

**本科生毕业设计（论文）**

**题 目：** **机器人联合探索环境的协同策略**

**姓 名：**  **牛 牧**

**学 号：**  **12112004**

**系 别：** **机械与能源工程系**

**专 业：**  **机器人工程**

“姓名、学号、指导教师、年级与专业、年月日”均用四号宋体打印，不得手写，各栏目下划线需统一长度

**指导教师：** **郑裕基,** **张晴**

2025年 4 月 26 日

**诚信承诺**

11.本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2.除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3.本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4.在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。



作者名：

2025 年 4 月 26 日

机器人联合探索环境的协同策略

牛牧

机械与能源工程系

指导教师：郑裕基、张晴

校外指导老师：Ong Chong Jin

[摘要]

本研究聚焦于多机器人协同探索中的策略设计，重点分析了传统分区方法（如K均值聚类）和确定性路径规划方法（如旅行商问题，TSP）存在的局限性。K均值聚类容易导致任务分配不均，造成某些区域巡逻负担过重；而TSP路径高度可预测，使攻击者能够在机器人离开巡逻点后立即发起攻击，从而显著提高攻击成功率。

为解决这些问题，本研究提出了改进的最小增量聚类(Improved Min Incremental Clustering, IMIC）分区方法，通过保持组内紧凑性和组间平衡来增强巡逻区域分区效果。此外，使用该方法结合马尔可夫转移模型的巡逻策略，以概率方式选择巡逻路径，从而降低了机器人运动的可预测性，提高了对等待攻击者的防御能力。

本研究通过模拟实验发现，IMIC分区方法在无障碍和有障碍环境中较K均值聚类分区方法分别提升了5%和17%的防御成功率。此外，基于IMIC的马尔可夫巡逻策略在抵御等待攻击者方面优于TSP巡逻策略，在无障碍和有障碍环境中分别提高了30%和20%的防御成功率。

这些结果证明了所提出分区方法和巡逻策略在增强多机器人巡逻策略方面的有效性，特别是在对抗性环境中。未来的研究可以整合视觉感知，实现实时威胁检测与自适应巡逻，以进一步提升系统性能。

[关键词]：多机器人; 区域巡逻; 路径规划; 聚类; 马尔可夫;

**[ABSTRACT]**

This study examines challenges in multi-robot area patrol strategies, particularly the vulnerabilities of traditional partitioning methods such as K-means clustering and deterministic approaches like the Traveling Salesman Problem (TSP). A key weakness of TSP-based patrols is their predictability—attackers can exploit this by striking immediately after a robot leaves a patrol point, reducing the effectiveness of surveillance.

To address these issues, an Improved Min Incremental Clustering (IMIC) method is proposed to enhance patrol region partitioning by maintaining both intra-group compactness and inter-group balance. Then integrated with a Markov transition model, this method enables probabilistic patrol path selection, reducing the predictability of robot movements and improving defense against waiting attackers.

Simulations conducted in Python, in environments with and without obstacles, show that the IMIC method improves defense success rates by 5% in obstacle-free environments and 17% in environments with obstacles. Additionally, the IMIC based Markov patrol strategy outperforms TSP by improving defense success against waiting attackers by 30% in obstacle-free environments and 20% in environments with obstacles.

These results demonstrate the effectiveness of the proposed approach in enhancing multi-robot patrol strategies, particularly in adversarial environments. Future work could further improve performance by integrating vision-based perception for real-time threat detection and adaptive patrolling.

