的复杂应用场景，如物流管理、无人机群飞行等。图 2-5 提供了一个具体的示例，假设 w_1 和 w_2 均为 1，则该示例当前的路径总长度为 20，完成工时为 10。

$$\begin{aligned}\sum_{i=1}^{3}|\pi_i| &= |\pi_0| + |\pi_1| + |\pi_2| \\ &= 10 + 5 + 5 \\ &= 20\end{aligned}$$

$$\begin{aligned}\max_{i\in\{1,2,3\}}|\pi_i| &= \max(10,5,5) \\ &= 10\end{aligned}$$

表 2-1 MAPF 问题符号定义表

符号	定义
G	环境图
V	环境节点集合
E	边集合
A	智能体集合
a_i	任意智能体
s_i	智能体 a_i 的任务起点
g_i	智能体 a_i 的任务终点
π_i	智能体 a_i 当前的任务路径
$\pi_i(t)$	表示智能体 a_i 在时刻 t 时的位置
Π	任务路径集合
e_i	边
v_i	节点/顶点

2.3 冲突搜索算法的实现与优化

2.3.1 算法结构与实现流程

CBS 算法的双层结构源于 MAPF 问题的复杂度分析。传统的路径规划问题（如单智能体的 A*算法）在多智能体场景下因冲突的存在而变得复杂，冲突数量随智能体数呈指数增长。为应对这一挑战，Sharon 等人[42] 提出了 CBS 算法，将问题分解为高层冲突协调和低层独立规划两个层次。高层搜索通过构建约束树（Constraint Tree，CT）逐步解决智能体间的冲突，而低层搜索则为每个智能体生成满足约束的路径。这种分层策略通过约束的动态添加，将复杂的全局优化问题转化为可管理的局部搜索问题，在保持解最优性的前提下降低计算复杂度。

高层搜索是 CBS 算法的核心，负责协调智能体间的冲突并寻找全局无冲突解，其主要任务是通过维护一棵约束树（CT）进行最佳优先搜索（Best-First Search）。约束树的每个节点 N 包含如下信息：

(1) 约束集 *ConsN*：其中的每一个约束都属于一个智能体。CT 的根包含一组空约束，后续每个节点的子节点都会继承父节点的约束，并且给某单个智能体添加一个新的约束；

算法 1：CBS 算法的高层搜索

输入：MAPF 实例

输出：无冲突的路径方案集合

```
Root.Constraints ← ∅; //根节点 Root 设为空约束集
Root.Π ← FindPaths( ); //调用底层路径搜索算法，为每个智能体计算初始路径
Root.cost ← SIC(Root.Π); //计算方案总成本
将 Root 节点加入列表 OPEN
while OPEN ≠ ∅ do
  P ← OPEN 中成本最低的节点
  if P 中所有路径均无冲突 then
    return P.Π //该路径方案为最终解
  end
  C ← P 中第一个冲突 (a_i, a_j, v, t)
  for 冲突中的每一个智能体 do
    生成新的子节点 A
    A.Constraints ← P.Constraints ∪ {(a_i, v, t)} //为智能体a_i添加约束
    A.Π ← P.Π //继承父节点的路径方案
    A.Π ← LowLevelSearch(a_i) //调用底层搜索算法更新
    A.cost ← SIC(A.Π) //计算新方案总成本
    if A.cost < ∞ then
      将 A 插入到 OPEN 中
    end
  end
end
```

(2) 路径方案集合Π_N：每个智能体一条路径方案，集合一共包含智能体总数量条路径，且每条路径都必须符合其对应智能体所需要遵循的约束条件。每条路径由底层搜索算法确定；

(3) 总成本 *Cost*(*N*)：当前解决方案的成本（所有智能体路径成本之和），即 $Cost(N)=\Sigma_{i=1}^{n}|\pi_i|$。

具体的工作流程如伪代码所示，第一步初始化：从根节点开始，根节点无约束，调用低层搜索为每个智能体计算初始路径；第二步节点选择：从优先队列中取出成本最低的节点。高层搜索通常采用最佳优先策略，选择 *Cost*(*N*)最小的节

点以追求全局最优解；第三步冲突检测与扩展：检查当前节点的路径集Π_N是否存在冲突。若无冲突，则返回该路径集作为解；若存在冲突，则根据冲突类型（如顶点冲突或边冲突）生成新的约束，创建子节点并加入队列。

Sharon 等人已经在[42] 中证明，只要图 G 上存在可行解，且成本函数单调递增，CBS 必然能通过有限次扩展找到成本最小的无冲突解。然而，高层搜索的效率高度依赖冲突数量和约束树规模，在智能体密度较高的场景下，树节点数可能呈指数增长。

低层搜索为每个智能体生成满足当前约束的路径，是 CBS 算法的基础模块。给定高层节点 N 的约束集 $Cons_N$，低层搜索的任务是为智能体 a_i 从起点 s_i 到目标 g_i 规划路径π_i，确保π_i满足 $Cons_N$ 中的所有约束，例如避免在指定时间到达受限位置。同时路径成本$|\pi_i|$尽可能小。

低层搜索通常采用 A*算法，但为了满足 CBS 算法的需要，A*算法在状态空间中搜索的时候，除了原本障碍物导致的顶点不可用之外，还需要满足约束集 $Cons_N$ 中对智能体的约束，剪枝不可行状态。例如，若 $Cons_N$ 中包含(a_i, v_k, t_k)，则智能体 a_i 在时间步 t_k 不得经过顶点 v_k，即 a_i 的路径π_i 不得包含状态(v_k, t_k)。

2.3.2 CBS 算法的冲突解决策略

通过 2.2 节对 MAPF 问题的分析可以看出，冲突在 MAPF 问题中是不可避免的，尤其是在智能体密度较高或地图复杂度增加的场景下。CBS 算法对于冲突的解决策略就是通过对发生的冲突进行特定形式的约束以输出可执行路径方案。与 MAPF 中的顶点冲突、边冲突相对应，CBS 算法的约束分为顶点冲突约束和边冲突约束这两种。

顶点冲突约束要求任意两个智能体 a_i 和 a_j 在同一时间步 t 内不能占据相同的节点位置，如公式 2-7 所示。

$$(a, v, t) = \begin{cases} 1, & v_i^t = v_j^t \\ 0, & v_i^t \neq v_j^t \end{cases}, \quad \forall i \neq j, \quad \forall t \in \{0,1,\dots,T-1\} \tag{2-7}$$

其中，v_i^t表示在时间步 t 时，智能体 a_i 的位置。此约束确保了智能体在路径规划过程中不会在同一时间占据同一位置，从而避免了直接碰撞（如图 2-6 所示）。

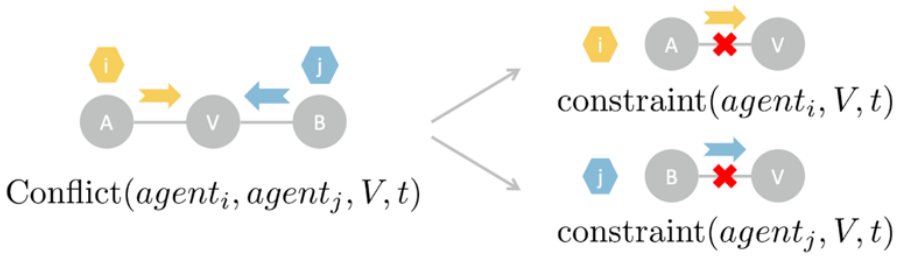


图 2-6 算法的顶点冲突约束生成

边冲突约束确保智能体在同一时间步内不会通过相同路径，从而避免路径的交叉,形式化表示为：

$$(a,(v^t,v^{t+1}),t)=\begin{cases}1, & (v_i^t,v_i^{t+1})=(v_j^t,v_j^{t+1})\\ 0, & (v_i^t,v_i^{t+1})\neq(v_j^t,v_j^{t+1})\end{cases},\quad \forall i\neq j,\quad \forall t\in\{0,1,\dots,T-1\} \tag{2-8}$$

在 CBS 算法中，冲突的检测由高层搜索结构负责，通过路径对比较实现：对每一对智能体(a_i, a_j)，比较其路径π_i和π_j在每个时间步 t 的位置 v。若$\pi_i(t)=\pi_j(t)$，记录冲突为元组(a_i, a_j, v, t)，表示智能体 a_i 和 a_j 在时间 t 发生了顶点冲突；若$(\pi_i(t), \pi_i(t+1))=(v_1, v_2)$且$(\pi_j(t), \pi_j(t+1))=(v_2, v_1)$，记录为边冲突$(a_i, a_j, (v_1, v_2), t)$。

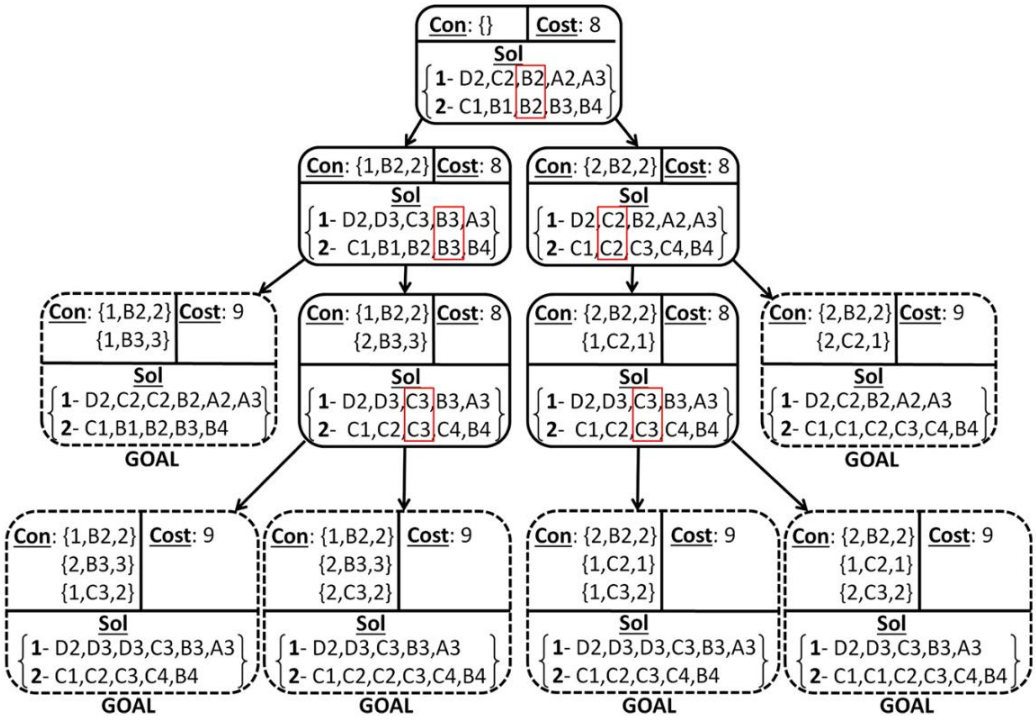


图 2-7 CBS 算法高层约束树节点的拓展[42]

冲突检测后，对于每个冲突生成一组新的约束，分别限制涉及的智能体。在约束树上则体现为，为每个冲突创建两个子节点，每个节点继承父节点的约束集，并分别添加一个新约束。例如，顶点冲突(a_i, a_j, v, t)生成：子节点 N_1，$Cons_{N1}=Cons_N \cup \{(a_i, v, t)\}$和子节点 N_2，$Cons_{N2}=Cons_N \cup \{(a_j, v, t)\}$。

随后将新约束传递给低层搜索，重新计算受影响智能体的路径。若无解，则丢弃该子节点。无论是否有解都会将结果反馈给高层，高层再根据更新路径检查新冲突，迭代直到无冲突解出现。

表 2-2 CBS 算法符号定义表

符号	定义
N	约束树的树节点
$Cons_N$	树节点对应约束集合
$Cost$	路径方案总成本
$\|\pi_i\|$	智能体 a_i 的路径成本
T	系统任务结束时间
t	时间步

2.4 本章小结

本章围绕 MAPF 问题展开了深入研究，重点探讨了 MAPF 的数学建模、路径规划目标以及冲突解决策略。MAPF 作为一种典型的多智能体协作问题，在人工智能、机器人调度和仓储自动化等领域具有广泛的应用背景，其核心目标是在复杂环境中为多个智能体规划一组无冲突路径，并在特定优化目标下提升系统的整体运行效率。

其次，本章详细介绍了 CBS 算法及其冲突搜索策略，CBS 采用双层搜索框架，将 MAPF 问题分解为高层冲突搜索和低层路径搜索两个部分。高层搜索通过约束树结构动态维护约束信息，采用最佳优先搜索选择代价最低的无冲突解。低层搜索则基于 A*算法为每个智能体独立计算符合约束条件的最优路径，并在冲突发生时调整搜索策略。下一章将探讨基于动态环境的增量式冲突搜索算法，研究如何在动态环境下高效调整路径，以减少计算开销，提高路径规划的实时性和适应性。

第3章　基于动态环境的增量式冲突搜索算法研究

3.1 引言

由第二章对 MAPF 问题和 CBS 算法的分析讨论可得，动态环境下的 MAPF 问题在实际应用中面临着复杂的挑战，尤其是在障碍物位置实时变化和任务动态分配的场景中，传统路径规划算法往往因全局重计算而导致响应速度慢、计算资源消耗高。尽管 CBS 算法及其改进方法在静态环境中表现优异，但面对频繁的动态变化，这些算法仍然难以避免全局路径重新规划的高昂代价。因此，如何在动态环境中高效调整路径以应对实时变化，成为当前 MAPF 研究中的重要课题。

为此，本研究提出了增量式冲突搜索算法（Incremental Conflict-Based Search, ICBS），以解决动态环境下多智能体路径规划的高效性和适应性问题。ICBS 算法通过构建动态环境图，明确智能体路径与环境变化的关系，并结合动态监测模块和增量更新机制，对受影响的路径部分进行局部优化，从而避免了传统 CBS 算法中全局重计算的瓶颈。相比于全局动态规划方法，ICBS 在动态变化场景中展现出更快的响应速度、更低的计算开销以及较高的路径质量。

3.2 增量式路径调整机制

3.2.1 ICBS 算法概述

ICBS 的核心思想是在路径规划过程中，通过实时监控环境变化并构建路径与环境之间的依赖关系，仅对受环境变化影响的路径进行局部调整。通过这种增量式更新策略，ICBS 避免了不必要的全局路径重计算，显著减少了计算时间和资源消耗，同时保持了路径的质量。ICBS 算法的实现可分为环境监测、增量式更新两个步骤。

因为环境是动态变化着的，所以与 CBS 算法不同的是 ICBS 问题会建模为随时间步 t 变化的无向图 G，初始时 $G(0) = (V, E(0))$，以及一组智能体集合 A。A 中的每个智能体 a_i 初始化其对应的路径集合 Π_i，这些路径从其起始位置 s_i 到目标位置 g_i。在初始化路径后，由环境监测模块，持续跟踪环境变化，如障碍物的增减或位置变化，从而更新环境图 $G(t) = (V, E(t))$。算法的主体伪代码如下所示：

算法 2：增量式冲突搜索（ICBS）算法

输入：初始环境$G(0)$，智能体集合 A 及其起点和目标点

输出：所有智能体的无冲突路径方案

初始化冲突搜索树根节点
使用底层搜索算法（A*）计算所有智能体的初始路径Π
持续监测环境变化
while True do
  if 检测到环境变化 then
    识别受影响的智能体集合$A_{affected}$以及对应的受影响路径集合$\Pi_{affected}$
    for 每个受影响的智能体$a_i \in A_{\text{affected}}$ do
      获取当前路径π_i
      重新计算路径$\pi_{i,new}$，考虑新的动态障碍物
      解决$\pi_{i,new}$与其他路径的冲突
    end
    更新A_{affected}的路径集合Π_{new}
  end
  if 所有路径均稳定，无进一步调整需求 then
    退出循环
  end
end

当环境 $G(t)$随时间变化导致路径中的某些边 e_i 变得不可用时，ICBS 将首先识别出受影响的智能体集合 $A_{affected}$。对于每个受影响的智能体 a_i，算法将根据新的障碍物布局重新计算路径$\pi_{i,new}$。为了避免路径冲突，ICBS 采用局部搜索策略来确保新路径的可行性与无冲突性，避免路径重新规划过程中产生新的冲突。

如图 3-1 的算法流程图所示，算法在环境监测过程中实时监控环境的变化，识别障碍物的添加或移除，并评估这些变化对现有路径的影响。若影响显著，系统将触发增量更新模块，对受影响的智能体集合 $A_{affected}$ 中的每个智能体 a_i 执行路径重新规划，快速为其找到避开新障碍物的路径$\pi_{i,new}$。

具体来说动态环境监测过程负责实时监测环境中的障碍物变化，包括障碍物的新增、移除或位置的变化。检测到环境变化后，模块识别出受影响的智能体路径，随后触发局部路径调整。

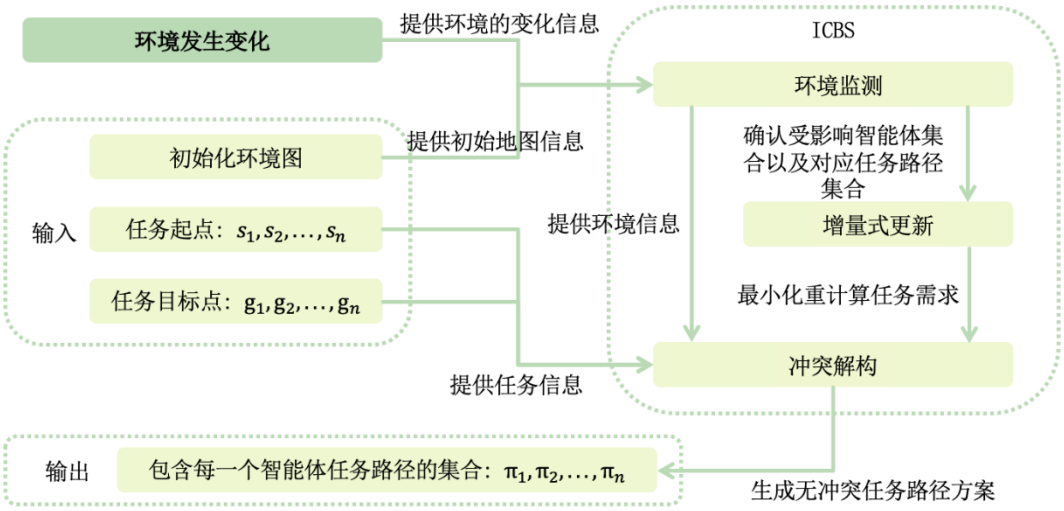


图 3-1 ICBS 算法流程图

在路径更新过程中，由于不同智能体 a_i、a_j 的路径$\pi_{i,new}$可能会互相影响，从而产生新的冲突，算法需要确保所有智能体的路径有效，并且保持无冲突性，满足公式 3-1。

$$\alpha\left(\left(\pi_i(t), \pi_i(t+1)\right), t\right) = \begin{cases} 1, & \text{若路径调整后仍然无冲突} \\ 0, & \text{否则} \end{cases}, \quad \forall i, \forall t \qquad (3\text{-}1)$$

增量式更新过程是对增量式路径调整机制的主要体现。旨在对环境监测模块中标记为受影响的智能体路径进行局部重规划，以降低计算负担，保证动态环境中的路径规划效率。

3.2.2 动态环境监测模块

在动态环境中，障碍物的出现、消失或位移会影响智能体的路径可行性。如果这些变化未能及时反映在路径计算中，可能会导致路径不可行或效率低下。动态环境监测模块不仅需要识别这些动态障碍物的变化，还要评估这些变化对智能体路径的具体影响，随后触发路径更新模块进行工作，对局部路径进行更新。该模块的伪代码如下所示。

动态环境监测模块的第一步，更新环境状态：当障碍物在环境中出现或消失时，环境图 $G(t) = (V, E(t))$中的边 e_i 的可行性需要响应变化。如公式 3-1 所示，边可行性函数$\alpha(e_i, t)$反映障碍物变化的影响。当动态障碍物导致环境图 $G(t)$的某条边 e_i 不可用时，更新 $G(t)$中对应的值，确保后续对路径受影响的评估能够基于最新的环境状态。

算法 3：环境监测

输入：当前环境图$G(t_{\text{now}})$, 障碍物变化$\Delta E(t_{\text{now}})$

输出：更新后的环境图$G(t_{\text{updated}})$，若必要则触发路径更新

$G(t_{\text{updated}}) \leftarrow \text{UpdateGraph}\left(G(t_{\text{now}}), \Delta E(t_{\text{now}})\right)$
$A_{\text{affected}} \leftarrow \emptyset$ //初始化受影响的智能体集合
for 每个智能体$a_i \in A$ do
  if 新增障碍物 $\Delta E(t_{\text{now}})$ 影响a_i的路径 then
    将a_i纳入A_{affected}，π_i纳入Π_{affected}
  end
  if 移除障碍物$\Delta E(t_{\text{now}})$可优化a_i的路径 then
    将a_i纳入A_{affected}，π_i纳入$\Pi_{affected}$
  end
end
if $A_{\text{affected}} \neq \emptyset$ then
  触发A_{affected}的路径更新
end

一旦环境图更新 $G(t)$，下一步任务就是识别哪些路径受到了影响。路径的可行性不仅与原本环境中已经记录的障碍物相关，还与更新后使得 $G(t)$发生变化的动态障碍物相关。环境监测模块需要对当前的路径方案进行逐一排查，识别哪些任务路径与动态障碍物的已记录轨迹发生了交集，最后将这些受影响的路径集合 $\Pi_{affected}$ 对应的智能体汇总成集合 $A_{affected}$。

如图 3-2 所示，该模块会遍历每个智能体的路径，如果发现路径中某个时间点的边 e_i 变得不可用，即 $\alpha(e_i, t)=0$ 时，该路径即为受影响路径，智能体 a_i 被加入到受影响智能体集合 $A_{affected}$ 中，随后触发增量更新模块开始工作。

识别受影响的智能体路径之后，监测模块也需要评估环境变化对路径的具体影响。例如，在一些情况下障碍物的出现可能仅使得路径的某一部分变得不可行，这时路径更新可能仅需局部调整；而在另一些情况下，障碍物的加入可能会导致路径完全无法继续前进，此时则需要对整个路径进行重新规划。