**[Keywords]**: Multi-robot; area patrol; path planning ; clustering; Markov model;

**目录**

[1. 引言 1](#_Toc196487324)

[2.文献综述 2](#_Toc196487325)

[3.方法论 3](#_Toc196487326)

[3.1 分区 3](#_Toc196487327)

[3.1.1 k均值聚类 3](#_Toc196487328)

[3.1.2改进的最小增量聚类 4](#_Toc196487329)

[3.2 巡逻规划 7](#_Toc196487332)

[3.2.1 基于TSP的方法 7](#_Toc196487333)

[3.2.2 基于马尔可夫的多机器人巡逻算法 8](#_Toc196487334)

[3.3 路径优化使用A\*进行障碍物避免 10](#_Toc196487339)

[3.3.1 A\*算法实现 11](#_Toc196487340)

[3.3.2 障碍物避免策略 11](#_Toc196487341)

[3.4 攻击者建模 12](#_Toc196487342)

[4. 实验和评估 12](#_Toc196487343)

[4.1 实验设置 12](#_Toc196487344)

[4.2 实验结果 13](#_Toc196487345)

[4.2.1 实验1：聚类方法比较 13](#_Toc196487346)

[4.2.2 实验2：防御成功率 15](#_Toc196487351)

[4.2.3 实验3：最小机器人数量以实现70%防御成功率 17](#_Toc196487354)

[4.3 结果讨论 18](#_Toc196487355)

[4.3.1 分区方法比较 18](#_Toc196487356)

[4.3.2 确定性和概率**性**方法比较 18](#_Toc196487357)

[4.3.3 分区方法与非分区方法比较 18](#_Toc196487358)

[4.3.4 非分区通信与非分区不通讯的比较 19](#_Toc196487359)

[5. 总结 19](#_Toc196487360)

[6. 未来工作方向 20](#_Toc196487361)

[**6**.1 基于视觉的巡逻 20](#_Toc196487362)

[**6**.**2** 障碍物感知和覆盖优化 20](#_Toc196487363)

[参考文献 21](#_Toc196487364)

[附录 22](#_Toc196487365)

[符号列表 22](#_Toc196487366)

[致谢 23](#_Toc196487367)

# 1. 引言

多机器人区域巡逻是指一组机器人协同执行任务，定期访问指定区域内的关键目标，以保障其安全。这一领域在监控、环境监测和工业自动化等领域有着广泛的应用[1]。随着机器人技术的快速发展，多机器人协同巡逻系统面临着诸多挑战，包括任务分配、路径规划、障碍物规避以及对抗性威胁等问题[2]。这些挑战不仅涉及单个机器人的自主决策能力，还包括多机器人之间的协同配合、环境感知与适应、以及对抗性环境下的策略优化等多个方面。

1.1 研究背景与意义

多机器人系统（Multi-Robot Systems, MRS）的研究可以追溯到20世纪90年代。Parker[3]首次系统地提出了多机器人系统的研究框架，为后续研究奠定了基础。该框架将多机器人系统分为任务分配、路径规划、通信协调和环境感知四个主要模块，并详细阐述了各模块之间的交互关系。近年来，随着人工智能和机器学习技术的发展，多机器人系统在自主性、适应性和协同性方面取得了显著进展[4]。深度学习、强化学习等新技术的应用，使得机器人能够更好地理解环境、预测变化并做出相应决策。

在安防监控领域，多机器人系统被广泛应用于大型设施、边境线等区域的持续监控[5]。通过部署多个具有不同功能的机器人，系统能够实现全天候、全方位的监控覆盖。例如，在大型工业园区，多机器人系统可以协同工作，实现重点区域的实时监控，及时发现安全隐患。在环境监测方面，多机器人系统在森林火灾监测、海洋环境监测等任务中发挥着重要作用[6]。机器人可以携带各种传感器，收集环境数据，为决策提供支持。在工业自动化领域，多机器人系统被用于工厂巡检、设备维护等任务[7]。通过协同工作，机器人能够提高生产效率，降低人力成本。在城市管理方面，多机器人系统在智慧城市建设中扮演着重要角色，包括交通监控、公共安全等[8]。

1.2 国内外研究现状

在区域巡逻任务中，分区策略是一个关键问题。传统方法如K均值聚类虽然计算效率高，但存在任务分配不均的问题。K均值聚类算法通过迭代优化来最小化类内距离，但容易受到初始点选择的影响，导致某些区域任务负载过重。为解决这一问题，研究人员提出了多种改进方法。Sea等人[9]基于非均匀访问频率改进了传统的k均值聚类算法，通过考虑各区域的访问频率差异，实现了更均衡的任务分配。Sugiyama等人[10]应用强化学习实现高效巡逻，使机器人能够根据环境变化动态调整巡逻策略。