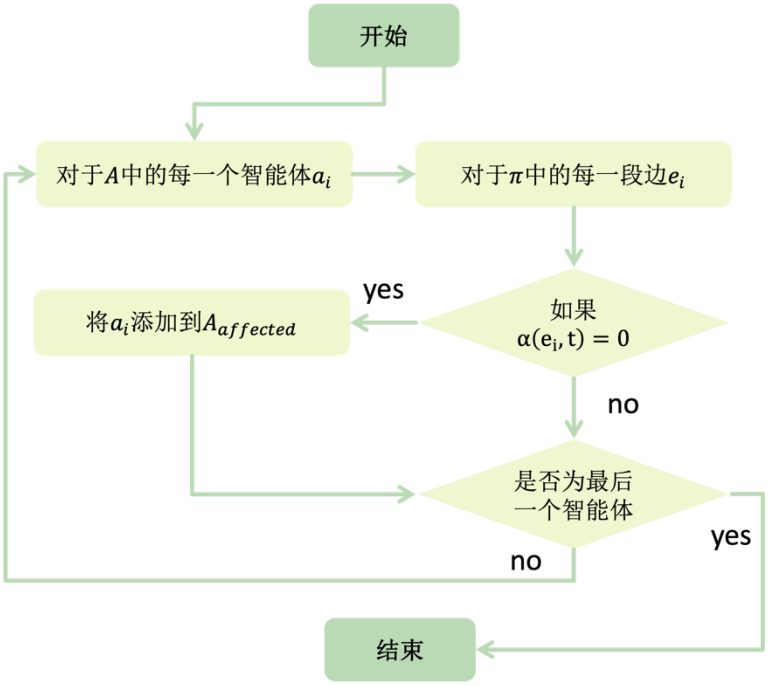


图 3-2 环境监测模块工作流程

如图 3-3 所示，原本五角星处的智能体准备从左下角出发到环境的右上角，初始时算法已经根据环境当前的状态为其规划了一个可执行路径方案。但在智能体执行任务的过程中，环境监测模块检查到存在一个动态的障碍物（圆形图例所示）由环境的右下角移动到环境的中央停止不动。那么该智能体当前的任务路径从整体上来看就失效不可执行了，但前几个时间步所规划的任务路径并未与该动态的障碍物产生冲突，所以仅需要对原本的任务方案进行局部的调整。

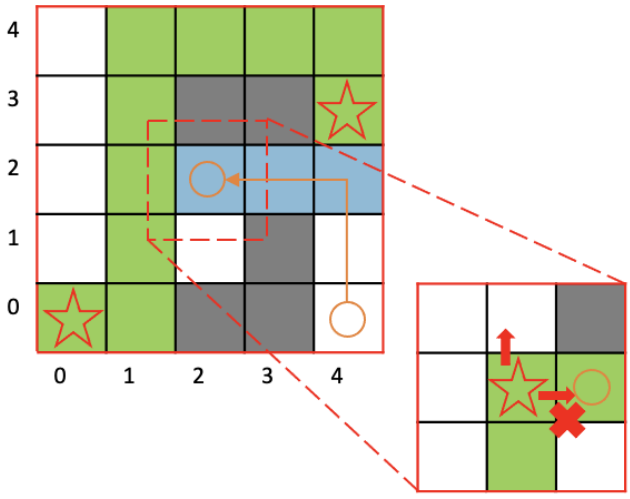


图 3-3 动态障碍物对原本任务方案的影响示意

图中原本后半部分的任务方案失效（图中蓝色路径），在动态障碍物停驻的节点前，下一时间步选择向上移动（如图中的绿色路径所示）。

总而言之，动态障碍物对于任务路径方案的影响分为局部变化和全局变化两个方面。首先在局部变化方面，如果障碍物的加入或移除仅影响路径的某些部分，则可以通过局部搜索对路径进行调整，避免大范围的重新规划；全局变化方面，如果障碍物导致路径的核心部分发生冲突或完全不可通行，则需要进行全局路径规划。通过如上评估策略，监测模块能够确定每条路径是否需要更新，并根据变化的影响程度选择局部调整还是全局规划。

3.2.3 增量式更新

算法 4：增量式路径更新

输入：受影响的智能体集合 A_{affected}，当前环境图 $G(t)$

输出：更新后的全局路径方案 Π_{new}

初始化优先队列 Q

for 每个受影响的智能体$a_i \in A_{\text{affected}}$ do

  识别路径中的受影响段 $\pi_{i,\text{affected}}$

  计算可复用路径段$\pi_{i,\text{reused}} \leftarrow \pi_i - \pi_{i,\text{affected}}$

  将 $(a_i, \pi_{i,\text{affected}})$ 加入 Q

end

while $Q \neq \emptyset$ do

  $(a_i, \pi_{i,\text{affected}}) \leftarrow$ 从 Q 中提取代价最低的智能体

  使用 A*搜索重新规划路径$\pi_{i,\text{new}}$，起点为$\pi_{i,\text{reused}}$终点为g_i

  if $\pi_{i,\text{new}}$ 有效 then

    更新环境图$G(t)$，确保新路径满足无冲突约束

    更新智能体a_i的全局路径$\pi_i \leftarrow \pi_{i,\text{reused}} \cup \pi_{i,\text{new}}$

  end

  else

    处理路径失败情况

  end

end

增量式路径更新的核心是基于环境监测模块的输出，对受影响路径集合$\Pi_{affected}$进行重新计算，未受影响的路径则保持不变，最终输出新的可执行路径方

案集合Π_{new}。特别是在仓储环境中，障碍物的变化通常仅影响局部区域，因此采用增量更新机制能够有效避免不必要的全局路径计算，提高路径调整的效率。

增量更新模块的核心逻辑如上述伪代码所示。模块的第一步先接收由环境监测模块传递来的受影响的路径集合$\Pi_{affected}$，通过环境图$G(t)$定位某个受影响的智能体对应路径π_i中的受影响部分$\pi_{i,\,affected}$。

$$\pi_{i,reused} = \pi_i - \pi_{i,affected} \tag{3-2}$$

随后对路径复用程度进行评估，即评估$\pi_{i,\,reused}$部分的有哪些是可以保留的，$\pi_{i,\,reused}$的计算如公式 3-2 所示。与上一节中对动态障碍物的影响相对应，路径复用指的是在路径更新过程中，某些路径段可能在障碍物变化后仍然是有效的，因此，路径重用策略允许系统将这些未受影响的路径段复用，而无需重新计算。这样，系统就能够在不损失效率的前提下保持路径的连贯性，并减少不必要的计算开销。

例如对于图 3-3 中的智能体来说，假设其原本的路径方案π_i的总长度为L_i，未受影响的前半部分路径长度为L_{reused}，受影响的后半部分路径长度为$L_{affected}$。则两部分对应的算法计算复杂度分别如公式 3-3、3-4 所示。

$$Complexity = O(L_{reused}\, log\, L_{reused}) \tag{3-3}$$

$$Complexity = O\left(L_{affected}\, log\, L_{affected}\right) \tag{3-4}$$

增量更新的总体复杂度为：

$$Complexity = O(L_i\, log\, L_i) - O(L_{reused}\, log\, L_{reused}) + O\left(L_{affected}\, log\, L_{affected}\right) \tag{3-5}$$

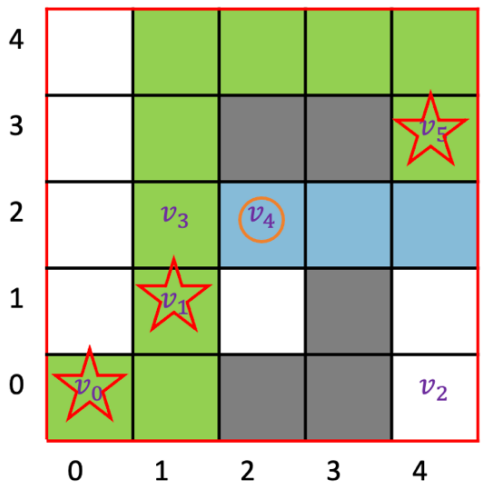


图 3-4 增量式更新部分流程示意

同样对于图 3-3 的例子来说，经过上一步路径复用评估之后可以得到原本的任务方案π_i中的前半部分$\pi_{i,\,reused}$是复用的，$\pi_{i,\,reused}$对应于图 3-4 中节点v_0到节点v_3。假设图中圆圈所代表的动态障碍物在智能体执行任务之前就已经开始运行，但是由于环境的阻隔，在初始时$G(0)$并未记录到该动态障碍物信息。假设障碍物在移动到节点v_4后停止不动，而此时智能体刚移动到节点v_1，同时环境监测检查到了该动态障碍物。那么对于路径复用评估后得到的$\pi_{i,\,reused}$，就可以拆分为起始节点v_0到当前时间步所处节点v_1，以及v_1到路径复用路段的最终节点v_3这两部分。增量式路径更新需要从当前时间步开始重新进行底层搜索规划（通常沿用 A*算法），如果此时$G(t)$记录的环境信息仍未与节点v_1到v_3的可复用路段产生冲突，则直接将该部分的可服用路段纳入新的规划方案$\pi_{i,\,new}$，并从节点v_3开始继续进行规划，直至得到完整的$\pi_{i,\,new}$。

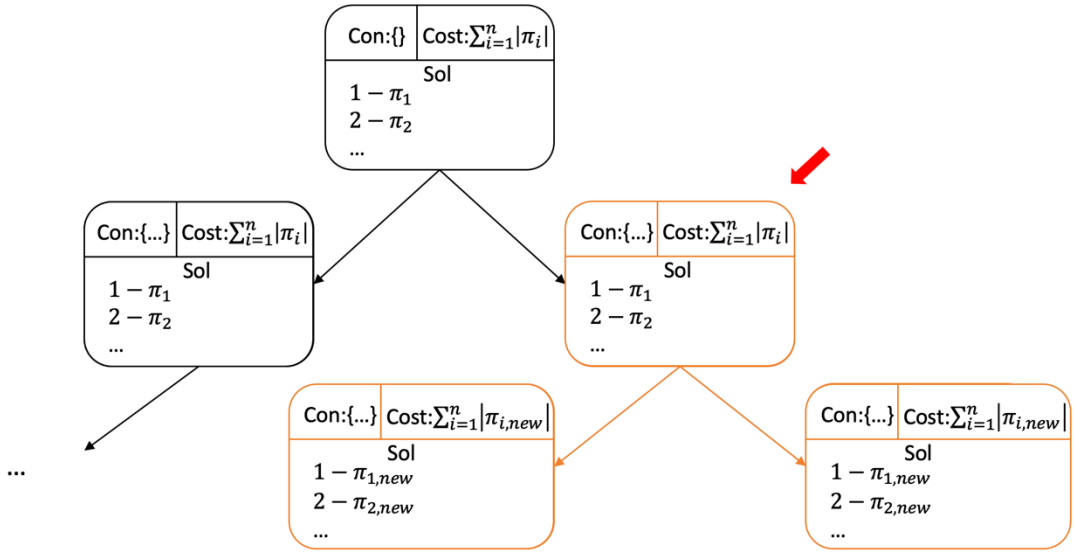


图 3-5 ICBS 算法的上层冲突搜索树拓展

这一过程映射到 ICBS 算法的上层冲突搜索树中如图 3-5 所示，当动态障碍物影响到当前树节点的规划方案时，需要根据新的环境图$G(t)$向下拓展增量式的子树节点，子树节点对应的路径方案集合Π_{new}也是需要保证是无冲突可执行的。这就要求 ICBS 算法的增量式更新过程有独特的冲突解构操作了，具体的目标就是将环境监测模块检查到的动态障碍物以约束的形式，纳入上层冲突搜索树节点的约束集合中去。保证冲突搜索树节点拓展的过程中满足新的环境限制、各个智能体之间的冲突约束以及动态障碍物带来的对应约束。

ICBS 算法的冲突解构策略会继承 CBS 算法的处理方式，将常见的冲突分为顶点冲突和边冲突这两种，对应于公式 2-1、2-2、2-3。在 ICBS 算法中，也会对应的记录成顶点冲突(a_i, a_j, v, t)、边冲突$(a_i, a_j, (v_1, v_2), t)$。但是对于动态环境下的 MAPF 问题来说，动态的障碍物不是仅针对某一个智能体的，而是对所有的智能体都会有不同程度的影响。因此动态障碍物在算法中的处理就应该类似于一个固定好路径轨迹的特殊智能体。

该步骤的伪代码如下所示：

算法 5：ICBS 的冲突解构策略

输入：路径集合Π，环境图$G(t)$，动态障碍物集合O_d

输出：无冲突的路径集合Π_{new}

```
建立冲突搜索树的根节点
OPEN ← 将初始路径方案Π插入优先队列
while OPEN ≠ ∅ do
    选取 OPEN 中代价最小的节点 P 作为当前路径方案
    if P 没有冲突 then
        返回 Π_new ← P 作为最终解
    end
    识别路径集合Π中的第一个冲突(a_i, a_j, v, t)或(a_i, a_j, (v_1, v_2), t)
    for 每个受影响的智能体a_i do
        生成新的子节点N_i，并继承父节点约束
        if a_i的路径与动态障碍物o_d ∈ O_d冲突 then
            记录动态障碍物约束(o_d, v, t)并加入N_i的约束集合
        end
        使用 CBS 低层搜索计算新的路径π_{i,new}
        if π_{i,new}可行 then
            计算新的路径成本N_i.cost并插入 OPEN
        end
    end
end
```

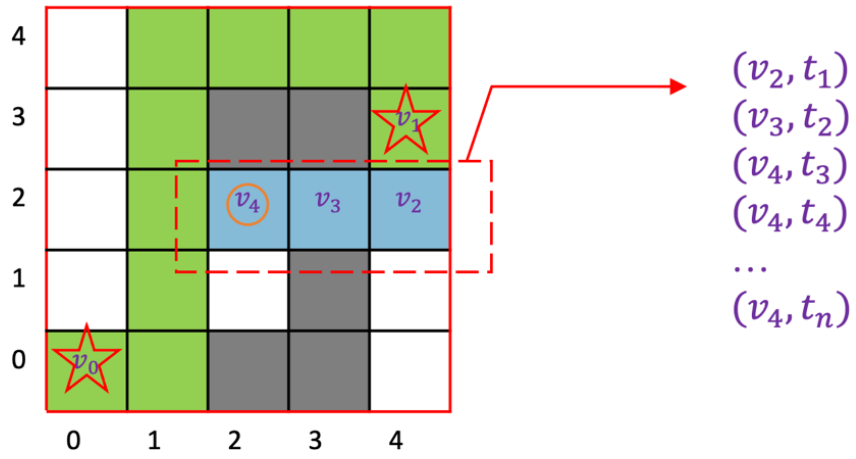


图 3-6 动态障碍物对应的约束条件

该特殊智能体只有在环境监测检过程中识别并判断为会对当前规划方案造成影响后，触发冲突搜索树向下拓展新的子节点。并且如果动态障碍物的轨迹不变，对应环境图 G 中的记录就不能改变且持续对后续所有子树节点生效。树节点中的约束集合会相应纳入新类型的约束，记为$(v_1, t_1), (v_2, t_2), \ldots, (v_n, t_n)$。其中 v_1 至 v_n 表示环境监测过程中识别到的该动态障碍物在对应时间步占据的节点资源。例如对于图 3-6 的示例，约束集合就需要纳入 $Cons$，如下所示。

$$\{(v_2, t_1), (v_3, t_2), (v_4, t_3), \ldots, (v_4, t_n)\} \rightarrow Cons$$

表 3-1 ICBS 算法符号定义表

符号	定义
$A_{affected}$	受影响的智能体集合
$\pi_{i,\,new}$	智能体 a_i 通过 ICBS 算法调整后的路径
Π_{new}	ICBS 算法调整后的路径集合
$\alpha(e_i, t)$	边可行性函数
$\pi_{i,\,reused}$	智能体 a_i 的可复用路段
$\pi_{i,\,affected}$	智能体 a_i 的受影响路段

3.2.4 实时性与效率

由于 ICBS 算法各个步骤的设计目标是确保其能够在动态变化的环境中实时响应障碍物的增减和路径受阻问题，同时将计算开销控制在合理范围内。为此，各流程中的关键结构使用更高效的数据结构表示。

(1) 环境图表示的优化

环境图 $G(t)$的表示方式可以直接决定障碍物变化的检测效率。比较常用的两种高效表示方法是邻接矩阵和邻接表。邻接矩阵一般适用于稠密图，存储复杂度为 $O(|V|^2)$，查询复杂度为 $O(1)$，但插入和删除边的操作需要更新整个矩阵，复杂度仍为 $O(|V|^2)$。

相比之下，邻接表对于稀疏图更为高效，存储复杂度为 $O(|V|+|E|)$，查询边的可用性需要 $O(deg(v))$，其中 $deg(v)$为顶点 v 的度，但动态更新的时间复杂度为 $O(1)$，适合处理动态环境变化的场景。在实际动态环境中，环境图通常为稀疏图，故选用邻接表作为环境图的数据结构。

(2) 受影响路径检测的复杂度分析

环境监测模块的关键任务是确认受环境变化影响的路径集合 $A_{affected}$。在确认过程中，如果算法逐步遍历各个智能体 a_i 的路径π_i，判断路径中是否包含受阻边 e。假设每条路径长度为L，智能体总数为n，则路径检测的总时间复杂度为$O(n \cdot L)$。

通常在实际操作中，动态变化常影响多个边 $e_1, e_2, \ldots, e_k$，形成一个动态变化区域。使用上述方法将会导致复杂度显著增加。因此，环境监测模块采用批量检测优化（区域索引）策略，通过建立路径索引表，直接查找经过受阻区域的路径集合 $A_{affected}$，进而将复杂度降低到：

$$Complexity = O(|E_{blocked}| + k \cdot \log(n)) \tag{3-6}$$

其中$|E_{blocked}|$为受阻边的数量，k 为受影响路径数，$log(n)$代表索引查询的复杂度。

(3) 增量式更新的分析

在环境监测模块触发路径更新后，增量更新的效率直接决定了系统的整体性能。以仓储环境中的动态变化为例：当某条主通道受阻时，增量更新的工作范围被限制在从断点到目标点的局部区域。这一优化使得在受影响路径集合 $A_{affected}$ 内，每条路径的更新范围从全局搜索空间 T 限制到局部区域 T_{local}，增量更新的复杂度从 $O(T \cdot logT)$降低至 $O(T_{local} \cdot logT_{local})$，其中 $T_{local} << T$ 仅在动态变化影响范围有限时成立，例如当障碍物仅影响某一通道，而非整个仓库环境。

表 3-2 ICBS 结构优化

优化策略	优化目标	优化前复杂度	优化后复杂度
局部化环境更新	只更新受影响的局部区域	$O(\|V\|+\|E\|)$	$O(\|E_{blocked}\|)$（仅更新受影响的部分）
批量障碍物检测	通过索引加速检测	$O(\|E\|)$	$O(\|E_{blocked}\|)$（只检测受阻边集合）
受影响路径索引查询	通过索引表查询路径影响情况	$O(n \cdot L)$	$O(\|E_{blocked}\|+k \cdot logn)$
增量路径更新	仅更新受影响路径的部分	$O(n \cdot TlogT)$	$O(k \cdot T_{local} logT_{local})$（仅在局部区域搜索）

(4) 实时响应性能

综上，环境监测模块经过监测环境变化（障碍物增删）、识别受阻区域（受影响的边集合 $E_{blocked}$）、检查哪些路径受影响、触发增量更新四个关键步骤及对应的策略优化后。将原本未优化情况下的总计算复杂度：

$$Complexity = O(|V| + |E|) + O(|E|) + O(n \cdot L) + O(n \cdot T \log T) \tag{3-7}$$

由于仓储环境通常是稀疏图，因此$|E| >> |V|$，并且 $n \cdot L$ 通常较大，导致计算开销非常高。最终优化后到：

$$\begin{aligned} Complexity_{monitor} &= O(|E_{blocked}|) + O(|E_{blocked}|) + O(|E_{blocked}| + k \log n) \\ &\quad + O(k \cdot T_{local} \log T_{local}) \\ &= O(|E_{blocked}| + k \log n) + O(k \cdot T_{local} \log T_{local}) \end{aligned} \tag{3-8}$$

如果$(k \cdot T_{local} logT_{local} >> |E_{blocked}|+k \cdot logn)$，则可以进一步近似为：

$$Complexity_{monitor} = O(k \cdot T_{local} \log T_{local}) \tag{3-9}$$

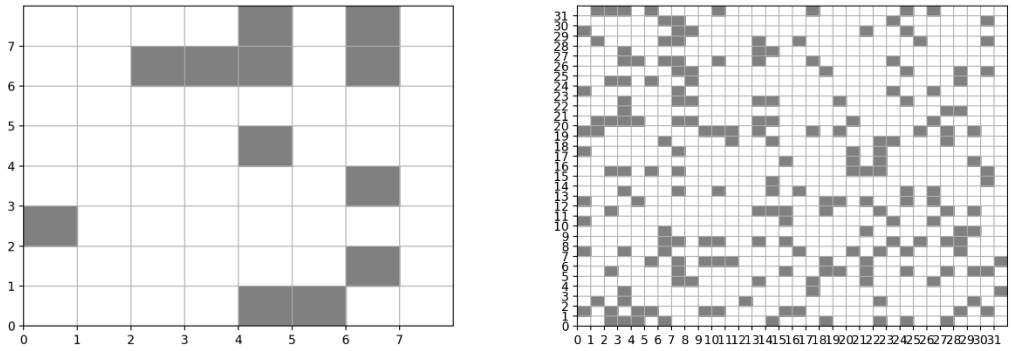
由于$|E_{blocked}| << |E|$且 $T_{local} << T$，所以优化后计算时间显著减少。若以动态障碍物密度ρ（受阻边占总边数的比例）为衡量指标，受影响区域的大小与障碍物密集度ρ呈线性关系。在稀疏环境中（$\rho << 1$），即障碍物影响范围较小时，增量更新能够有效减少计算开销，使路径更新近似达到线性时间复杂度。

3.3 仿真实验

3.3.1 实验设计

该实验旨在观察 ICBS 算法在动态环境下的性能表现，验证其在仓储环境中的适应性，尤其是在动态障碍物变化频繁的情况下，ICBS 是否能够有效降低计算开销，同时保持较好的路径质量和任务完成率。

实验使用网格地图模拟仓库结构，并通过障碍物的随机变动模拟动态环境。地图尺寸分为 8×8 和 32×32 两种，用以代表简单仓储环境和较为复杂的仓储环境（如图 3-7 所示）。在这些地图上，随机设置动态的障碍物以及多个智能体的起始点和目标点。



(a) 8x8 地图　　(b) 32x32 地图实例

图 3-7 地图实例

同时智能体数量将分为多组：在 8×8 小规模地图上为 4、5 和 6 个智能体，32×32 大规模地图上为 10 和 20 个智能体，以评估 ICBS 在不同任务密集度下的适应性。每个智能体从随机的起始位置出发，向目标位置移动，并在任务过程中根据环境信息的动态改变，触发算法的运行响应动态障碍物的影响。

为了全面评估 ICBS 的性能，实验主要关注算法的响应时间、路径质量以及系统吞吐量这几个指标，使用如下两种算法作为对比算法评估 ICBS 在动态环境中的表现：

(1) 全局 SIPP 算法(Global Dynamic Safe Interval Path Planning, GD-SIPP)

SIPP 算法[56] [57] [58] [59] 是一种专门处理动态环境下的路径规划方法，算法使用安全时间间隔来处理动态障碍物，为每个移动障碍物计算出一个安全的时间窗口，以确保智能体能够避免与障碍物发生碰撞。

(2) 全局 CBS 算法(Global Dynamic Conflict-Based Search, GD-CBS)

GDCBS 通过动态路径更新优化 CBS，使得部分路径能够在环境变化后不完全重规划，但仍然采用全局优化策略。

3.3.2 实验结果与分析

(1) 计算效率对比

算法响应时间是评估 ICBS 在动态环境下计算效率的核心指标。通过记录算法从路径规划到完成任务所用的时间，可以有效衡量 ICBS 是否能在环境变化频繁的情况下，仍保持较低的计算开销。如下几张图分别是 ICBS 算法在不同尺寸大小的栅格地图上，应对随机的突发移动障碍物的算法计算时长，与 GD-CBS、GD-SIPP 算法的比较。其中移动障碍物是随机产生的，具有随机的移动轨迹可能对任务方案产生不同程度的影响。

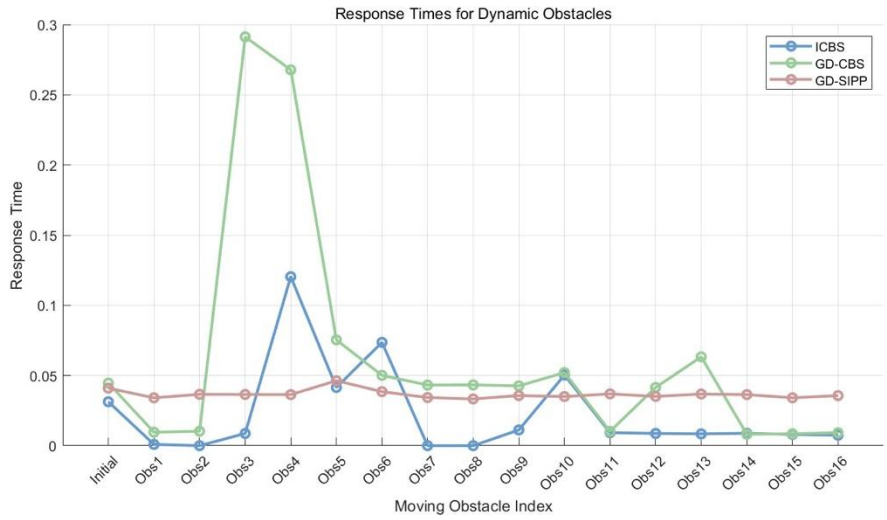
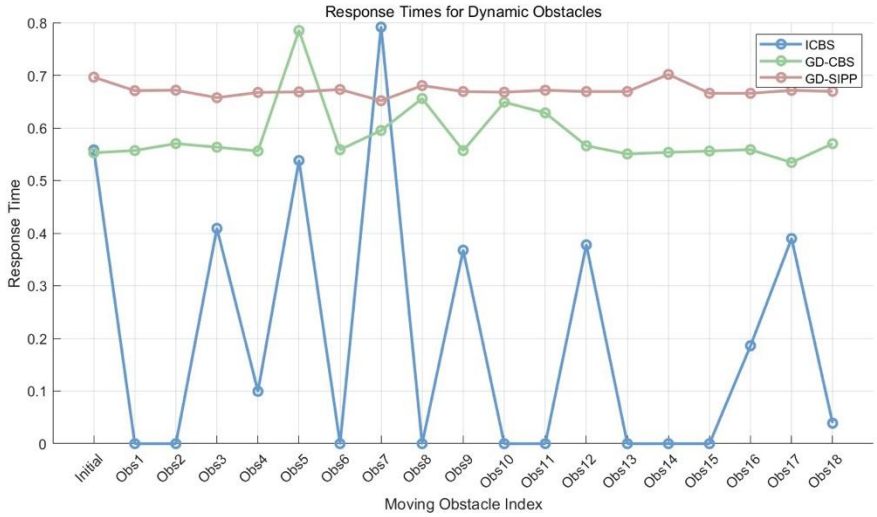


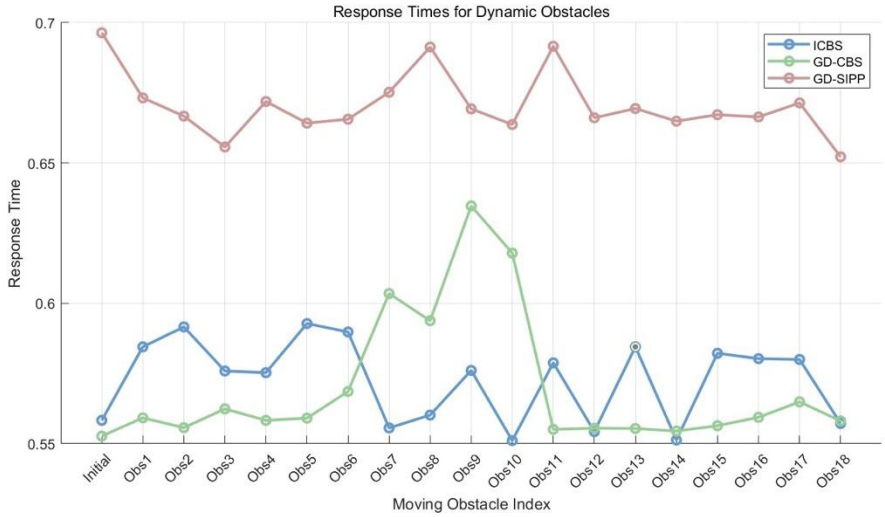
图 3-8 8x8 地图上不同算法对随机移动障碍物的响应时间

从各个算法在 8x8 小尺寸地图上收集到的算法响应时间可以看出，ICBS 执行时间普遍较低，大部分数据接近 0s，最高值仅 0.12s 左右，表明该算法整体执行效率较高且稳定。甚至存在无线逼近于 0s 的情况，这是因为 ICBS 的环境监测模块判断该动态障碍物对当前任务路径方案的影响非常小，所以基本无需复杂的计算处理。与其相比 GD-CBS 算法的整体算法响应时间和 ICBS 的波动趋势类

似，但远比 ICBS 算法的波动幅度要大，说明在计算过程中存在一定的复杂处理，某些情况下耗时明显增加。与两者不同的是 GD-SIPP 算法的波动幅度更小，但响应时间基本高于 ICBS，有时甚至不如 GD-CBS 算法，说明其算法执行效率相比另外两者还是存在一定的差距。



(a) 第一组动态障碍物的实验数据



(b) 第二组动态障碍物的实验数据

图 3-9 32x32 地图上不同算法对随机移动障碍物的响应时间

在较大尺寸的 32x32 地图上，算法响应时间的表现与小尺寸地图类似，ICBS 算法和 GD-CBS 算法虽有较大的波动，但基本都优于 GD-SIPP 算法。尤其是 ICBS 算法因其增量式路径更新机制，会使得部分情况下任务执行效率非常高，明显优于其他两者。

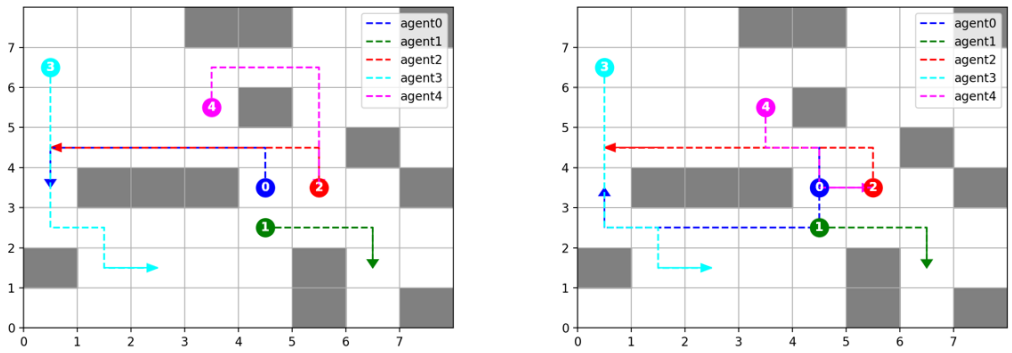
表 3-3 计算时间比较

	地图尺寸	均值(s)	最大值(s)	最小值(s)
ICBS		0.0231	0.1204	0.0009
GD-CBS	8x8	0.0630	0.2913	0.0081
GD-SIPP		0.0366	0.0463	0.0333
ICBS		0.3811	0.7916	0.0009
GD-CBS	32x32	0.5781	0.7850	0.5343
GD-SIPP		0.6704	0.7016	0.6516

通过表 3-3 的数据也可以看出，算法响应时长会随着环境范围的增大而增长，并且如果动态的障碍物对当前任务路径方案影响较大的话，三种算法的计算时长都会对应有一定程度的增长。但 ICBS 算法在相同的情况下基本都会显著低于 GD-SIPP 算法和 GD-CBS 算法的算法表现，或与两者表现相差无几。特别是当地图尺寸较大时，其计算效率的优势会更加明显。在低复杂度环境下 ICBS 算法计算时间比 GD-CBS 减少 63.3%，比 GDCBS 减少 36.8%。高复杂度环境下 ICBS 算法计算时间比 GD-CBS 减少 34.1%，比 GDCBS 减少 43.2%。这些数据都表明 ICBS 的增量更新策略能够有效减少路径规划中的计算冗余。

(2) 路径质量分析

路径质量是衡量算法最优性的重要指标之一。由于 ICBS 采用增量更新策略（如图 3-10 (b)所示），仅调整受影响路径，因此相较于 GD-CBS 算法可以节省不少任务执行开销（如图 3-10 (a)所示），减少路径方案的成本。



(a) GD-CBS 算法的响应结果　　(b) ICBS 算法的响应结果

图 3-10 8x8 地图上各算法对动态障碍物的响应结果

因为 GD-SIPP 算法在地图尺寸增加、智能体数量增加到一定程度后，算法经常会无法得到可执行的任务路径方案。所以在此仅将 ICBS 算法的任务方案成本与 GD-CBS 算法所得到的数据进行对比。在 32x32 尺寸的地图上收集到的数据如下图 3-11 所示。

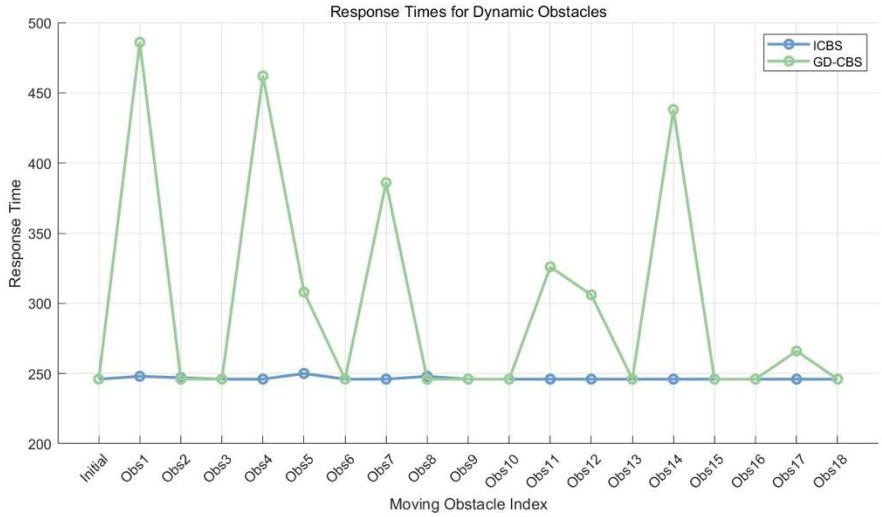


图 3-11 在 32x32 地图上算法的任务方案成本

可以看出 ICBS 算法在应对随机动态障碍物的带来的影响时，整个冲突解决方案所得到的任务方案成本数据非常稳定。与之相比较 GD-CBS 因其对动态环境的适配性较差，无论是可执行任务方案成本也好，对不同情况下动态障碍物的处理也好，存在较大的资源消耗以及波动。由表 3-4 的数据也可以看出 ICBS 任务方案的平均成本要比 GD-CBS 所得到的平均成本低 17.6%，因此在高动态环境下，ICBS 仍然是计算效率与路径质量的平衡方案。

表 3-4 任务路径成本对比

	均值	最大值	最小值
ICBS	246.47	250	246
GD-CBS	299.16	486	246

最后，在抽取了多张不同尺寸的地图上，随机设置不同数量的智能体以及随机的任务目标后，在这些不同的条件下进行重复实验最终收集到如表 3-5 所示的实验数据。

表 3-5 不同尺寸地图和智能体数量和情况下的平均实验数据

地图尺寸	智能体数量	平均任务成本			平均响应时间(s)		
		GD-CBS	GD-SIPP	ICBS	GD-CBS	GD-SIPP	ICBS
8x8	4	24.33	24.33	19	0.0048	0.0142	0.0037
	5	45.67	47.67	28.33	0.0215	0.0253	0.0102
	6	50.33	52	35	0.1072	0.0333	0.0485
32x32	10	393	失败	233	1.2551	失败	1.0695
	20	753	失败	513	1.7009	失败	1.2977

整体上来看，在小尺寸地图上 GD-SIPP 和 GD-CBS 算法的成本接近，但 ICBS 拥有最低的平均成本。同样的 GD-CBS 和 ICBS 算法的算法响应时间的浮动可能比 GD-SIPP 更为剧烈，但前两者在环境影响较小的时候总是能响应的减少冗余计算。其中 ICBS 的表现明显更优，平均算法响应时间在三者之间也明显更小。在大尺寸地图上三种算法的表现与小尺寸地图类似，但是 GD-SIPP 算法因其不能很好的处理边冲突，所以会导致任务失败率较高。相比较之下 ICBS 算法就能更好的应对当前实验尺寸地图以及智能体数量下的动态环境影响。

3.4 本章小结

本章围绕动态环境下多智能体路径规划的挑战，提出了一种基于增量式冲突搜索的优化算法，以减少传统 CBS 算法在动态环境中的计算冗余和全局重规划开销。ICBS 通过环境监测模块识别受影响的路径，并采用增量式更新策略，调整必要部分，从而提高计算效率和系统的实时响应能力。同时，算法引入优化后的冲突解构机制，确保路径调整过程中保持无冲突性。

实验结果表明，ICBS 在不同规模的动态环境下均表现出优越的计算效率和路径质量，相较于传统 CBS 和其他改进算法，其平均计算时间降低 30%至 60%，任务路径成本亦有所优化。这表明 ICBS 能够在复杂环境中有效适应动态变化，在保证无冲突路径的同时，有效减少计算开销，提升算法在动态环境下的稳定性和计算效率。其增量式路径调整机制为动态 MAPF 问题提供了一种高效可行的解决方案。下一章将从另一个角度，进一步探索动态环境下的多智能体路径规划问题的解决方案，并结合应用场景进行更深层次的优化分析。

第4章 动态探索型冲突搜索算法的设计与分析

4.1 引言

在 ICBS 算法提供的的增量式更新机制下，虽然算法其在处理环境变化时表现出一定的性能提升，但受限于依赖图的约束，路径调整必须按照特定的依赖关系逐步执行。这在高动态环境中可能限制路径优化的自由度，导致部分调整后的路径并非全局最优解。此外，ICBS 主要采用被动的增量更新策略，即仅在检测到路径受影响后才进行调整，而不会主动探索未知环境。在某些动态环境（例如智能体需要进入新区域但缺乏足够的环境信息）下，ICBS 由于缺乏主动探索能力，可能选择次优路径，从而影响路径质量。

因此，为了进一步提升算法在动态环境下的适应性，本章从多智能体系统主动探索能力的角度，提出动态探索型冲突搜索算法（Dynamic Exploratory Conflict-Based Search, DE-CBS）。

4.2 DE-CBS 的设计与实现

4.2.1 DE-CBS 的基本框架

DE-CBS 的核心在于其动态探索机制的实现，使得智能体在路径计算过程中能够结合探索信息，优化路径搜索与路径调整策略。在一个标准的 MAPF 问题中，假设智能体集合为 $A = \{a_1, a_2, \ldots, a_n\}$，环境为离散网格图 $G = (V, E)$，则每个智能体 a_i 具有其特定的起始位置 $s_i \in V$、目标位置 $g_i \in V$，以及路径 $\pi_i = (s_i, v_1, v_2, \ldots, g_i)$，满足 $\pi_i(t) \in V$，其中 $\pi_i(t)$ 表示智能体在时间步 t 时的位置。而 DE-CBS 则在 CBS 结构的基础上，扩展了路径计算的约束模型，引入探索路径集合 Π^{exp}，使得路径计算能够结合探索数据优化路径规划：

$$\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\},\ \ \Pi^{\exp} = \{\pi_1^{\exp}, \pi_2^{\exp}, \ldots, \pi_n^{\exp}\} \tag{4-1}$$

其中 Π 为当前无冲突路径集合，代表智能体的任务路径，Π^{exp} 为探索路径集合，代表智能体在路径计算过程中主动发现的可通行路径。在路径搜索过程中，DE-CBS 结合 Π^{exp} 进行路径优化，以提高路径计算的稳定性和适应性。

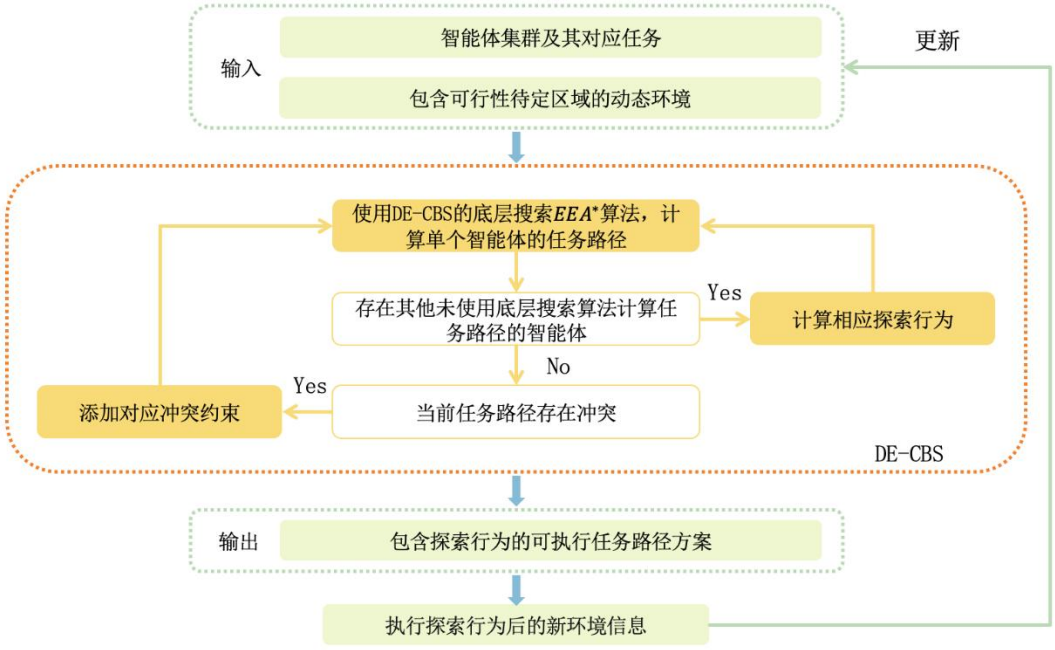


图 4-1 DE-CBS 算法工作流程

在结构上 DE-CBS 沿用了 CBS 的高低层冲突搜索框架，并在此基础上扩展了低层搜索优化、高层搜索优化（冲突树优化）及依赖图管理，以提高路径计算的适应性。

DE-CBS 的低层使用探索增强型 A*算法（Exploratory Enhanced A*, EEA*）进行路径搜索，使智能体能够在路径计算过程中结合探索信息，提高路径适应性。该算法与 A*算法的区别在于，它在路径搜索过程中，引入探索因子$\phi(v)$以优化路径选择，并结合探索启发式策略提高路径规划的全局最优性。DE-CBS 的高层在 CBS 传统冲突树搜索框架的基础上，扩展搜索节点结构，使得路径优化能够结合探索数据进行路径调整。在路径冲突检测过程中，结合探索路径信息，提高路径优化的全局协调性。

同时在 CBS 结构的依赖图基础上，增加探索路径管理机制，使得智能体能够共享探索信息，提高路径优化能力。通过探索依赖图记录智能体的探索路径Π^{exp}，确保探索信息不会影响任务路径，并优化路径调整的计算开销。

4.2.2 低层搜索：探索增强型 A*算法

(1) EEA*的优化目标

传统 CBS 的底层搜索一般采用 A*添加时序纬度的方式，以在环境已知的情况下寻找最短无冲突路径。在工业仓储环境中 AGV 经常需要与工人协作完成任

务，其面临或处理的环境是高动态的，这就会导致环境存在一定的未知性。为了解决这一特定问题，EEA*算法应运而生。

EEA*算法沿用了 A*算法的基本框架，例如使用$f(v) = g(v)+h(v)$作为评估函数[36] 的基础进行路径搜索，其中 $g(v)$表示从起始点到当前节点 v 的累积代价；$h(v)=||v-g_i||$表示智能体 a_i 从 v 到目标点 g_i 的启发式估计值，具体函数通常采用曼哈顿距离或欧几里得距离[61] （如公式 4-2 所示）。

$$h_{\text{Manhattan}}(v) = |x_1 - x_2| + |y_1 - y_2| \tag{4-2}$$
$$h_{\text{Euclidean}}(v) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

但当某一区域包含未探测的障碍物时，为了使 EEA*算法能够提前考虑该区域可能造成的路径阻碍，同时对其探索行为进行一定的控制，避免智能体在路径搜索过程中仅关注最短路径，而不会考虑可能的绕行方案或潜在的可通行路径。EEA*在路径计算过程中引入探索因子$\phi(v)$，使得路径搜索不仅基于当前环境的最优路径，还能够结合探索行为优化路径规划。

(2) EEA 的路径代价函数

具体而言，对 EEA*中 A*的代价函数进行扩展，使其能够在路径搜索过程中结合探索信息进行动态优化：

$$f(v) = g(v) + h(v) + \lambda \cdot \phi(v) \tag{4-3}$$

式中λ是探索因子的权重系数，控制探索行为对路径计算的影响。$\phi(v)$是探索因子，即探索潜力函数（Exploration Potential Function），用于衡量当前节点 v 是否值得探索。而探索潜力函数$\phi(v)$由以下因素决定：

$$\phi(v) = \sum_{u \in N(v)} \beta \cdot \big(1 - \alpha(u)\big) \tag{4-4}$$

其中 $N(v)$表示节点 v 的邻接节点集合。$\alpha(u)$表示节点 u 是否已被探索（$0 \le \alpha(u) \le 1$）。若$\alpha(u) = 1$，表示 u 已被探索，则$\phi(v)$贡献较小。若$\alpha(u) = 0$，表示 u 尚未被探索，则$\phi(v)$贡献较大。β是探索影响因子，用于调整探索行为的强度。