基于图论的分区方法为多机器人系统提供了新的思路。最小生成树（MST）方法通过构建最小权重的树形结构，实现了区域的有效划分[11]。这种方法特别适用于需要最小化总路径长度的场景。谱聚类方法利用图的拉普拉斯矩阵特征值，将区域划分为多个子区域[12]。通过分析图的谱特性，谱聚类能够发现数据的内在结构，实现更合理的分区。Voronoi图方法则基于距离度量，将空间划分为多个 Voronoi 单元，每个单元由最近的机器人负责[13]。这种方法在保证覆盖均匀性的同时，也考虑了机器人的移动能力。

路径规划是多机器人巡逻的核心问题之一。在确定性路径规划方面，旅行商问题（TSP）及其变体被广泛应用于寻找最优巡逻路径[14]。TSP问题虽然计算复杂度高，但通过启发式算法和近似算法，可以在合理时间内得到较好的解。例如，遗传算法、模拟退火算法等都被成功应用于求解TSP问题。A\*算法及其改进版本在路径规划中表现出色，通过结合启发式函数和实际代价，能够快速找到最优路径[15]。动态规划方法则通过将问题分解为子问题，逐步求解，适用于具有特定约束的路径规划问题[16]。

在概率性路径规划方面，马尔可夫决策过程（MDP）提供了一种基于状态转移的决策框架[17]。通过定义状态、动作和奖励函数，MDP能够帮助机器人做出最优决策。这种方法特别适用于具有不确定性的环境。部分可观察马尔可夫决策过程（POMDP）进一步考虑了环境的不确定性，使机器人能够在部分可观察的情况下做出决策[18]。强化学习方法则通过试错学习，使机器人能够从经验中学习最优策略[19]。这些方法为多机器人系统提供了更灵活的路径规划解决方案。

多机器人协同控制是确保系统高效运行的关键。在任务分配方面，基于市场的方法通过模拟市场机制，实现了任务的高效分配[20]。在这种方法中，每个机器人作为市场参与者，通过竞标来获取任务。基于拍卖的算法则通过多轮竞价，确保任务被分配给最合适的机器人[21]。基于合同网的方法则通过建立任务合同，明确各机器人的职责和权利[22]。这些方法各有特点，能够适应不同的任务分配需求。

在通信与协调方面，基于图论的通信拓扑为多机器人系统提供了可靠的通信框架[23]。通过分析机器人之间的通信关系，系统可以优化通信结构，提高通信效率。分布式控制策略使机器人能够根据局部信息做出决策，同时保持全局目标的一致性[24]。共识算法则确保多个机器人在没有中央控制器的情况下，能够达成一致[25]。这些方法共同构成了多机器人系统协同控制的理论基础。

在对抗性环境中，巡逻策略需要特别考虑攻击者的行为模式。研究人员提出了多种攻击者模型，包括随机攻击者模型、等待攻击者模型和智能攻击者模型[26]-[28]。随机攻击者不考虑机器人巡逻行为，在随机位置发起攻击。等待攻击者则监视机器人移动，在机器人离开节点后立即发起攻击，利用巡逻时间表中的空档。智能攻击者则能够分析机器人的巡逻模式，选择最优的攻击时机和位置。

针对这些攻击模式，研究人员提出了多种防御策略。概率性巡逻通过引入随机性，降低机器人运动的可预测性[29]。自适应巡逻则根据环境变化和攻击模式，动态调整巡逻策略[30]。预测性巡逻通过分析历史数据，预测可能的攻击位置和时间，提前部署防御力量[31]。这些策略在对抗性环境中都展现出了良好的效果。

环境感知与障碍物规避是多机器人系统在实际应用中必须面对的问题。在环境建模方面，栅格地图将环境离散化为网格，每个网格包含环境信息[32]。这种方法简单直观，但计算复杂度随环境大小线性增长。拓扑地图则通过节点和边表示环境的关键特征[33]，更适合大规模环境。混合地图结合了栅格地图和拓扑地图的优点，既保持了细节信息，又降低了计算复杂度[34]。

在障碍物规避方面，基于势场的方法通过建立势场函数，引导机器人避开障碍物[35]。这种方法计算简单，但可能陷入局部最优。基于速度障碍的方法则通过分析障碍物的运动，预测可能的碰撞，提前调整路径[36]。基于RRT（Rapidly-exploring Random Tree）的方法通过随机采样和树形扩展，快速找到可行路径[37]。这些方法各有优势，能够适应不同的避障需求。

近年来，随着深度学习技术的发展，基于学习的避障方法也取得了显著进展。通过训练神经网络，机器人能够从经验中学习避障策略，提高避障效率。同时，多传感器融合技术的发展，使得机器人能够更准确地感知环境，为避障决策提供更可靠的信息支持。

1.3 研究内容与创新点

基于对现有研究的深入分析，本文提出了一种改进的最小增量聚类（Improved Min Incremental Clustering，IMIC）方法，用于优化多机器人巡逻系统的区域划分。该方法通过保持组内紧凑性及组间工作量平衡，实现了更合理的巡逻区域划分。在此基础上，我们结合马尔可夫转移模型，构建了一种概率性巡逻策略，显著降低了机器人运动的可预测性，增强了对等待攻击者的防御能力。

本文的主要创新点包括：

首先，在分区策略方面，本文提出的IMIC方法通过最远点初始化并逐步最小化组内平均距离的增加，有效平衡了组内距离最小化与方差控制。与传统的K均值聚类相比，IMIC方法不仅考虑了空间距离，还引入了时间维度的考虑，使得分区结果更符合实际巡逻需求。通过动态调整分区权重，IMIC方法能够适应不同环境下的巡逻任务，提高了系统的适应性和鲁棒性。

其次，在巡逻策略方面，本文将马尔可夫决策过程应用于分区结果，实现了更合理的巡逻策略。通过建立状态转移概率矩阵，系统能够根据当前状态和环境影响，动态选择最优的巡逻路径。这种概率性的决策机制，使得机器人的运动轨迹具有不可预测性，有效防止了等待攻击者的伏击。同时，通过引入时间权重因子，系统能够优先访问长时间未巡逻的区域，提高了整体巡逻效率。

最后，在对抗性环境下的防御能力方面，本文提出的方法展现出了显著优势。通过实验验证，在无障碍环境中，IMIC方法比传统K均值聚类提高了5%的防御成功率；在有障碍环境中，提升幅度达到17%。特别是在面对等待攻击者时，基于IMIC的马尔可夫巡逻策略比传统TSP巡逻策略在无障碍和有障碍环境中分别提高了30%和20%的防御成功率。这些结果证明了所提方法在对抗性环境中的有效性。