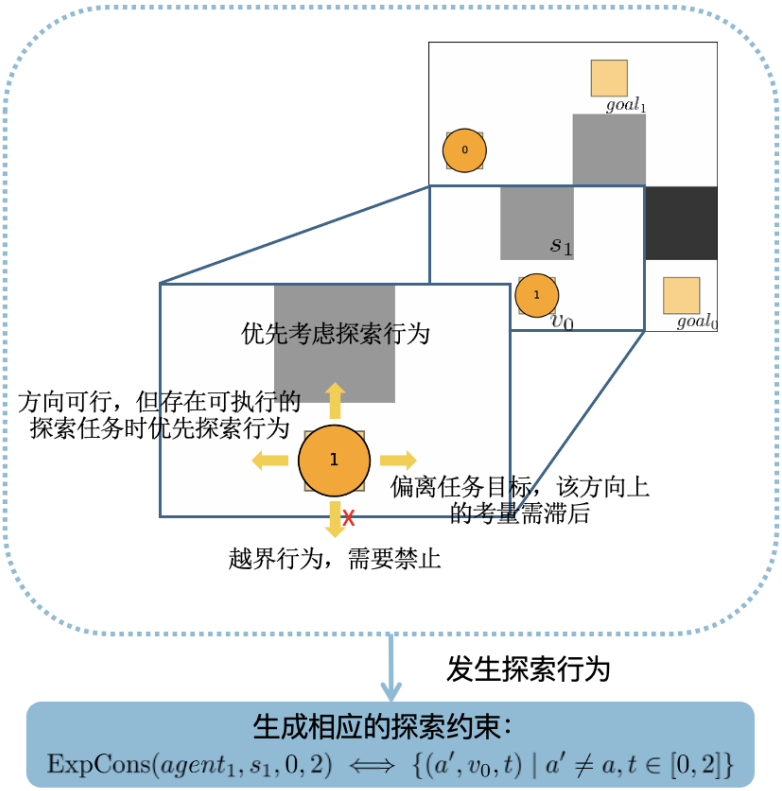


图 4-2 EEA*对可行性待定区域的探索以及对应约束生成示意

直观上理解，若$\phi(v)$取值较大，则意味着节点 v 周围尚有大量未探索区域，智能体应考虑进行探索。若$\phi(v)$取值较小，则智能体应遵循最短路径搜索策略，优先执行任务路径。通过调整$\lambda(\lambda < 0)$，智能体可控制探索行为的优先级，使得在高动态环境下，探索行为既不会影响任务完成效率，又能有效收集环境信息，提高路径计算的全局最优性。

(3) EEA*在动态环境下的路径优化

EEA*的路径搜索过程如伪代码所示。首先是初始化阶段，将智能体的起始位置 s_i 作为起点，并初始化搜索队列 Q。紧随其后进行路径扩展，对于当前节点 v，计算$f(v) = g(v)+h(v)+\lambda \cdot \phi(v)$。若$\phi(v)$较大，智能体在选择路径时倾向于探索未知区域。最后判断是否达成路径搜索终止条件，若搜索到目标位置 g_i，则路径搜索结束。若智能体在搜索过程中发现新的可通行区域，则更新路径规划，并返回优化后的路径。

算法 6：探索增强型 A*算法（EEA*）

输入：起始节点 s，目标节点 g，环境图 G

输出：从 s 到 g 的路径 π 或失败

使用$\big(s, f(s) = g(s) + h(s) + \lambda \cdot \phi(s)\big)$初始化优先队列 Q
初始化$g(s) = 0$，所有其他节点的$g(v) = \infty$
标记所有节点未被探索
while Q 不为空 do
  弹出 Q 中$f(v)$最小的节点 v
  if v == g then
    返回路径π
  end
  for 每个相邻节点 u do
    计算代价$g(u) = g(v) + c(v, u)$
    计算启发式$h(u) = \text{heuristic}(u, g)$
    计算潜力函数$\phi(u) = \sum_{w \in N(u)} \beta\big(1 - \alpha(w)\big)$
    计算评估函数$f(u) = g(u) + h(u) + \lambda \cdot \phi(u)$
    if $g(u)$ 优于之前的路径 then
      记录最佳路径节点 u
      将$\big(u, f(u)\big)$纳入 Q
    end
  end
end
return 失败

图 4-3 是对这一过程的具体展示。开始时 $agent_1$ 位于其对应的任务起点 s_1 处，此时 $agent_1$ 的下一时刻状态有三个可选项，分别是原地不动、向左移动到节点 v_1 或者向右移动到节点 v_2。由公式 4-3、4-4 计算此时三个下一状态的代价评估函数数值，具体 s_1、v_1、v_2 的f值分别如下：

$$\begin{aligned} f(s_1) &= h(s_1) + g(s_1) + \lambda \cdot \phi(s_1) \\ &= 4 + 0 + (-2) \times \big(2 \times (0 + 0 + 1)\big) \\ &= 0 \end{aligned} \tag{4-5}$$

$$\begin{aligned} f(v_1) &= h(v_1) + g(v_1) + \lambda \cdot \phi(v_1) \\ &= 6 + 0 + (-2) \times \big(2 \times (0 + 0)\big) \\ &= 6 \end{aligned} \tag{4-6}$$

$$\begin{aligned} f(v_2) &= h(v_2) + g(v_2) + \lambda \cdot \phi(v_2) \\ &= 3 + 0 + (-2) \times \big(2 \times (0 + 0 + 0)\big) \\ &= 3 \end{aligned} \tag{4-7}$$

由上可得$f(s_1)$的值最小，此时下一状态选择停留在原地并触发 $agent_1$ 的动态探索行为会被优先考虑。故随后如图 4-3 的第二张子图所示，$agent_1$ 开始进行对应的对周遭环境的可行性确认。

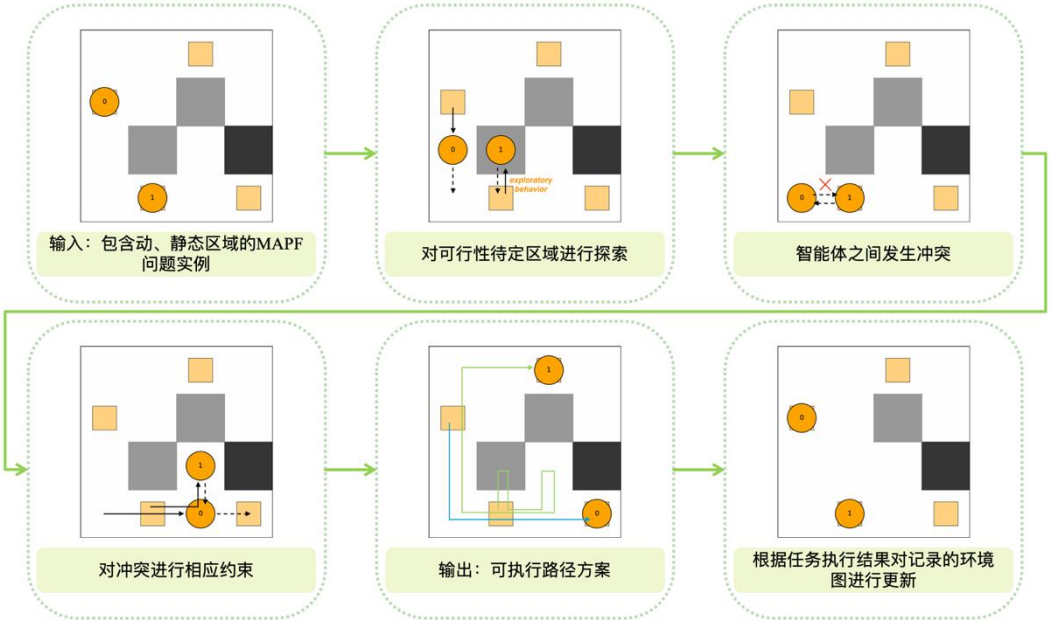


图 4-3 未探索区域探索过程

探索任务结束后 $agent_1$ 回到原本的任务起点，此时通过一般的 CBS 算法流程计算，原本应该前往的下一节点与 $agent_0$ 的当前位置产生了冲突。故 $agent_1$ 需要对其进行避让，如后续一系列状态转移所示一直到最终两者的任务结束。

(4) EEA*的数学分析

综上述分析可得，EEA*的计算复杂度主要取决于$\phi(v)$的计算开销。假设智能体路径搜索的平均展开分支数为 b，搜索深度为 d，则传统 A*的计算复杂度为 $O(b^d)$。而 EEA*在每个节点计算$\phi(v)$，其计算复杂度增加至 $O(b^d+|N(v)|)$，其中$|N(v)|$为探索潜力计算的邻接节点数量。

当探索因子$|\lambda|$取值较小时，EEA*的计算复杂度接近 A*。当$|\lambda|$取值较大时，EEA*可能需要额外计算多个探索路径，导致计算复杂度略有增加。但由于$\phi(v)$仅在部分关键区域被计算，因此整体计算开销仍然可控。

EEA*仍然满足一致性启发式，即：

$$h(v) \le c(v,u) + h(u), \quad \forall v,u \tag{4-8}$$

其中 $c(v, u)$为从 v 到 u 的路径代价。由于$\phi(v)$仅在路径搜索过程中提供启发式引导，并不会改变 A*的最优性，因此 EEA*保证最终搜索出的路径仍然是最优路径。

4.2.3 高层搜索：冲突树的优化

DE-CBS 算法的高层搜索结构伪代码如下所示。

算法 7：DE-CBS 高层搜索

输入：初始路径集合Π，冲突集合 C，探索路径集合Π^{exp}

输出：无冲突的路径集合 Π 或失败

初始化优先队列 Q
插入根节点$N_0 = (\Pi, C, Cost)$到 Q
while $Q \neq \emptyset$ do
  弹出 Q 中代价最小的节点$N = (\Pi, C, Cost)$
  if C 为空 then
    返回Π作为最终解
  end
  选择优先级最高的冲突$c = (a_i, a_j, v, t)$
  if 存在探索路径Π^{exp}可消解 c then
    更新路径集合$\Pi \leftarrow \Pi - \Pi^{exp}$
    计算新的冲突集合 C 并插入新的搜索节点到 Q
  end
  else
    生成子节点N_1和N_2，并为a_i和a_j添加约束
    使用底层 EEA*算法计算新路径Π_1和Π_2
    插入N_1和N_2到 Q
  end
end

在 CBS 结构中，高层搜索负责管理路径冲突，并通过冲突树逐步优化智能体的路径集合，使得最终计算出的路径能够满足无冲突约束。传统 CBS 通过构建二分搜索结构来解析路径冲突，即在发生冲突时，为冲突的智能体分别生成两个分支，并在每个分支中添加约束，形成冲突树的扩展（如图 2-7 所示）。

虽然这种策略可以有效解决冲突，但在高动态环境下，其计算开销较大。因为冲突树扩展速度很快，搜索空间会迅速增长，导致计算复杂度上升。同时路径调整策略单一，无法主动规避未来可能的冲突，仅能依赖二分策略逐步排除冲突，缺乏全局优化能力。并且冲突调整未结合探索信息，导致部分路径优化时仍然基于静态环境信息，而无法结合已探索数据优化路径更新。

为了提高 CBS 在高动态环境中的搜索效率，DE-CBS 在高层搜索中引入冲突树优化策略，并结合探索依赖图（Exploration Dependency Graph, EDG），使得路径冲突的优化不仅依赖已有路径信息，还能结合探索数据，使路径调整更加智能化和全局优化。本节将首先分析 CBS 传统冲突树的数学模型，然后探讨 DE-CBS 在冲突树优化方面的改进策略，并给出相应的数学推导。

(1) 冲突树的数学建模

在 CBS 结构中，冲突树（CT）用于管理路径冲突，每个节点 N 包含一组该节点规划出的任务路径集合 $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$，探索路径集合 Π^{exp} 及其相应的冲突约束集合 $Cons$ 等：

$$N = (\Pi, Cons, \text{Cost}) \tag{4-9}$$

其中Π为当前任务路径集合。$Cons$ 为冲突约束集合包含动态探索约束和一般冲突约束两种。C 为路径冲突集合，每个冲突 $c \in C$ 由(a_i, a_j, v, t)组成，表示在时间步 t，智能体 a_i 和 a_j 在位置 v 发生冲突$\pi_i(t) = \pi_j(t)$。$Cost$ 表示当前任务方案的总代价，通常设为所有路径长度之和，如式 4-7 所示。

$$Cost = \sum_{i=1}^{n} \left| \pi_i + \pi_i^{\exp} \right| \tag{4-10}$$

在 CBS 结构中，当检测到冲突 $c = (a_i, a_j, v, t)$时，冲突树会进行二分搜索扩展，即在 N 的基础上生成两个子节点 N_1 和 N_2：N_1 添加约束 $cons_1 = (a_i, v, t)$，即智能体 a_i 不能在时间步 t 到达 v。同样的 N_2 添加约束 $cons_1 = (a_j, v, t)$，即智能体 a_j 不能在时间步 t 到达 v。然后再对 N_1 和 N_1 进行低层搜索，分别计算新的无冲突路径集合Π_1 和Π_2，并将其加入搜索队列。

这种搜索策略保证了路径的无冲突性，但在高动态环境下，其搜索效率较低。首先冲突树扩展过快，导致搜索空间指数级增长，计算复杂度达到 $O(2^{cn})$，其中 cn 为冲突数量。其次未结合探索数据，路径调整过于依赖当前路径信息，而未结合智能体可能发现的新可行区域。为了解决这些问题，DE-CBS 在 CBS 结构的

基础上，优化冲突检测和路径调整策略，使得搜索过程能够结合探索信息，提高路径计算的全局优化能力。

DE-CBS 主要从冲突检测优化、路径调整策略优化和探索依赖图（EDG）管理三个方面优化冲突树，使路径计算更加智能化。在 CBS 结构中，冲突检测主要依赖路径交叉检测，即判断是否存在：

$$\pi_i(t) = \pi_j(t) \quad 或 \quad \pi_i(t+1) = \pi_j(t) \wedge \pi_i(t) = \pi_j(t+1) \tag{4-11}$$

DE-CBS 在此基础上引入探索冲突检测：如果 a_i 和 a_j 的路径发生冲突，则首先检查是否存在探索路径 π_i^{exp}、π_j^{exp} 能够避免冲突。若 π_i^{exp} 可行，则优先使用 π_i^{exp} 替换 π_i，避免额外的冲突树扩展。仅在 π_i^{exp} 和 π_j^{exp} 都不可行时，才进行传统的二分约束扩展。当 $\pi_i^{exp}(t) \neq \pi_j(t)$ 时，路径冲突可由探索路径消解，无需扩展冲突树。而当 $\pi_i^{exp}(t) = \pi_j(t)$ 时，则需要执行传统 CBS 二分搜索。

同时 DE-CBS 在 CBS 传统的二分约束搜索策略基础上，引入路径权重启发式（Path Weight Heuristic, PWH），用于调整冲突解决的优先级。路径权重函数如下所示：

$$W(\pi_i) = \sum_{t=0}^{T} \left(|\pi_i(t) - g_i| + \gamma \cdot h\left(\pi_i(t)\right) \right) \tag{4-12}$$

式中 $|\pi_i(t)-g_i|$ 表示路径当前步与目标步的距离，$h(\pi_i(t))$ 是从 $\pi_i(t)$ 到目标点 g_i 的启发式估计函数，γ 为启发式估计的权重参数。路径权重越大，表示路径调整的代价越高，DE-CBS 在选择约束添加策略时，优先选择权重较小的路径进行调整，以减少全局路径开销。

算法在 CBS 结构的依赖图之上扩展探索依赖图（EDG），用于管理智能体的路径调整顺序：若某智能体 a_i 发现新的可行探索路径 π_i^{exp}，则 EDG 记录该路径的可行性。若 π_i^{exp} 影响了 a_j，则在 EDG 中建立 (a_i, a_j) 依赖关系，表示路径调整的影响范围。在执行路径调整时，优先使用 EDG 管理的探索路径，减少全局路径调整开销。

(2) 依赖图构建与探索约束管理

在 MAPF 问题中，路径计算通常涉及多个智能体的相互影响。在用于解决 MAPF 问题的方法中，依赖图常被用于管理智能体之间的路径依赖关系，确保路

径调整的合理性，并减少计算冗余。DE-CBS 算法沿用并拓展了这一结构，使其能够记录探索路径信息、管理路径调整顺序，并优化路径依赖约束。

EDG 首先是一个有向图，用于描述智能体路径之间的依赖关系及探索路径对路径优化的影响。使用 $G_D = (V_D, E_D)$记为探索依赖图，其中 V_D 表示智能体路径，每个智能体 a_i 的路径π_i 作为一个独立的依赖节点 v_i，$v_i \in V_D$。E_D 表示路径间的依赖关系。若智能体 a_i 的路径π_i 的更新会影响 a_j 的路径π_j，则建立一条有向边 $(a_i, a_j) \in E_D$，表示路径更新的依赖关系。

在 DE-CBS 结构中，探索依赖图不仅记录路径调整关系，还记录智能体在路径搜索过程中发现的探索路径信息。在一般的路径优化过程中，智能体 a_i 的路径 π_i 与智能体 a_j 的路径π_j 存在的依赖关系主要包括如下两种：

(a) 任务路径依赖：

$$\pi_i(t) = \pi_j(t) \vee \pi_i(t) = \pi_j(t+1), \quad \exists t \tag{4-13}$$

若两个智能体在某一时间步 t 发生路径冲突，则 a_j 的路径π_j 需要在 a_i 重新规划路径后进行调整。

(b) 探索路径依赖：

$$\pi_i^{exp}(t) \neq \pi_i(t), \quad \exists t \tag{4-14}$$

若智能体 a_i 发现了一条探索路径π_i^{exp}，则在路径调整时，a_i 应优先参考π_i^{exp}，而不是直接进行全局路径重计算。

构建一个依赖图 G_D 第一步需要对其进行初始化，对所有智能体 $a_i \in A$，建立对应的依赖图节点 v_i，并初始化无依赖边 $E_D = \varnothing$。随后对任务路径依赖进行构建，计算出所有智能体路径π_i 是否存在路径交叉点$\pi_i(t) = \pi_j(t)$，若存在，则建立路径依赖边$(a_i, a_j) \in E_D$。然后再构建探索路径依赖，计算所有智能体的探索路径π_i^{exp}，并检查其对其他智能体路径的影响，若π_i^{exp} 可优化 a_j 的路径π_j，则在 G_D 中建立依赖边(a_i, a_j)。最后根据 G_D 的拓扑结构，整理路径更新顺序，路径优化时优先更新无前驱节点的智能体路径，确保路径调整的合理性。实现这一过程的具体伪代码如下所示：

算法 8：探索依赖的构建

输入：智能体集合 A，初始路径集合Π，探索路径集合Π^{exp}

输出：探索依赖图$G_D = (V_D, E_D)$

初始化探索依赖图$G_D = (V_D, E_D)$
for 每个智能体$a_i \in A$ do
  在V_D中添加节点v_i
  for 每个与存在冲突的智能体a_j do
    if π_i与π_j发生任务路径冲突 then
      在E_D中添加任务路径依赖边(v_i, v_j)
    end
  end
  for 每个探索路径$\pi_i^{\exp}$ do
    for 每个受$\pi_i^{\exp}$影响的智能体a_j do
      if $\pi_i^{\exp}$可优化π_j then
        在E_D中添加探索路径依赖边(v_i, v_j)
      end
    end
  end
end
return G_D

除了路径优化过程中所需要遵循的两种依赖关系，探索路径π_i^{exp}还需要满足一定的约束，以确保其不会影响主任务路径的执行。第一个是路径交叉约束，其目的是为了确保探索路径不会与其他智能体路径发生冲突，如下所示：

$$\pi_i^{\exp}(t) \neq \pi_j(t), \quad \forall i, j, t \tag{4-15}$$

第二个是任务优先约束，目的是确保探索路径不会影响主任务路径执行：

$$\sum_{t=0}^{T} \left|\pi_i^{\exp}(t) - g_i\right| \geq \sum_{t=0}^{T} \left|\pi_i(t) - g_i\right|, \quad \forall i \tag{4-16}$$

该约束保证了了 DE-CBS 的任务路径优先级调度策略，确保任务路径π_i在调整过程中优先级高于探索路径π_i^{exp}。路径优先级函数如公式 4-9 所示。在路径调整时，若 $W(\pi_i^{exp}) < W(\pi_i)$，则优先考虑探索路径进行路径优化，否则继续使用原任务路径。

表 4-1 符号定义表

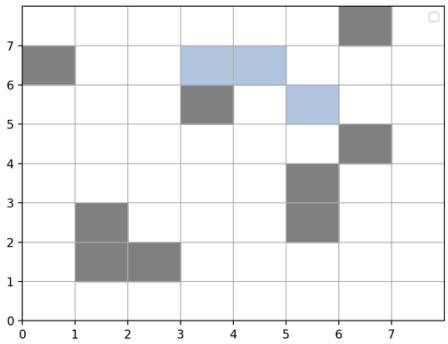
符号	定义
Π^{exp}	探索路径集合
π_i^{exp}	智能体 a_i 的探索路径
$\phi(v)$	节点 v 的探索潜力函数
$f(v)$	节点 v 的代价评估函数
λ	探索因子的权重系数
β	探索影响因子
$N(v)$	节点 v 的邻接节点集合
$\alpha(v)$	节点 v 的被探索标记
$c(v, u)$	从节点 v 到节点 u 的路径代价
N	冲突树上的树结点
C	路径冲突集合
$Cons$	冲突约束集合
$cons_i$	冲突约束
$W(\pi_i)$	路径权重启发式函数

4.3 仿真实验

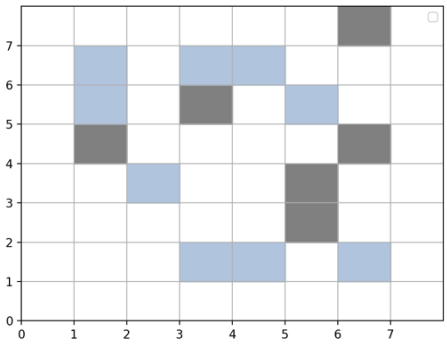
4.3.1 实验设计与评估

为了验证 DE-CBS 算法在动态环境中的有效性，本节设计了一系列实验，旨在评估 DE-CBS 在高动态环境下的性能，并与传统的 CBS 算法进行对比。通过这些实验，验证 DE-CBS 在解决动态环境中多智能体路径规划问题时，能否提供比传统 CBS 更好的性能和优化效果。

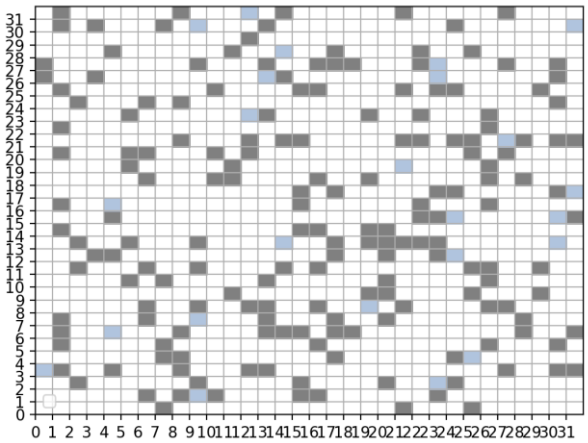
与上一章 ICBS 算法的实验条件设置类似，本章实验也采用标准网格环境进行仿真，选择 8×8 和 32×32 两种不同规模的地图，分别模拟小型仓储和大规模仓储环境确保实验结果具有代表性。实验的主要变量是环境的动态性和智能体的数量。对于环境动态性，根据可行性待定区域在障碍物区域中的比例，实验设计了低动态、中动态和高动态三种场景，以模拟不同规模的环境变化对路径规划的影响。智能体数量设定为 4、5、6、10、20 等多组，用于测试不同任务密度下的算法性能。同时为了确保实验的统计显著性，每种设置进行 50 轮独立实验，以减少随机误差。



(a) 低动态实例



(b) 高动态实例



(c) 中动态实例

图 4-4 地图规模为 32x32 且不同动态条件下的实验环境实例

为了全面评估 DE-CBS 的性能，除了算法响应时间（计算时间）和任务方案成本（路径方案质量）这两个常用评估指标之外，还设置了冲突分解频率、动态探索范围这两个观测指标。

4.3.2 实验结果与讨论

(1) 算法响应时间

算法响应时间是衡量路径规划算法性能的关键指标，直接决定了系统的实时性。特别是在高动态环境下，算法的响应速度直接影响任务的完成效率。图 4-5、图 4-6 和图 4-7 对比了 DE-CBS 和 CBS 在不同地图尺寸及动态环境下的响应时间表现，因数据是由大量重复实验所得，故使用箱线图进行展示。

可以看出 DE-CBS 作为一种迭代优化算法，其计算成本整体上会随着迭代次数的增加而呈现下降趋势，而 CBS 由于缺乏自适应优化能力，在不同轮次的运行中计算负担基本保持不变。

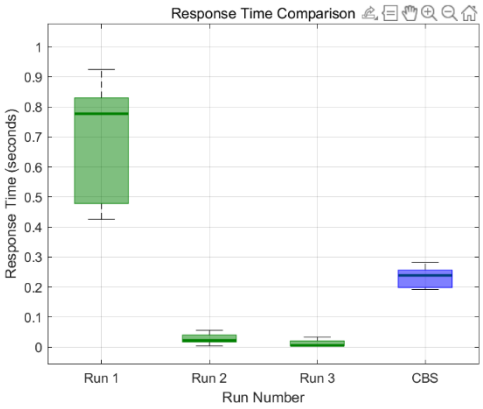


图 4-5 8x8 低动态环境算法响应时间对比

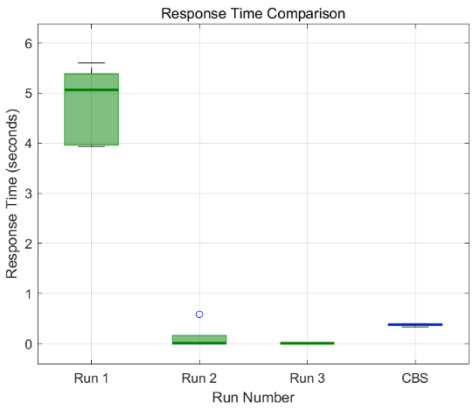


图 4-6 8x8 高动态环境算法响应时间对比

具体的，如上图所示 DE-CBS 的初始响应时间较高，在高动态环境下初始算法平均响应时间甚至在 5 秒左右，但随着迭代优化的进行，第二次执行相同或类似的任务算法的响应时间就能得到骤降，幅度可以达到 90%以上。因为地图尺寸较小，所以基本在第三轮就可以停止迭代了。

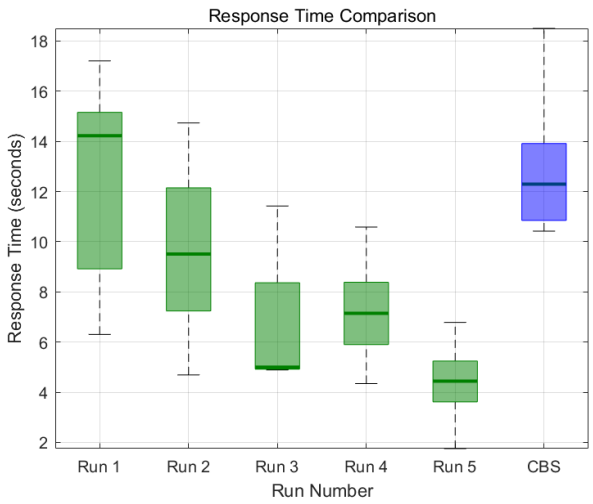


图 4-7 32x32 中动态环境算法响应时间对比

与之类似的情况在大尺寸地图中动态环境下也得到了一定的体现，如图 4-7，因为智能体任务的欧几里得距离总和随着地图尺寸的增加进行了有了一定的上升，所以 DE-CBS 算法的不同轮次优化程度相较于小尺寸地图更为平缓。同样的 DE-CBS 和 CBS 算法的计算负担也随着地图尺寸的增加而增多。甚至 DE-CBS 的初次平均响应时间高达 12 秒，表明在大地图和高动态环境下，算法需要进行

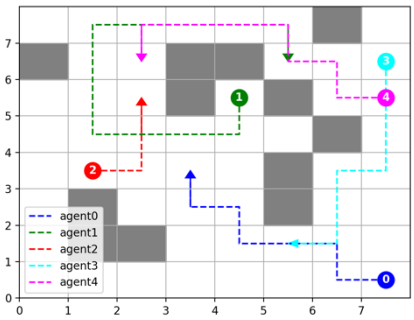
更多计算来适应环境的变化。然而，随着运行次数增加，DE-CBS 仍然展现出了较好的优化能力，迭代在第五轮基本迎来终止，平均响应时间也下降至 4.3 秒左右。这表明 DE-CBS 在该环境下的优化能力极强，初始计算成本虽然较高，但经过几次迭代后，其计算负担远低于 CBS。

表 4-2 不同条件下算法计算时间的对比

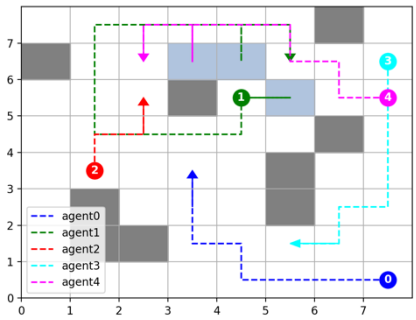
	均值(s)	最大值(s)	最小值(s)	环境动态性
CBS	0.3727	0.4003	0.3338	高动态环境 8x8 地图
DE-CBS 初始	4.7759	5.6027	3.9352	
DE-CBS 终态	0.0060	0.0066	0.0053	
CBS	0.2325	0.2829	0.1918	低动态环境 8x8 地图
DE-CBS 初始	0.6844	0.9246	0.4253	
DE-CBS 终态	0.0132	0.0334	0.0050	
CBS	12.9186	18.5002	10.4208	中动态环境 32x32 地图
DE-CBS 初始	12.3990	17.2053	6.3015	
DE-CBS 终态	4.3900	6.7849	1.7536	

由表 4-2 对数据的整体收集可以看出 DE-CBS 在所有动态环境中都表现出显著的计算时间优势，尤其是在小尺寸地图上，不管是高动态环境还是低动态环境，DE-CBS 的平均计算时间都要比传统 CBS 少 95%以上，说明其动态探索机制有效减少了路径规划的计算开销。随着地图尺寸的增大计算复杂度上升，但 DE-CBS 算法仍能比 CBS 算法少 66%左右，并且 DE-CBS 在初始状态下的平均响应时长也能够和 CBS 算法保持在同一水平了。

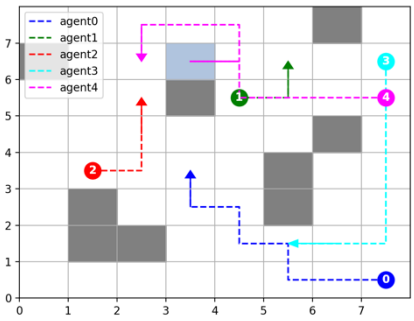
图 4-8 的几张子图分别展示了 CBS 算法的某次任务执行情况，以及 DE-CBS 算法在小尺寸地图低动态环境下的优化过程。



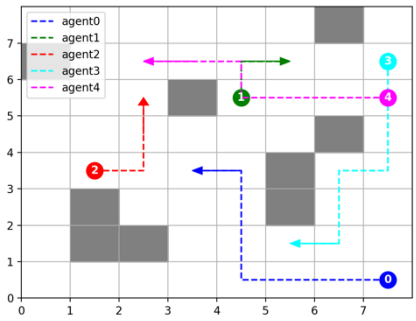
(a) CBS 算法的运行结果



(b) DE-CBS 算法初次运行结果



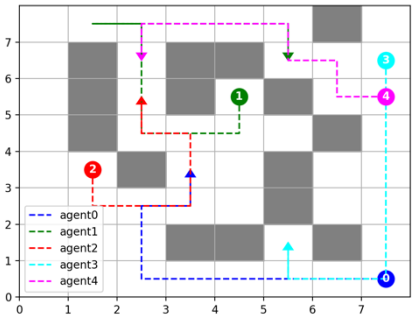
(c) 第二次任务执行情况



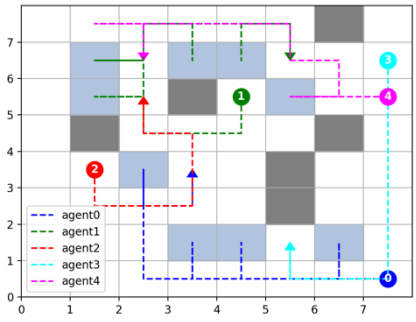
(d) 最终优化后的任务执行情况

图 4-8 低动态环境下 CBS 算法和 DE-CBS 算法的处理结果

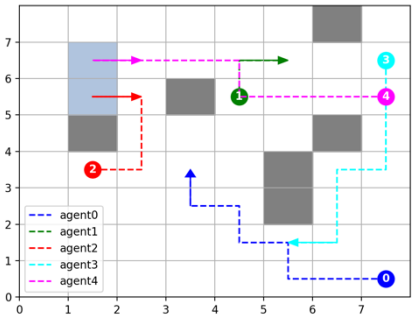
类似的，图 4-9 也展示了同样在小尺寸地图上，高动态环境下 CBS 算法和 DE-CBS 算法的某次任务执行过程。



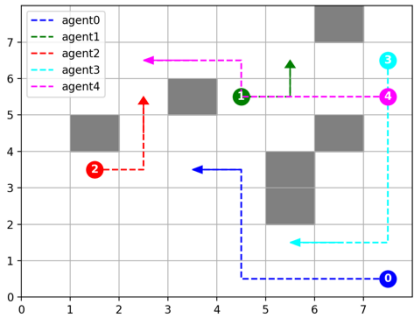
(a) CBS 算法的运行结果



(b) DE-CBS 算法初次运行结果



(c) 第二次任务执行情况



(d) 最终优化后的任务执行情况

图 4-9 高动态环境下 CBS 算法和 DE-CBS 算法的处理结果

(2) 路径质量对比

路径质量能够衡量智能体执行任务时的路径最优性，本实验通过任务路径方案的成本来映射出路径质量的高低。如下图所示：

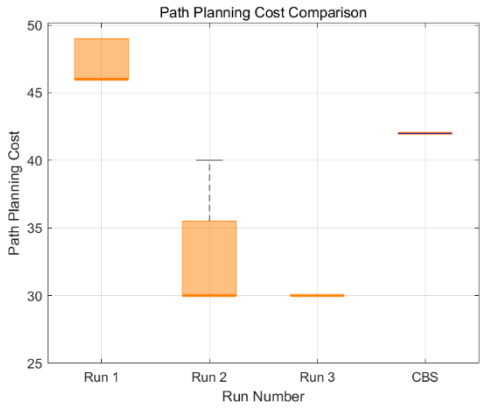


图 4-10 8x8 低动态环境路径质量对比

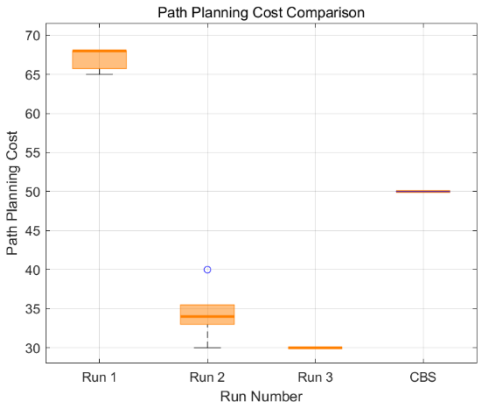


图 4-11 8x8 高动态环境路径质量对比

与算法响应时间类似，在 8x8 低动态环境地图中，DE-CBS 在初始运行时的路径规划成本较高，平均约 47。随着迭代优化的进行，在第二次执行相同或类似任务时路径规划成本显著下降至 30-40，并在第三次进一步稳定到 30，表明算法已经收敛至最优路径。相比之下，CBS 的路径规划成本在 42 左右，甚至不如 DE-CBS 在第二次优化时带来的收益高，同样在高动态环境下也是相同的情况。

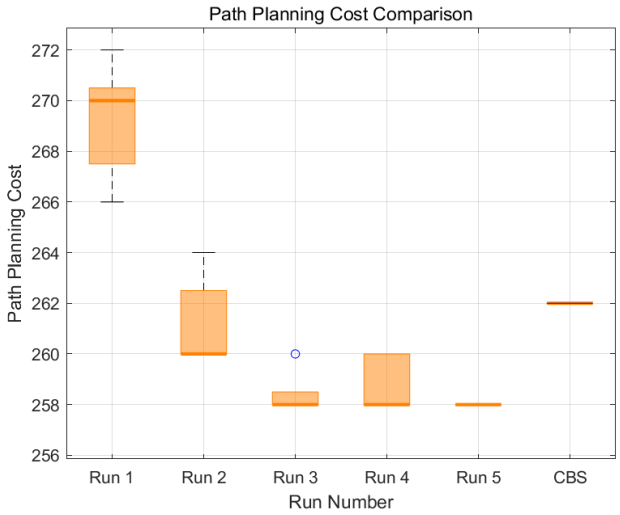


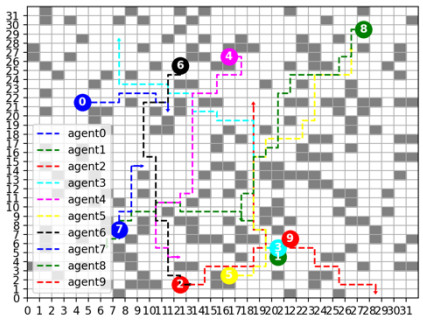
图 4-12 32x32 中动态环境路径质量对比

在 32x32 中动态环境中，DE-CBS 的初始路径规划成本进一步上升，初始算法运行所得到的任务方案平均成本达到 269.2，随后在第二次运行迅速降低至 260-264，三至五次时逐步收敛至 258。从第二次开始就已经优于 CBS 在该环境下的路径规划成本 262。

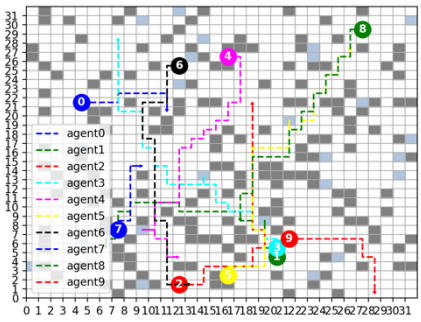
表 4-3 不同条件下任务方案成本的对比

	均值	最大值	最小值	环境动态性
CBS	50.0	50	50	高动态环境 8x8 地图
DE-CBS 初始	67.0	68	65	
DE-CBS 终态	30.0	30	30	
CBS	42.0	42	42	低动态环境 8x8 地图
DE-CBS 初始	47.2	49	46	
DE-CBS 终态	30.0	30	30	
CBS	262.0	262	262	中动态环境 32x32 地图
DE-CBS 初始	269.2	272	266	
DE-CBS 终态	258.0	258	258	

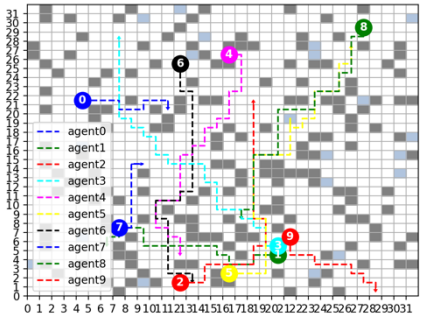
如表 4-3 所示，优化程度分别达到了 40%、28.6%以及 1.6%。中动态环境下的具体任务执行过程如图 4-13 所示。



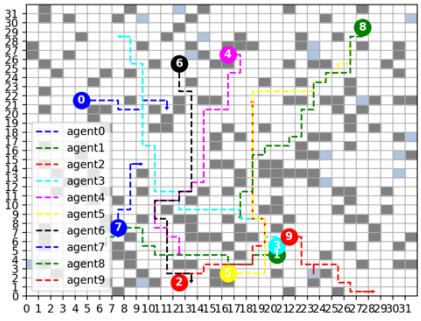
(a) 中动态环境 CBS 算法的运行结果



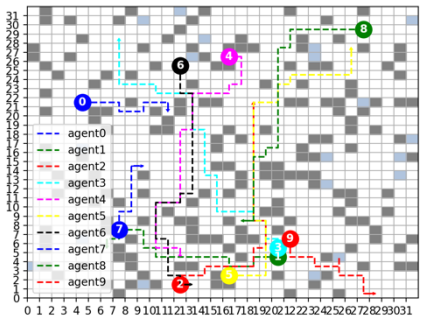
(b) 同情况 DE-CBS 算法初次运行结果



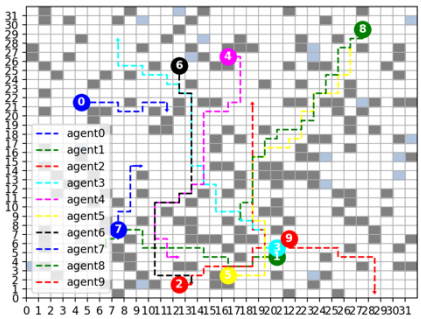
(c) 第二次相同任务的执行情况



(d) 第三次相同任务的执行情况



(e) 第四次相同任务的执行情况



(f) 最终优化后的相同任务执行情况

图 4-13 中动态环境下 CBS 算法和 DE-CBS 算法的处理结果

结果表明，DE-CBS 在不同动态环境和地图尺寸下均能优化路径规划成本，并且最终得到的路径方案，甚至优化迭代初期得到的路径方案优于 CBS。在小规模地图和低动态环境下，DE-CBS 仅需少数运行即可找到最优路径，而在高动态环境和更大规模的地图中，DE-CBS 仍然能够优化路径，但优化过程相比较之下就会稍变得平缓一些。同时在大规模地图中环境的动态性影响也会对应有所降低。

(3) 其他数据对比

除了以上两种常见的指标，本实验还对冲突分解频率以及动态探索范围进行了观测。冲突分解频率能够反映出 CBS 类的算法的搜索效率，如果冲突分解频率高，则说明当前环境对当前智能体群所要执行的任务“不友好”，会导致智能体各自的任务路径冲突频发。如果冲突分解频率较低，则说明当前环境状态对于智能体任务影响很小。

如图 4-14、图 4-15 和图 4-16 所示，分别表示在不同尺寸不同动态环境下的冲突分解频率对比数据。可以看出该指标的数据表现和算法响应时间的对比数据基本一致，但该指标更能够体现出冲突树的拓展频率。由图 4-15 中的数据可以发现，DE-CBS 在 8x8 小尺寸地图上的初次运行时间之所以异常的慢，是因为环境对当前的智能体群的任务影响过大，不确定性因素过多。

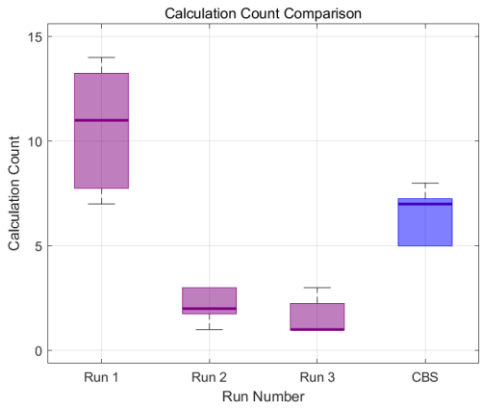


图 4-14 低动态环境下冲突分解频率对比

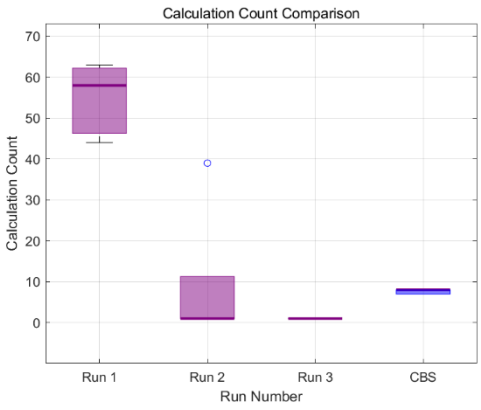


图 4-15 高动态环境下冲突分解频率对比

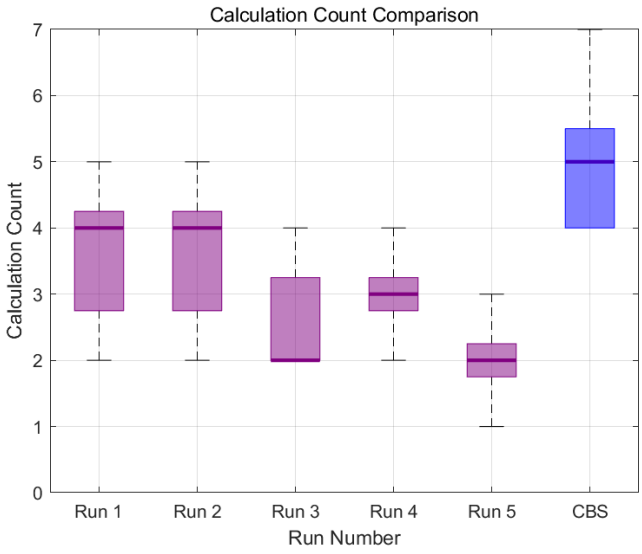


图 4-16 中动态环境下冲突分解频率对比

动态探索范围能够体现出 DE-CBS 算法的影响，如果在某次迭代优化过程中改指标较高则说明当前环境中存在着较大的可行性待定区域。随着相同或相似任务执行轮次的增加，可行性待定区的范围随着动态探索行为的一步步增加变得越来越小，对应的从初始 DE-CBS 算法执行之后的动态探索范围也逐渐减小。如图 4-17 和图 4-18 的数据显示，在小尺寸地图上基本在第三次运行时就已经收敛至最优。同样如图 4-19 所示，在大尺寸地图上也能在第五次运行达到最优。

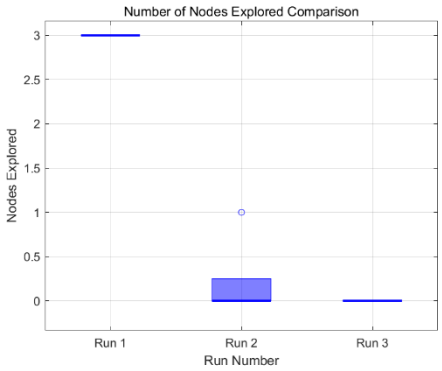


图 4-17 低动态环境下动态探索范围对比

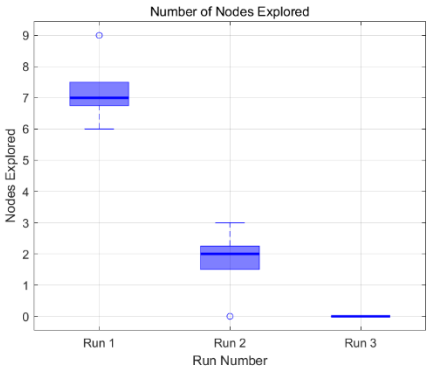


图 4-18 高动态环境下动态探索范围对比

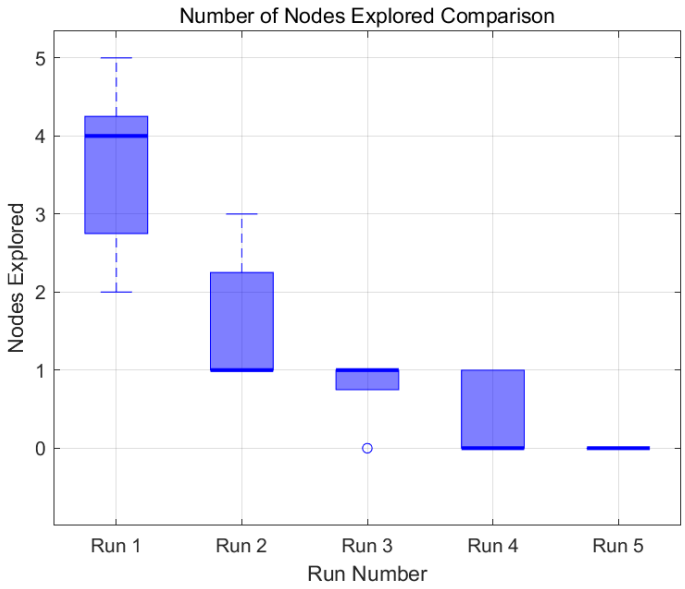


图 4-19 中动态环境下动态探索范围对比

由表 4-4 对 CBS 和 DE-CBS 算法的路径规划性能对比可以得出，DE-CBS 在所有实验环境下都能有效降低路径规划成本，高效适应动态环境并提供更高的计算速度。而且经过 DE-CBS 算法的迭代优化后，在所有环境下的冲突分解次数明显低于 CBS，使得优化后的路径规划更加稳定。

同样，在表 4-4 的最后一列也可以看出 DE-CBS 通过少量轮次的任务执行即可完成优化。在 8x8 尺寸地图中、高动态环境下，DE-CBS 仅需一到两轮优化即可收敛。32x32 尺寸地图中动态环境下，DE-CBS 需要 2 到 7 轮优化即可提供更优的路径方案，且优化稳定。

表 4-4 不同环境条件下算法的路径规划性能对比

地图尺寸	动态性比例	任务方案成本		算法计算时长(s)		冲突分解频率		动态探索范围	优化所需轮次
		CBS	DECBS	CBS	DECBS	CBS	DECBS		
8x8	20%	42	30	0.2325	0.0132	6.4	1.6	3	1
	33%	41	30	0.2506	0.0150	7.2	1.8	5-6	1-2
	43%	50	30	0.3727	0.0059	7.6	1.0	6-9	1-2
32x32	12%	262	258	12.92	4.389	5.0	2.0	6-10	2-4
	15%	262	248	11.69	4.603	4.4	3.8	7-11	3-5
	18%	266	244	12.99	2.281	4.0	2.6	12-16	5-7

4.4 本章小结

本章围绕动态环境下多智能体路径规划的适应性问题以及优化能力，提出了动态探索型冲突搜索算法。相比于 ICBS 的被动增量更新，DE-CBS 结合探索增强型A^*算法，使智能体在路径计算过程中主动探索未知区域，并基于探索信息优化路径调整策略，避免受限于静态环境信息而选择次优路径。此外，在高层搜索中，DE-CBS 改进了冲突树优化机制，引入探索路径集合和路径权重启发式策略，减少冲突树扩展带来的计算负担，并优化路径调整顺序，提高全局路径计算的稳定性和效率。同时，DE-CBS 扩展了传统 CBS 的依赖图管理机制，通过探索依赖图记录和管理智能体的探索路径，以增强路径调整的协调性和全局优化能力。

本章最后通过实验验证了 DE-CBS 在动态环境下的性能优势，并与传统 CBS 算法进行了对比分析，实验结果表明 DE-CBS 能够在动态环境下高效适应环境变化，同时保持较好的计算效率和路径质量，通过主动探索机制优化路径计算，使得智能体不仅能够基于当前环境进行路径规划，还能够结合探索信息动态调整路径，提高多智能体系统在复杂环境中的任务执行能力。

第5章 动态环境下的路规算法融合与实验

5.1 引言

在动态环境下的 MAPF 中，路径调整的计算开销和环境适应性是影响任务执行效率的关键因素。现有算法往往在计算效率和路径最优性之间存在权衡，例如 ICBS 算法通过增量式路径调整机制减少全局重规划的计算负担，但在应对未知环境时存在局限，而 DECBS 算法则通过主动探索提升路径适应能力，却可能增加计算开销和收敛时间。因此，如何结合两种方法的优势，使路径规划既能快速响应环境变化，又能主动优化未知区域，是值得研究的问题。

本章在此背景下，探讨 ICBS 和 DECBS 的融合方法，以提高算法在复杂动态环境中的适应性。通过将 ICBS 的增量式路径调整与 DECBS 的动态探索机制相结合，期望在降低计算成本的同时，提高路径规划的全局优化能力。为验证该融合算法的有效性，本章设计了一系列仿真实验（基于 DAO 地图）和实际仓储环境实验（基于无人叉车），评估其在计算效率、路径优化能力和环境适应性等方面的性能表现。

5.2 算法融合与实现

5.2.1 算法融合的目标

在动态环境下，MAPF 需要在计算效率与路径适应性之间取得平衡，以确保智能体能够在复杂、多变的环境中高效执行任务。ICBS 算法采用增量式路径调整机制，仅对受影响的路径进行局部优化，显著减少了计算开销并提升系统的实时响应能力。然而，ICBS 依赖被动更新策略，仅在检测到路径受影响后才进行调整，缺乏对环境未知区域的主动探索能力，可能导致路径次优。另一方面，DECBS 算法通过主动探索可行性待定区域，提高路径计算的全局优化能力，使智能体能够适应环境变化。然而，DECBS 在任务初期需要额外的计算开销，并可能因探索范围较大而影响计算效率。

算法 9：Hybrid Path Planning Algorithm

输入：Initial state, goal state

输出：每个智能体的最终规划结果

```
path ← icbs_planner(initial_state, goal_state) //初始化
while 存在未到达任务目标的智能体时 do
  if 环境存在变化 then
    path ← incrementally_update_path(path) 执行增量式路径更新
    for each agent_i in path do
      if 存在智能体agent_i位于可行性待定区域附近 then
        path ← dynamic_exploration(agent_i, path) //动态探索可行性待定区域
      end
    end
  end
  if detect_path_conflict(path) then //检查并存在路径冲突
    path ← resolve_conflict(path) //解决冲突
  end
end
return path
path ← optimize_path(path) //优化下次任务路径方案
```

为取两者所长，本章提出了一种融合 ICBS 和 DECBS 优点的混合路径规划算法。在该算法中，ICBS 的增量式路径调整机制作为主导框架，在环境发生变化时，对受影响的路径进行局部优化，以减少计算负担。而当环境变化较复杂或存在未探测区域时，结合 DECBS 的动态探索机制，主动收集环境信息，优化路径规划，使智能体能够更好地适应动态障碍物与环境不确定性。通过两种方法的互补融合，使得算法在面对高度动态环境时，既能快速调整路径，又能优化全局规划效果，提高智能体的任务完成效率。

5.2.2 算法融合的实现

融合后的算法结合了增量式路径调整和动态探索的优点，具体而言，融合算法的工作机制包括以下几个关键点：

(1) ICBS 增量式路径调整为主导框架：融合算法的核心框架由 ICBS 的增量式路径调整机制主导，增量式路径调整机制负责根据环境变化对智能体的路径进行局部更新。当环境变化较小或发生短期变化时，增量式路径调整能够高效地更新路径，避免重新规划整个路径，从而节省计算资源并确保多智能体的实时响应。

(2) DECBS 的动态探索机制作为补充：在 ICBS 主导的增量式路径调整过程中，如果路径经过 DECBS 算法标记的可行性待定区域，则会启动动态探索机制。探索过程通过采集新的环境数据、重新评估路径可行性、及时发现新的障碍物或空旷区域，刷新可行性待定区域的信息，从而优化路径。

(3) 实时反馈与迭代优化：在路径调整和动态探索的过程中，系统不断将新的环境数据反馈到路径规划模块，并通过算法的迭代优化，逐步改进路径方案。最终，随着任务轮次的逐渐增加，算法对环境的认知更加全面，规划所得的路径方案趋于最优。

5.3 实验环境与平台配置

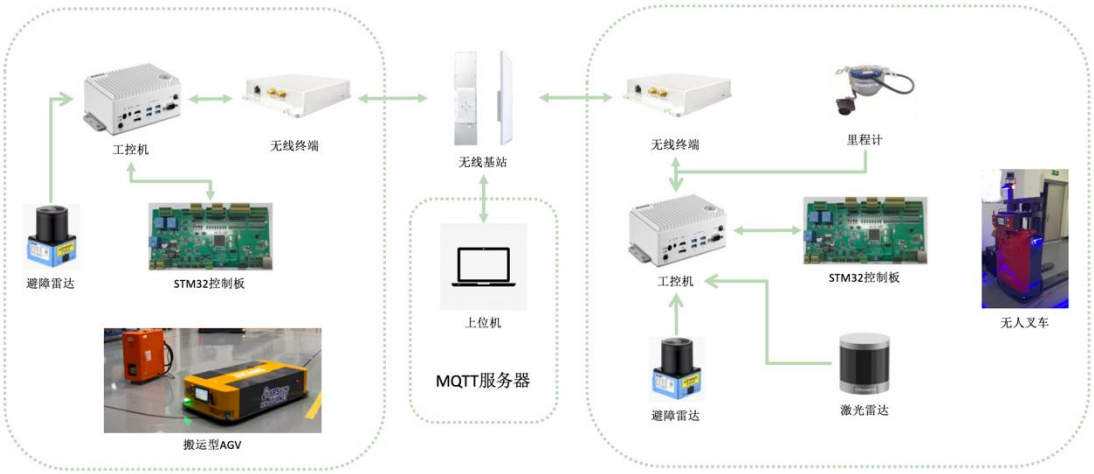


图 5-1 实验平台架构示意

上图展示了多智能体系统实验平台的硬件结构与软件通信架构。仓储实验部分将基于前文所提出的 ICBS 和 DE-CBS 算法框架，ICBS 应对动态变化路径调整，DE-CBS 用于路径主动探索，两者融合使用以提升路径规划的稳定性、计算效率与环境适应性，验证算法在实际场景中的工程可行性与优势。

5.3.1 硬件环境

本实验平台采用高精度传感器与嵌入式计算设备相结合的方式，构建了完整的多智能体路径规划系统。实验环境中的核心硬件设备包括计算设备、AGV 硬件、传感器模块、控制模块、安全模块、电源模块以及无线通信设备，以保证系统在复杂动态环境下的高效运行和稳定通信。

首先，实验采用一主一备用两台计算机作为上位机，统一安装相同的系统以及编程环境，主要用于运行 ROS、执行路径规划算法、处理传感器数据以及管理任务调度。两台计算设备的详细配置如表 5-1 所示：

表 5-1 计算设备具体参数

设备型号	处理器	内存	操作系统	其他配置
设备 1	Intel Core i5 (2.3 GHz 四核)	8 GB LPDDR3	Ubuntu 18.04	Intel Iris Plus Graphics 655，磁盘空间 256 GB
设备 2	Intel Core i7 (12th Gen, 2.3 GHz)	16 GB	Ubuntu 18.04	64 位操作系统，磁盘空间 256 GB

这两台计算设备分别承担路径规划计算、任务调度、通信管理以及实验数据的存储和处理，支持多线程并行计算，以提高路径规划算法的运行效率。

实验平台的多智能体系统基于移动机器人架构，采用激光雷达+轮式里程计进行组合定位，并配备控制模块、电源模块以及避障安全模块，以确保机器人在动态环境中的稳定运行。

在定位模块，AGV 采用速腾 16 线激光雷达（如图 5-2 所示），具有高精度、全方位感知的能力，能够实时获取周围环境信息。考虑到激光雷达采样信号存在一定的时间间隔，为提高定位精度，实验平台还搭配了轮式里程计（如图 5-3 所示）进行补偿定位，结合两者的数据可实现高精度的机器人位姿估计。具体参数如表 5-2 所示。

图 5-2 速腾 16 线激光雷达

图 5-3 轮式里程计

图 5-4 避障雷达

控制模块主要用于调度各个传感器和执行器的运行状态，确保机器人按照规划路径执行任务。本实验的控制模块由 STM32 控制板和 EI-52 工控机组成。STM32 控制板负责机器人底层运动控制，通过 RS485 协议与其他传感器进行通信，保证系统的低延迟性。EI-52 工控机作为 AGV 的工控机，负责路径规划计算、传感器数据处理与任务分配，并支持 TCP/IP 与 UDP 通信协议。

表 5-2 硬件参数表

设备名称	主要参数
激光雷达	16 线，通信协议 TCP/IP，最大测距 150m，角分辨率 0.1°，扫描角度 360°，帧率 5Hz/10Hz/20Hz
轮式里程计	17 位分辨率，绝对信号频率 1/100s，供电电流 110mA
STM32 控制板	通信协议 RS485，线性精度+0.05%FS，分辨率 17 位，量程 4000mm
EI-52 工控机	通信协议 TCP/IP、UDP、RS485，运行内存 8GB，存储空间 256GB，功率 90W
激光避障雷达	通信协议 TCP/IP、UDP/IP，扫描半径 0.05m-5m，扫描角度 270°，分辨率 1mm，角分辨率 0.5°，工作频率 20Hz
锂电池	输出电压 48V，支持 RS485 和 CAN 协议
无线终端	频段 5.8G，支持 802.11n，最大输出功率 30dBm，数据传输速率 300Mbps
无线基站	频段 5.8G/2.4GHz，最大传输速率 600Mbps，支持 AP 模式

在安全与能源模块，每一个 AGV 都配备有一套激光避障雷达，用于检测前方的障碍物，并在危险情况下采取避障措施。避障雷达采用 TCP/IP 和 UDP/IP 通信协议，扫描半径可达 5 米，分辨率 1mm，确保机器人能够准确检测障碍物并执行相应的规避动作。同时 AGV 设备使用锂电池作为主要供电设备，支持 RS485

和 CAN 通信协议，可以通过电路板进行升降压调节，以满足不同硬件设备的电压需求。

为了保证多智能体系统的稳定通信，实验平台采用了无线 AP 基站和无线终端设备，搭建了独立的无线网络环境。无线通信模块的主要任务包括任务调度、路径更新以及机器人状态信息的同步。具体参数如表 5-2 所示。实验所使用的上位机以及 AGV 设备上装备的工控机均采用 MQTT 协议进行多智能体之间的通信，通过 EMQX 5.3.2 服务器实现机器人与上位机的实时信息交互。

5.3.2 软件环境

实验设备的操作系统为 Ubuntu 系统 18.04 版本，并选用与其对应的机器人操作系统 ROS Melodic 版本。ROS 作为一个开源代码的机器人操作系统，为多智能体系统的路径规划、运动控制、传感器数据处理、通讯管理等提供了强大的功能支持。

为了实现机器人之间的高效通信，实验采用了 EMQX 5.3.2 版本的 MQTT 服务器。MQTT 协议采用发布/订阅模式，适用于低带宽、高延迟的网络环境。在本实验中，EMQX 服务器负责多智能体之间的实时任务分配、路径更新和状态同步。机器人通过 MQTT 协议与服务器通信，确保路径规划结果及时传递给每个机器人，并协调各智能体之间的任务执行。

除了 ROS 和 MQTT，实验还使用了 Gazebo 和 Rviz 作为仿真与可视化工具。Gazebo 用于构建虚拟环境并进行路径规划实验，Rviz 则用于实时展示机器人路径和规划结果。在仿真过程中，仿真地图、障碍物设置和机器人状态信息将在 Rviz 中实时显示，方便调试与结果验证。

在本实验环节搭建的多智能体系统中，上位机使用融合后的算法为不同类型的 AGV 分配不同的任务，生成无冲突的多智能体路径规划方案，并通过 MQTT 服务器下发给响应 AGV。各个 AGV 再通过自身的感知数据和路径方案，独立执行任务。这种中心化的任务调度方式有效的提高了系统的执行效率，尽可能避免了因环境的变化或不同设备之间的任务冲突所导致的异常。

5.4 基于 DAO 地图的仿真实验

图 5-5 lak303d 194x194

图 5-6 den520d 256x257

为了验证所提出融合算法在动态环境中的有效性，本实验基于 DAO（Dragon Age Origins）地图进行仿真实验，选取地图编号为 den520d 和 lak303d，尺寸分别为 256x257、194x194 的网格地图（如图 5-5、图 5-6 所示）。DAO 地图是学术界常用的标准测试环境之一，其特点是复杂的拓扑结构、多种障碍物分布模式，以及适用于多智能体路径规划（MAPF）任务的测试。在此特别鸣谢移动 AI 实验室所提供的地图文件资源[6]。实验旨在模拟真实的动态环境，考察融合算法在应对突发障碍物、路径质量、任务执行效率等方面的表现，将不同优化轮次的融合算法与 ICBS 进行对比分析。

实验步骤如下：首先，在 DAO 地图上初始化智能体的起点与目标点，每个智能体的路径由不同算法规划生成，初始环境中的障碍物为静态分布。随后，在智能体行进过程中，模拟动态障碍物的加入与移动，以测试各算法的适应性。障碍物的变化方式包括随机移动（如沿既定路径移动）和突发加入（如某个区域突然变为不可通行）。实验在多组多个个智能体的情况下进行测试，评估不同规模下算法的表现。

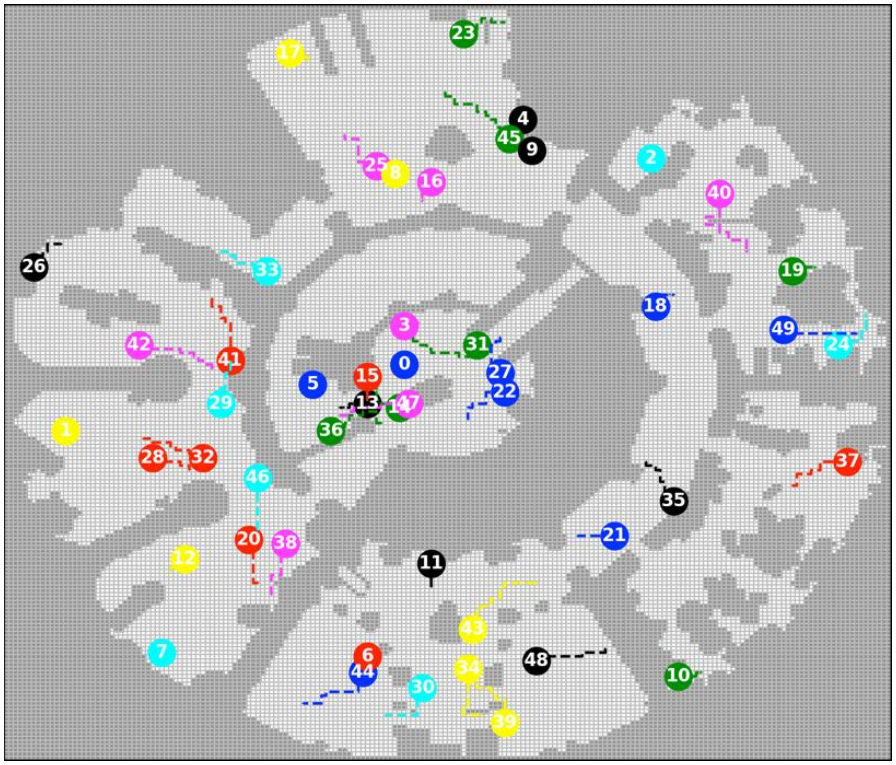


图 5-7 ICBS 遇到突发的障碍物阻挡时进行的路径调整

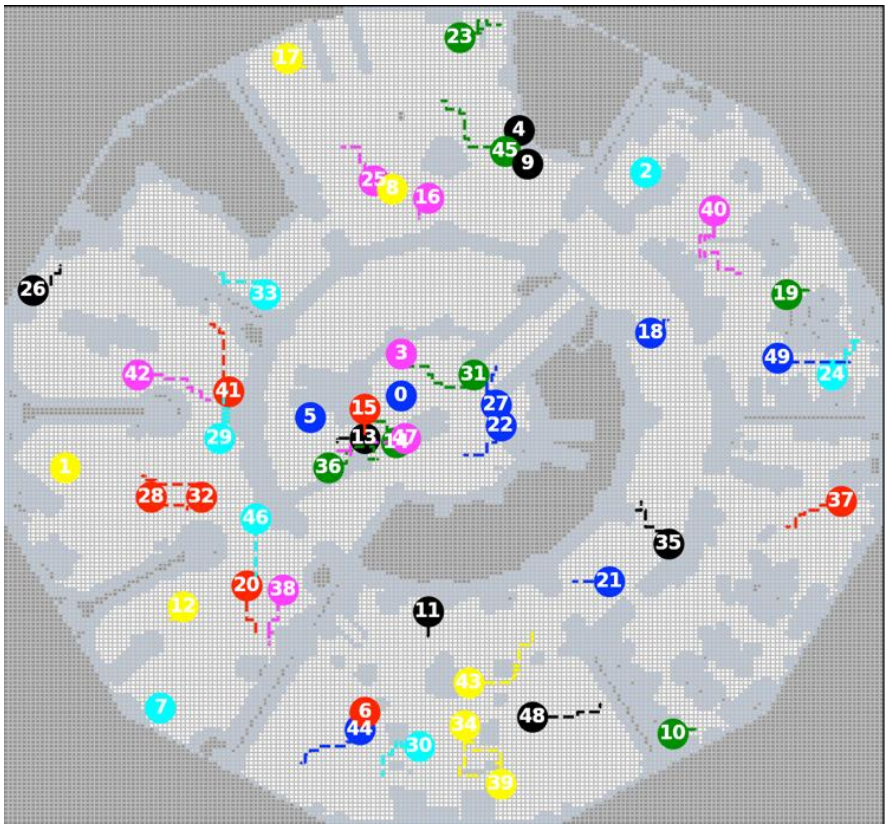


图 5-8 融合后算法在初始情况进行一定的探索行为

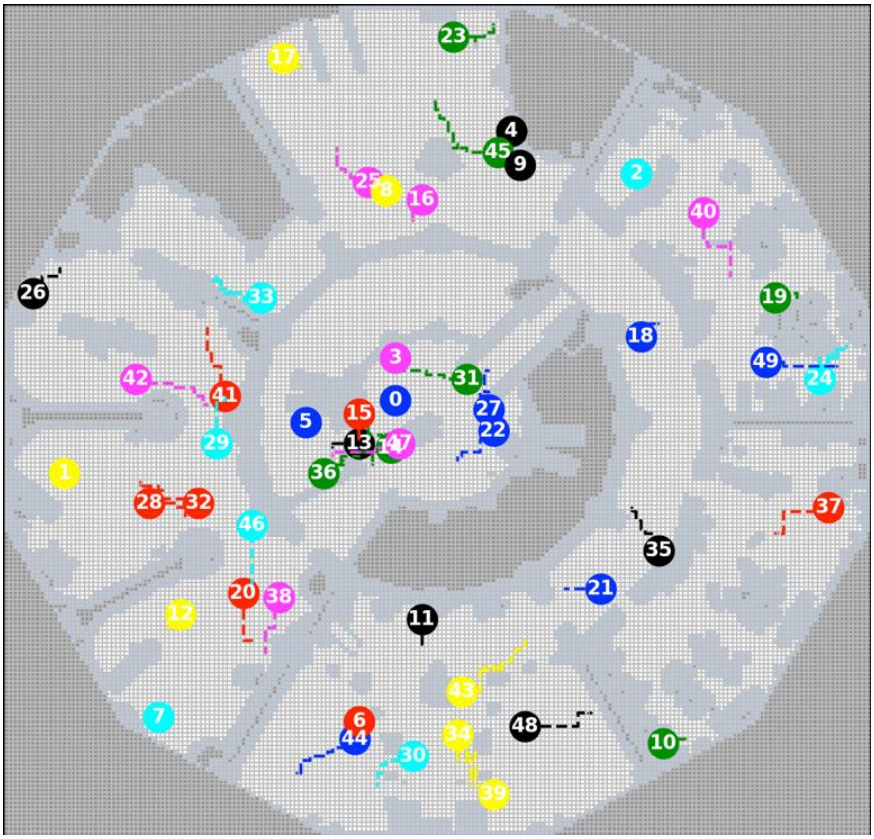


图 5-9 部分环境被勘探后，优化后的任务执行情况

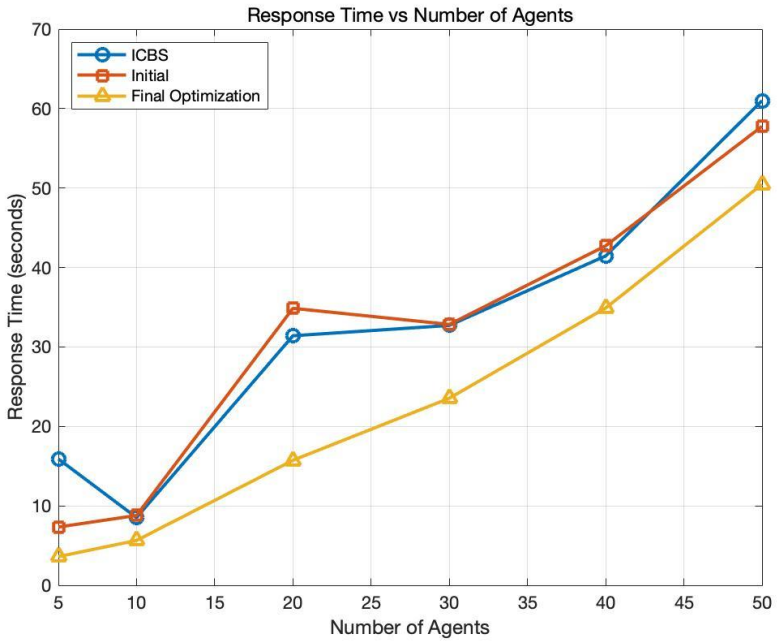


图 5-10 算法平均响应时间

在计算效率方面，图 5-10 显示了不同算法在不同智能体数量下的平均响应时间变化趋势。从表中数据可以看出，ICBS 在智能体数量较少时（10 个以下）计算开销相对较低，但随着智能体数量的增加，其计算时间呈现指数级增长，特

别是在 50 个智能体时，计算时间高达 61，这表明其扩展性较差。与其相比较，融合后的算法在初始环境未进行任何探索时，部分任务规模下的计算时间略高于 ICBS。例如在 20 个智能体情况下，其平均响应时间达到 34.87，略高于 ICBS 的 31.41，说明优化过程初期会增加一定的计算复杂度。然而，在优化轮次增加后，最终融合后算法的计算时间大幅减少，比如同样是在 20 个智能体情况下，其计算时间仅 15.73，相比 ICBS 降低了 49.9%，相比融合算法在初始情况下降低了 54.9%。这一趋势在 30、40、50 个智能体任务中也保持一致，说明融合后的算法在处理大规模任务时具有明显的计算效率优势。

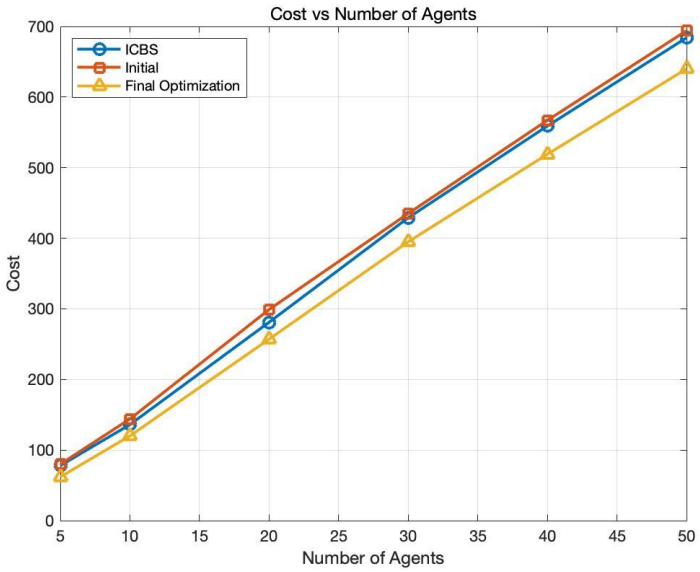


图 5-11 路径方案平均成本

在路径规划质量方面，图 5-11 反映了不同算法的路径成本随智能体数量变化的趋势。在每组智能体任务的欧几里得距离之和相仿的条件下，ICBS 的路径成本随智能体数量的增加呈线性增长，例如在 10、20、30 个智能体情况下，其路径成本分别为 136、281、429。融合算法在初始情况下的路径成本在部分任务规模下同平均响应时间一样会略高于 ICBS，例如在 20 个智能体情况下，其路径成本为 299，比 ICBS 高 6.4%，这是由于融合后算法在初始情况下需要对环境进行探索认知，导致路径方案需要产生一定的额外开销。而最终算法的路径成本则在所有任务规模下均显著降低，例如在 50 个智能体情况下，其路径成本为 640，相比 ICBS 的 684 降低了 6.4%，说明融合后的算法在优化迭代次数增加后，使得路径选择更加合理，从而减少了行驶成本。

表 5-3 算法数据对比（lak303d）

数量	响应时间(s)			方案成本		
	ICBS	Initial	Final Optimization	ICBS	Initial	Final Optimization
5	15.90	7.33	3.61	78	80	62.56
10	8.53	8.79	5.64	136	144	120.08
20	31.40	34.86	15.73	281	299	257.89
30	32.70	32.83	23.54	429	435	395.86
40	41.45	42.73	34.90	559	567	519.45
50	61.00	57.76	50.46	684	694	640.69

整体来看，实验结果表明，最终本章提出的方法在计算效率和路径质量上均优于 ICBS，特别是在大规模任务环境下，优化迭代次数的增加有效降低了计算时间，并优化了路径选择，提高了整体任务执行效率。这一现象同样能在 DAO 地图 den520d 上体现，如图 5-12、图 5-13 所示。

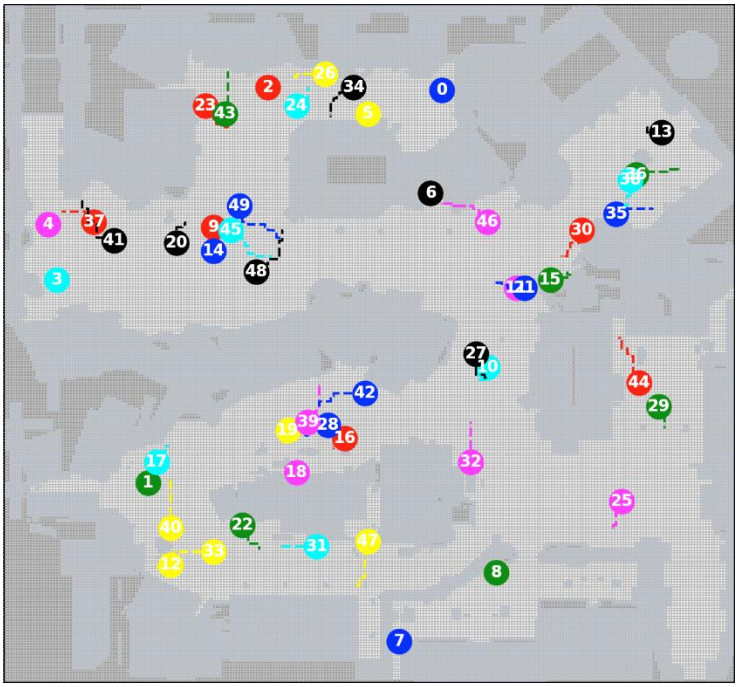


图 5-12 地图 den520d 上初次执行融合后算法（较理想条件下的样例）

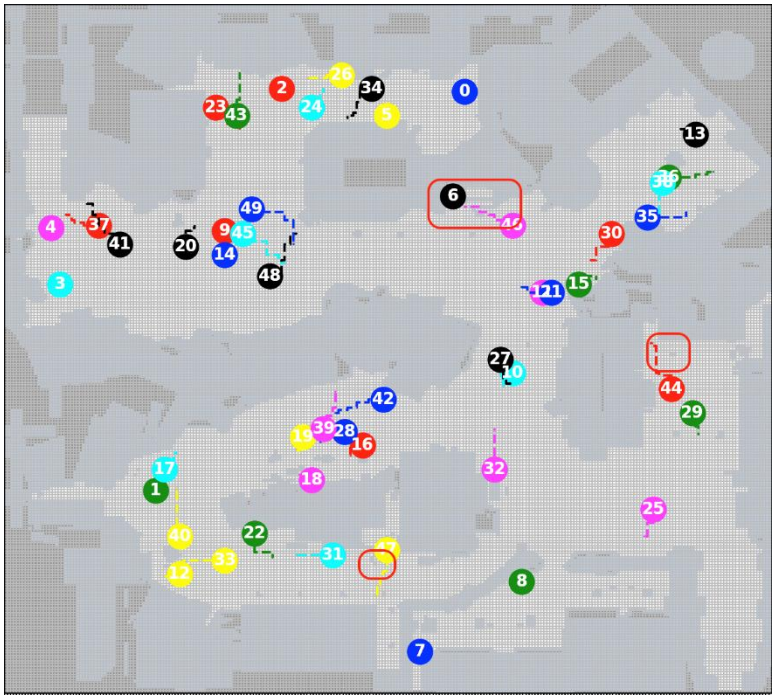


图 5-13 地图 den520d 上经过融合算法迭代优化后的任务执行情况

5.5 仓储环境下的无人叉车实验

为了进一步验证所提出的融合算法在实际场景中的可行性，本节基于仓储环境进行无人叉车实验，模拟多智能体在复杂动态环境下的路径规划、避障及任务执行过程。该环境包含多个存储区域、货架、通道（如图 5-14 所示），以及用于模拟动态障碍物的临时堆放区。实验主要依托两台无人叉车（如图 5-1 所示），并依托 ROS 进行任务调度，同时使用 MQTT 通信协议进行数据同步和任务下发。激光雷达用于实时环境感知，确保无人叉车在运行过程中能够自主避障并调整路径。任务规划过程中，ICBS 作为增量路径调整主导，而 DECBS 在路径执行过程中进行动态探索，从而优化路径方案。

图 5-14 实验室仓储环境

实验开始前，需要对仓储环境进行扫描建图。首先无人叉车需要在实验室中模拟的仓储环境内绕行一圈，使用 SLAM 记录环境点云数据，通过回环检测方法识别相同区域，修正定位误差，提高建图精度。构建完成点云地图如图 5-15 所示，紧接着需要将其转化为栅格地图的格式以适用于算法所需要的环境信息输入。如图所示，PGM 图片上的灰色部分表示实验室场景内的障碍物信息，并且根据数据的密度来划分到对应栅格地图上的黑色静态障碍物区域以及灰色可行性待定区域，如图 5-17 所示。

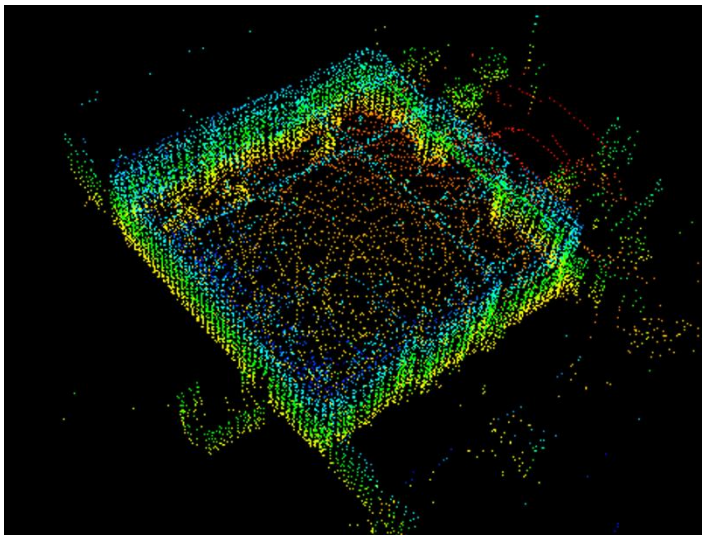


图 5-15 实验室仓储环境的点云地图

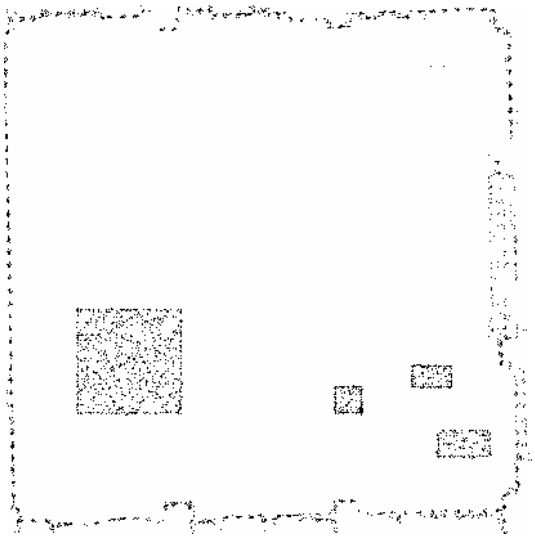


图 5-16 占据栅格地图

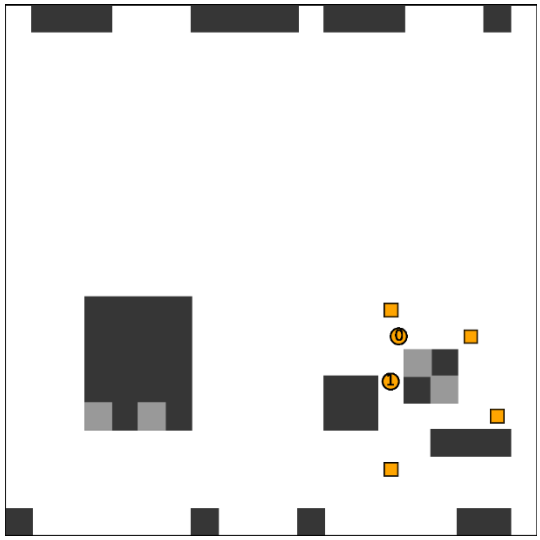


图 5-17 栅格地图

建图完成后，系统根据特定任务需求生成多个无人叉车任务。每个任务包含叉车的起点、目标点以及任务类型，并通过 MQTT 服务器发布至相应的话题（topic），如图 5-18 所示。无人叉车的上装载的工控机会订阅自身对应的任务，并基于融合后算法进行路径规划，确保所有叉车的路径不会相互冲突。在路径执行过程中，叉车沿着规划路径行驶，同时不断感知环境，并根据实时情况调整路径。

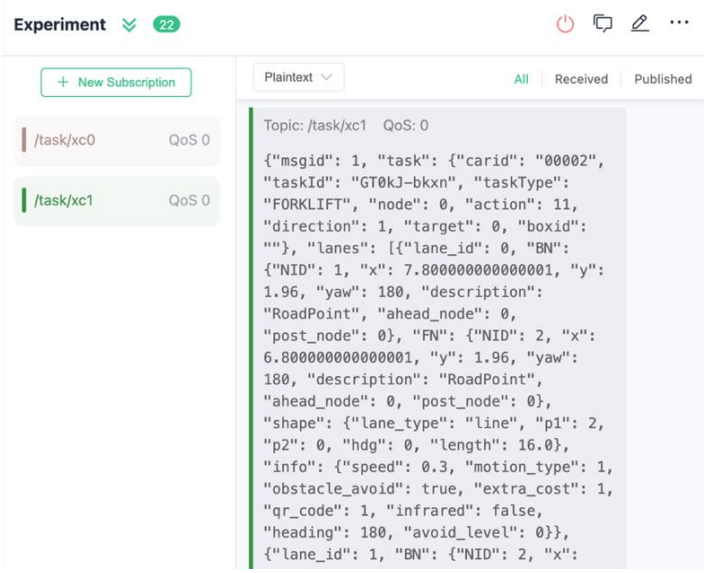


图 5-18 上位机通过 MQTT 服务器下发具体任务

在任务执行阶段，系统需要处理可能出现的动态障碍物，如其他叉车占道、货物临时堆放等。此时，ICBS 增量式路径调整机制会实时调整路径，使叉车能够绕开障碍物并继续执行任务。若路径不可行，系统将触发全局路径重新规划。

同时，为了优化未来任务的路径规划，DECBS 机制会对路径经过的区域进行探索，记录哪些区域存在高概率的动态障碍，并反馈给任务调度系统，以在后续路径规划中规避不稳定区域，如图 5-19 所示。

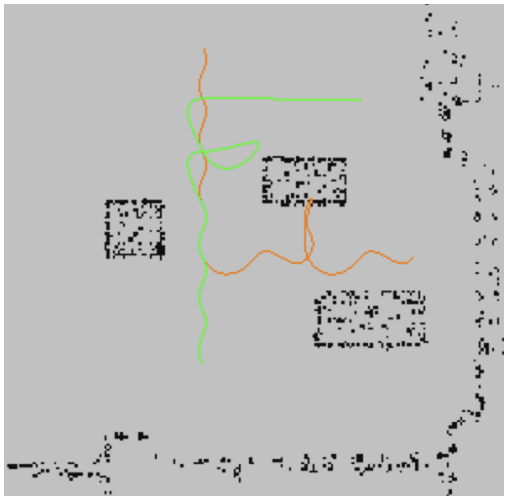


图 5-19 初始时 AGV 的任务路径

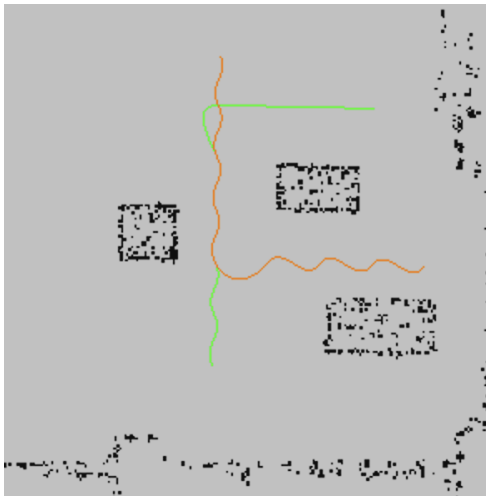


图 5-20 优化后的任务路径

由于融合算法需要通过多次执行任务来优化对环境的认知，实验将继续进行优化后的任务执行。在优化阶段，DECBS 依据历史任务数据，对可行性待定区域进行进一步优化，使路径选择更加合理。无人叉车以及搬运型 AGV 的任务执行情况都会被系统记录，迭代优化后的系统输入环境在执行相同或类似的任务时，可以明显得到改善，如图 5-20 所示。

上图所示过程对部分环境进行了一定的探索与更新，但是任务对于 AGV 设备来说并不是每一次的任务都是完全相同的，对于环境图的更新也不是一次任务执行就能结束的。故还需要在实验场地上随机设置不同的任务，并且还需要将环境中的障碍物进行动态的改变。如图 5-21 就是在执行完图 5-17 所示的任务后，执行其他的任务。从图中环境图所记录的环境信息可以看出，经过上一次的任务后，多智能体系统对环境的记录刷新了（黑色、灰色方块数量发生变化）。

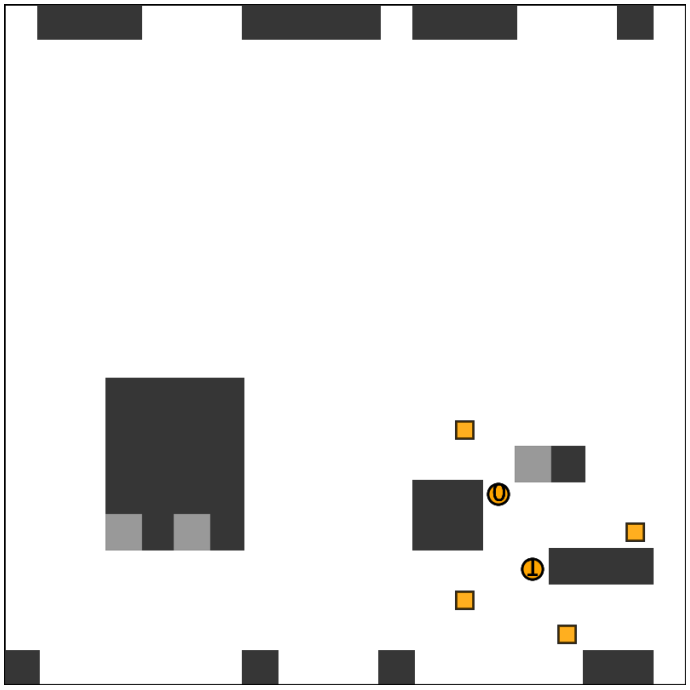


图 5-21 同实验场景下执行其他任务

如下图所示的结果分别是使用常规 CBS 算法在实验环境下的任务执行结果（如图 5-22 所示）以及算法融合后，经过一次对环境的优化后再次执行相同任务的执行结果（如图 5-23 所示）。可以明显看出融合算法能够有效的降低路径方案成本。

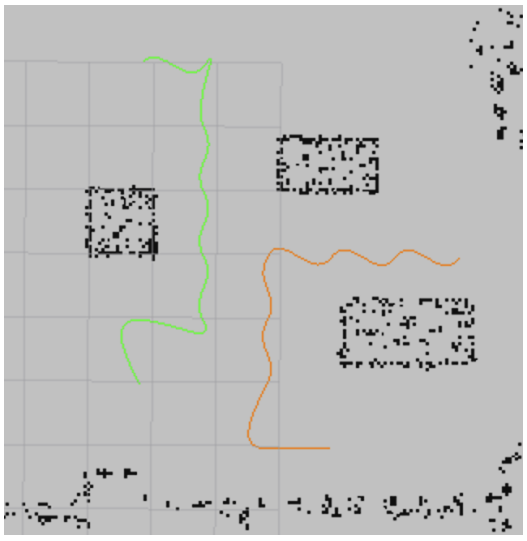


图 5-22 CBS 算法的执行结果

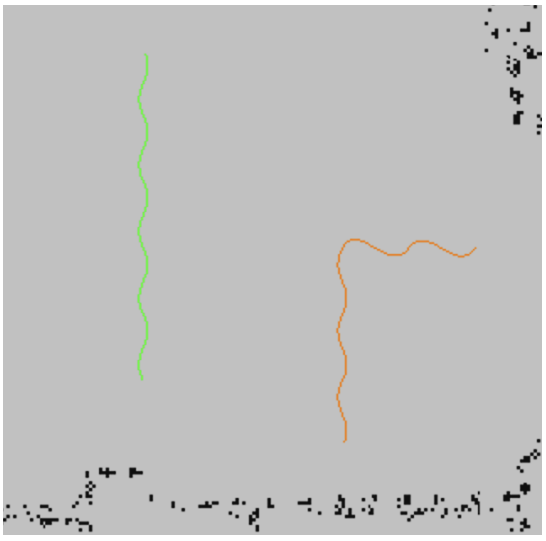


图 5-23 算法融合后的执行结果

在实验场地上进行诸如此类的多次任务之后收集到的数据如下表 2-1 所示：

表 5-4 融合后算法与 CBS 算法在随机任务下的路径成本成本对比

	CBS			融合后算法		
	AGV1	AGV2	总成本	AGV1	AGV2	总成本
Task1	7.95	7.73	15.68	5.15	5.34	10.49
Task2	11.10	11.17	22.27	8.40	8.74	17.14
Task3	10.42	14.66	25.08	7.81	8.47	16.28
Task4	13.16	5.09	18.25	10.70	4.35	15.05

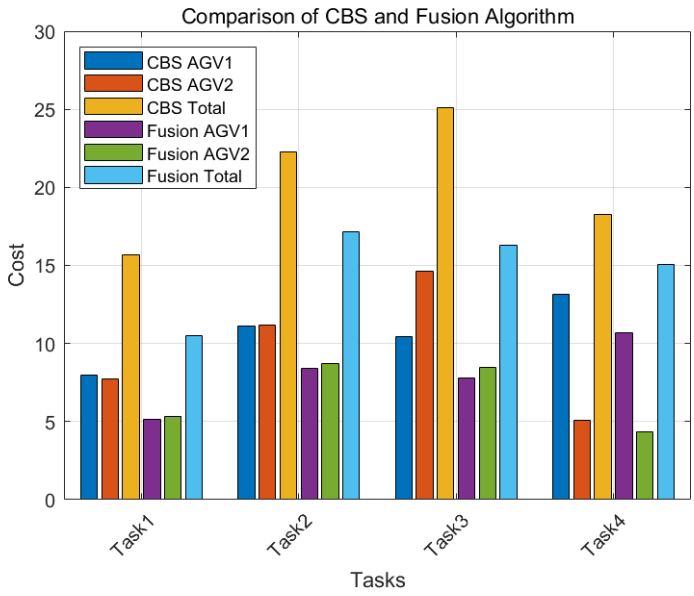


图 5-24 融合后算法与 CBS 算法在随机任务下的路径成本成本对比

数据比较了 CBS 算法和融合后算法在四个随机任务下的 AGV1、AGV2 以及总成本。总体来看，融合后算法在所有任务中都表现出更低的总成本，表明其优化效果优于 CBS 算法。从单个 AGV 的角度，融合后算法在 AGV1 和 AGV2 的成本方面也比 CBS 低，说明算法的调度效率更高，减少了资源占用。尤其在 Task3 和 Task4，CBS 的总成本显著高于融合后算法，说明在复杂任务下，融合后算法的优化优势更加明显。因此，从整体成本控制和 AGV 调度效率来看，融合后算法比 CBS 算法更具优势。

5.6 本章小结

本章研究了 ICBS 与 DECBS 的融合算法，旨在提高多智能体路径规划在动态环境中的计算效率和适应性。融合算法以 ICBS 的增量式路径调整为核心，确保路径局部优化的高效性，同时结合 DECBS 的动态探索机制，主动感知环境中

的未知区域，以优化路径规划过程。该方法在环境变化较小时采用增量调整降低计算开销，而在复杂环境下引入探索策略，确保路径方案的全局最优性。

在 DAO 地图仿真实验与仓储环境无人叉车实验中，融合算法展现出良好的计算效率、路径优化能力及环境适应性。实验结果表明，与单独使用 ICBS 或 DECBS 相比，融合算法在不同规模的动态环境下均能有效降低计算时间（最高减少 50%以上），并优化路径成本（降低约 6-10%），尤其在大规模任务环境中，其优化效果更为显著。此外，仓储实验进一步验证了融合算法在复杂实际场景中的可行性，证明其能够有效应对突发障碍物、路径冲突及任务执行不确定性，为智能调度和自主导航提供了一种高效、稳定的解决方案。

第6章 总结与展望

6.1 总结

本研究针对动态环境下的 MAPF 问题，在传统冲突搜索算法的基础上，提出了增量式冲突搜索算法和动态探索型冲突搜索算法，以期提升路径规划的计算效率与环境适应性。本文的研究内容涵盖路径优化算法的改进设计、动态环境自适应机制的构建、实验平台的搭建以及算法的性能验证，旨在提升并验证多智能体系统在复杂环境中的任务执行效率和扩展性。

在算法创新方面，ICBS 提出了一种增量式路径调整机制，通过环境监测、增量更新和冲突解构，实现对受环境变化影响路径的局部优化，避免传统 CBS 在动态环境中因全局重规划带来的计算冗余，提高了系统的实时性和可扩展性。而 DE-CBS 则在 CBS 框架下引入探索增强型 A*算法，使智能体能够在路径计算过程中主动识别并探索未知区域，结合探索因子和探索依赖图进行路径优化，提高路径的全局适应性和规划质量。两种改进策略在应对动态环境变化时各有侧重，ICBS 侧重于计算效率，减少环境变化对任务执行的影响，而 DE-CBS 通过主动探索提升路径计算的全局最优性。

在实验评估方面，本文在不同尺寸与智能体规模的随机仿真地图、标准 DAO 地图仿真环境和仓储无人叉车实验中，对所提出的 ICBS 和 DE-CBS 算法进行了系统验证。实验结果表明，相较于传统 CBS 算法，ICBS 减少了约 20%-40%的路径规划计算开销，计算时间降低了 20%-50%，有效提升了系统对动态环境变化的响应速度。而 DE-CBS 的主动探索策略降低了路径冲突率，使任务完成时间缩短了 60%以上，特别是在环境复杂度较高的情况下，其全局优化能力更为显著。此外，融合两种改进方法后，实验进一步表明该混合算法在大规模多智能体任务中具有更好的可扩展性和环境适应性，能够在任务执行过程中不断优化路径，为智能调度和自动化导航提供了一种高效、稳定的解决方案。

6.2 展望

尽管本文提出的 ICBS 和 DE-CBS 算法在动态环境下取得了较优的性能，但仍存在一定的局限性。首先，增量式更新策略在高动态环境中仍然可能遇到路径

反复调整的问题，影响任务的执行稳定性。其次，DE-CBS 虽然在复杂环境下具备良好的全局探索能力，但当环境的动态性极高（如频繁变化的障碍物）时，可能会导致探索路径的计算成本上升。此外，本文的实验主要基于网格地图和仓储环境，未来需要进一步扩展至更复杂的动态场景，如智能交通、无人机编队和机器人协作系统，以验证算法的通用性和适应性。

基于上述研究成果和局限性，未来的研究方向可以从以下几个方面展开：

(1) 引入智能学习机制：结合强化学习、图神经网络等方法，实现路径调整策略的在线学习与自适应优化，进一步提升系统在复杂未知环境中的自主决策能力。

(2) 拓展至多场景协作应用：将算法推广应用于无人车交通管控、机器人协作搬运、无人机编队导航等复杂异构多智能体系统中，研究跨系统之间的路径协同与任务分配机制，以提升算法的通用性与工程实用性。

综上所述，本文的研究为多智能体路径规划在动态环境下的优化提供了一种可行的解决方案，并在实验中验证了其有效性。未来，希望通过更深入的优化和扩展，使智能体路径规划在实际应用中具备更强的智能化、适应性和高效性，为自动化调度、智能交通、机器人导航等领域的发展提供更加先进的算法支持。

参考文献

[1] Dobrev Y, Vossiek M, Christmann M, et al. Steady delivery: Wireless local positioning systems for tracking and autonomous navigation of transport vehicles and mobile robots[J]. IEEE Microwave Magazine, 2017, 18(6): 26-37.

[2] Sartoretti G, Kerr J, Shi Y, et al. PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning[J]. IEEE Robotics and Automation Letters, 2019, 4(3): 2378-2385.

[3] Wurman P R, D'Andrea R, Mountz M. Coordinating hundreds of cooperatives, autonomous vehicles in warehouses[C]//IAAI'07: Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2. Vancouver: AAAI Press, 2007: 1752-1759.

[4] Silver D. Cooperative pathfinding[C]//Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2005, 1(1): 117-122.

[5] Dresner K, Stone P. A multiagent approach to autonomous intersection management[J]. Journal of Artificial Intelligence Research, 2008, 31: 591-656.

[6] Stern R, Sturtevant N R, Felner A, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks[J]. Symposium on Combinatorial Search (SoCS), 2019: 151-158.

[7] 舒翔翔. 多机器人最优路径规划研究[J]. 信息与电脑(理论版), 2017, (15): 62-64.

[8] YU J, LAVALLE S. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs [J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2013, 27 (1):1443-1449.

[9] 褚晶, 李佩文, 岳颀. 基于约束优化的多智能体协同编队与避障[J]. 南京航空航天大学学报, 2024, 56(03): 545-560.

[10] 宿浩, 张宝琳, 籍艳, 等. 基于虚拟排斥力的移动多智能体覆盖控制动态博弈算法[J]. 中国科学: 信息科学, 2022, 52(12): 2195-2212.

[11] 陈龙, 刘坤华, 周宝定, 等. 多智能体协同高精地图构建关键技术研究[J]. 测绘学报, 2021, 50(11): 1447-1456.

[12] D'Andrea R, Wurman P. Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities[C]//2008 IEEE International Conference on Technologies for Practical Robot Applications, 2008: 80-83.

[13] Liu Y, Chen M, Huang H. Multi-agent pathfinding based on improved cooperative A* in Kiva system[C]//2019 5th International Conference on Control, Automation and Robotics (ICCAR), 2019: 633-638.

[14] Yang C, Yuan B, Zhai P. Actor-hybrid-attention-critic for multi-logistic robots path planning[J]. IEEE Robotics and Automation Letters, 2024, 9(6): 5559-5566.

[15] Zhou X, Wen X, Wang Z, et al. Swarm of micro flying robots in the wild[J]. Science Robotics, 2022, 7(66): eabm5954.

[16] Lee Y, Ahn T, Lee C, et al. A novel path planning algorithm for truck platooning using V2V communication[J]. Sensors, 2020, 20(24).

[17] Chen X, Leng S, He J, et al. Deep-learning-based intelligent intervehicle distance control for 6G-enabled cooperative autonomous driving[J]. IEEE Internet of Things Journal, 2021, 8(20): 15180-15190.

[18] 宣志玮,毛剑琳,张凯翔. CBS 框架下面向复杂地图的低拓展度 A*算法[J]. 电子学报, 2022, 50(08): 1943-1950.

[19] 王卓然,文家燕,谢广明等. 基于改进 CBS 算法的多智能体路径规划[J]. 智能系统学报, 2023, 18(06): 1336-1343.

[20] 张洪琳, 吴耀华, 胡金昌, 等. 一种基于改进冲突搜索的多机器人路径规划算法[J]. 控制与决策, 2023, 38(05): 1327-1335.

[21] 纪苗苗, 吴志彬. 考虑工人路径的多智能体强化学习空间众包任务分配方法[J]. 控制与决策, 2024, 39(01): 319-326.

[22] Omidshafiei S, Agha-Mohammadi A, Amato C, et al. Decentralized control of partially observable Markov decision processes using belief space macro-actions[C]//2015 IEEE International Conference on Robotics and Automation (ICRA), 2015: 5962-5969.

[23] Beynier A, Mouaddib A I. Decentralized Markov decision processes for handling temporal and resource constraints in a multiple robot system[C]//In: Alami R, Chatila R, Asama H. Distributed Autonomous Robotic Systems 6. Tokyo: Springer Japan, 2007: 191-200.

[24] Amato C, Chowdhary G, Geramifard A, et al. Decentralized control of partially observable Markov decision processes[C]//52nd IEEE Conference on Decision and Control, 2013: 2398-2405.

[25] Han Y, Gmytrasiewicz P J. IPOMDP-Net: A deep neural network for partially observable multi-agent planning using interactive POMDPs[C/OL]//AAAI Conference on Artificial Intelligence, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:69490477

[26] Zhang S, Li J, Huang T, et al. Learning a priority ordering for prioritized planning in multi-agent path finding[C]//Symposium on Combinatorial Search, 2022.

[27] Li W, Chen H, Jin B, et al. Multi-agent path finding with prioritized communication learning[J]. 2022 International Conference on Robotics and Automation (ICRA), 2022: 10695-10701.

[28] 史殿习, 彭滢璇, 杨焕焕, 等. 基于 DQN 的多智能体深度强化学习运动规划方法[J]. 计算机科学, 2024, 51(02): 268-277.

[29] 熊丽琴, 曹雷, 赖俊, 等. 基于值分解的多智能体深度强化学习综述[J]. 计算机科学, 2022, 49(09): 172-182.

[30] 李庆华, 王佳慧, 李海明, 等. 一种双阶段多智能体路径规划算法[J]. 科学技术与工程, 2021, 21(22): 9425-9431.

[31] 刘辉,肖克,王京擘. 基于多智能体强化学习的多 AGV 路径规划方法[J]. 自动化与仪表, 2020, 35(02): 84-89.

[32] Nowé A, Vrancx P, De Hauwere Y M. Game theory and multi-agent reinforcement learning[M]//In: Wiering M, van Otterlo M. Reinforcement Learning: State-of-the-Art. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 441-470.

[33] Deng L, Gong W, Li L. Multi-robot exploration in unknown environments via multi-agent deep reinforcement learning[C]//2022 China Automation Congress (CAC), 2022: 6898-6902.

[34] Saifullah M, Papakonstantinou K G, Andriotis C P, et al. Multi-agent deep reinforcement learning with centralized training and decentralized execution for transportation infrastructure management[J]. ArXiv, 2024, abs/2401.12455.

[35] Foerster J N, Assael Y M, de Freitas N, et al. Learning to communicate with deep multi-agent reinforcement learning[C]//NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems. Curran Associates Inc., 2016: 2145-2153.

[36] Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths[J]. IEEE Transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107.

[37] Han L, Wu X, Sun X. Hybrid path planning algorithm for mobile robot based on A* algorithm fused with DWA[C]//2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), 2023: 1465-1469.

[38] Lehnert L, Sukhbaatar S, McVay P, et al. Beyond A*: Better planning with transformers via search dynamics bootstrapping[Z]. 2024. ArXiv: 2402.14083 [cs.AI].

[39] Cao X, Xu Y, Yao Y, et al. An improved hybrid A* algorithm of path planning for hotel service robot[J]. International Journal of Advanced Computer Science and Applications, 2023, 14(10).

[40] Grenouilleau F, Hoeve W J V, Hooker J N. A multi-label A* algorithm for multi-agent pathfinding[J]. Proceedings of the International Conference on Automated Planning and Scheduling, 2021, 29(1): 181-185.

[41] Silver D. Cooperative pathfinding[C]//Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2005, 1(1): 117-122.

[42] Sharon G, Stern R, Felner A, et al. Conflict-based search for optimal multi-agent pathfinding[J]. Artificial Intelligence, 2015, 219: 40-66.

[43] Wen L, Liu Y, Li H. CL-MAPF: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints[J]. Robotics and Autonomous Systems, 2022, 150: 103997.

[44] Barer M, Sharon G, Stern R, et al. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem[C]//Symposium on Combinatorial Search, 2014.

[45] Li J, Ruml W, Koenig S. EECBS: A bounded-suboptimal search for multi-agent path finding[C]//AAAI Conference on Artificial Intelligence, 2020.

[46] Hou Y, Tao G, Zang Z, et al. Multi-agent path finding for non-conflicting paths: An infinite speed conflict-based search approach[C]//2023 42nd Chinese Control Conference (CCC), 2023: 5500-5505.

[47] Ren Z, Rathinam S, Choset H. A conflict-based search framework for multi-objective multiagent path finding[J]. IEEE Transactions on Automation Science and Engineering, 2023, 20(2): 1262-1274.

[48] Jin J, Zhang Y, Zhou Z P, et al. Conflict-based search with D* lite algorithm for robot path planning in unknown dynamic environments[J]. Comput. Electr. Eng., 2023, 105: 108473.

[49] Lee H, Motes J, Morales M, et al. Parallel hierarchical composition conflict-based search for optimal multi-agent pathfinding[J]. IEEE Robotics and Automation Letters, 2021, 6(4): 7001-7008.

[50] Yu J, LaValle S M. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics[J]. IEEE Transactions on Robotics, 2016, 32(5): 1163-1177.

[51] Wang H, Chen W. Multi-robot path planning with due times[J]. IEEE Robotics and Automation Letters, 2022, 7(2): 4829-4836.

[52] Mao Z, Wu Z, Fang X, et al. Agent maze path planning based on simulated annealing Q-learning algorithm[C]//2022 41st Chinese Control Conference (CCC), 2022: 2272-2276.

[53] Lamini C, Benhlima S, Elbekri A. Genetic algorithm-based approach for autonomous mobile robot path planning[J]. Procedia Computer Science, 2018, 127: 180-189.

[54] Neumann N M P, de Heer P, Chiscop I, et al. Multi-agent reinforcement learning using simulated quantum annealing[J]. Computational Science – ICCS 2020, 2020, 12142: 562-575.

[55] Tao Q, Sang H Y, Guo H W, et al. Improved particle swarm optimization algorithm for AGV path planning[J]. IEEE Access, 2021, 9: 33522-33531.

[56] Phillips M, Likhachev M. SIPP: Safe interval path planning for dynamic environments[C]//2011 IEEE International Conference on Robotics and Automation, 2011: 5628-5635.

[57] Narayanan V, Phillips M, Likhachev M. Anytime safe interval path planning for dynamic environments[C]//2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012: 4708-4715.

[58] Yakovlev K, Andreychuk A, Stern R. Revisiting bounded-suboptimal safe interval path planning[C]//Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS), 2020: 300-304.

[59] Ali Z A, Yakovlev K. Prioritized SIPP for multi-agent path finding with kinematic constraints[C]//Ronzhin A, Rigoll G, Meshcheryakov R. Interactive Collaborative Robotics. Springer International Publishing, 2021: 1-13.

[60] Ali Z A, Yakovlev K. Safe interval path planning with kinodynamic constraints[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2023, 37(10): 12330-12337.

[61] Russell S, Norvig P. Artificial intelligence: A modern approach[M]. 3rd ed. Upper Saddle River: Pearson, 2010.

攻读硕士学位期间参与的科研项目以及学术成果

论文发表情况：

[1] X. Xing, B. Wang and J. Cheng. Incremental CBS: Adaptive Multi-Agent Pathfinding in Dynamic Environments[C]//2024 7th International Conference on Robotics, Control and Automation Engineering (RCAE). Wuhu, China: IEEE, 2024: 164-169.（已见刊，本人第一作者）

参与项目：

[1] 徐州徐工特种工程机械有限公司横向项目：“搬运型 AGV 运动控制及作业调度系统研发”

[2] 安徽工程大学-鸠江区产业协同创新新专项基金项目：“基于 5G 云化的无人叉车及其作业调度系统研发与应用”

[3] 安徽工程大学产业创新技术研究有限公司横向项目：“数字孪生驱动的复合移动机器人协同制造关键技术研究”

致谢

三年光阴倏忽而过，犹记得二二年那个夏日，蝉声阵阵，阳光明亮，我怀着几分憧憬与忐忑，踏入这所校园，开启人生新篇。转眼间时光已老，回首这一段求学之路，有亲情相伴，有师恩如山；有困顿踟蹰之时，也有豁然开朗之日。此时此刻，心绪如潮，唯恨言语浅短，情意难尽。

首先要感谢我的导师汪步云教授。感谢汪老师为我提供了扎实的实验平台、丰富的学术资源以及与企业合作的宝贵机会，使我得以拓宽视野、明确方向，在困难中不断寻求突破。汪老师年轻有为，治学严谨，勤勉专注，是我十分敬佩的楷模。他今日之成就，是我今后努力的目标。愿我日后行稳致远，不负所学，不负师恩。

其次，在此我还要衷心感谢程军教授在这段旅程中的陪伴与指引。程老师学识渊博，性情开朗，总能在我陷入学术难题或生活迷茫时，给予恰如其分的提醒和鼓励。他专业上认真严谨，生活中宽厚温和，是我非常尊敬也十分感激的老师。他身上那份从容与坚实，是我今后希望慢慢靠近的模样。

此外，也想对一路同行的朋友与师兄弟们说声谢谢。三年里，我们携手并肩砥砺前行，在焦虑中相互安慰，在松弛中共享片刻的欢愉。无数日夜的陪伴，构成了这段时光中最鲜亮的一抹颜色。因为有你们，这段旅途才不觉孤单。

最后，要感谢我的家人。你们始终是我最温暖的依靠、最坚实的后盾。因为你们的理解和支持，我才能在奔波与挑战中保持前行的勇气。同时，也想轻声对自己说一句谢谢。走到今天，每一步都是经历，每一刻都是馈赠。愿你心中有光，脚下有路，未来无问西东，自在前行。

尚德教学 唯实惟新