1.4 文章结构

本论文共分为六章，具体结构如下：

第一章为引言与文献综述，介绍了研究背景、意义，以及相关领域的研究现状。

第二章详细介绍了用于环境分区的关键方法，包括传统K均值聚类方法和本文提出的IMIC方法，并分析了各种方法的优缺点。

第三章介绍了模拟实验设置以及实验结果，比较了不同策略的防御表现，包括分区方法比较、防御成功率分析和最小机器人数量需求分析。

第四章对实验结果进行了深入讨论，分析了不同分区方法、确定性和概率性方法、分区与非分区方法、以及通信与非通信方法的性能差异。

第五章对研究成果进行了总结，归纳了主要创新点和贡献。

第六章探讨了未来的改进方向，包括基于视觉的巡逻机制与障碍物感知策略，为后续研究提供了思路。

# 2.方法论

2.1 分区

本研究的分区方法比较了两种方法：k均值聚类和改进的最小增量聚类。改进的最小增量聚类方法的一个关键创新是基于最远化分区的种子初始化，这确保了集群的良好分布起点。

2.1.1 k均值聚类

k均值聚类是一种迭代算法，它将一组点分成个集群。目标是最大限度地减少总集群内方差，定义为：

|  |  |  |
| --- | --- | --- |
|  |  | （1） |

其中：

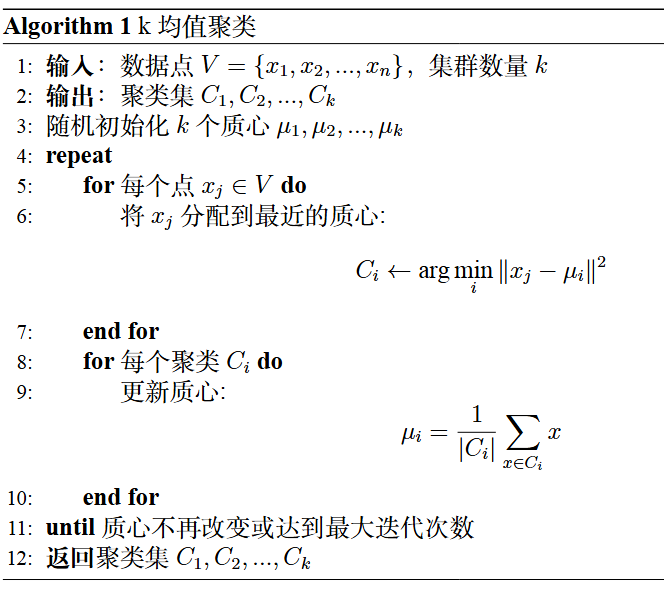
• 表示集群 i

• : 是集群的质心。

• : 是点 x 和质心之间的欧几里得距离平方。

所提出的分区算法基于k均值方法的详细信息如算法1所示。k均值聚类算法从初始化k个质心（第3行）开始，这些质心是随机选择的。在分配步骤（第4行到第7行）中，每个数据点根据欧几里得距离分配给最近的质心。然后更新质心（第8行到第10行）通过计算分配给每个集群的点的平均值。该过程重复，直到质心稳定或达到最大迭代次数（第11行）。最后，算法返回集群（第12行）。

k均值聚类具有相对较低的时间复杂度并且易于收敛，但它对粒子初始选择非常敏感。



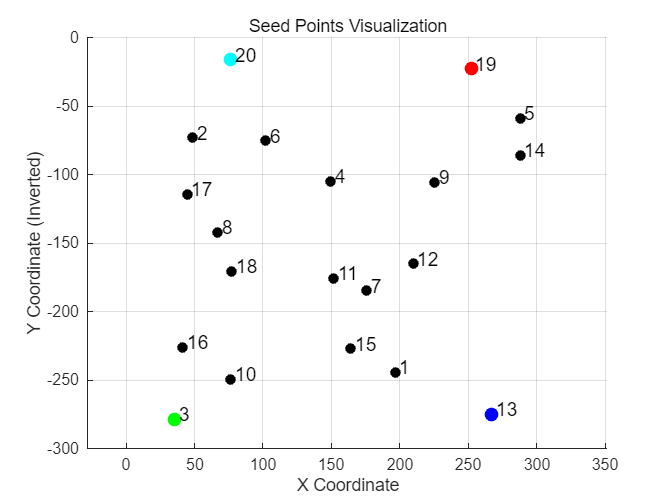
2.1.2改进的最小增量聚类

种子初始化

以下公式实现贪婪最大最小采样策略：

|  |  |  |
| --- | --- | --- |
|  |  | （2） |

其中包含前k个选定的种子。这确保初始点分布覆盖最大可能区域，如图2.1所示即为将20个巡逻点，分为4组的初始点初始化。具有颜色的点即为初始点的位置。



**图2.1 初始点可视化**

动态分配

对于每个未分配点p，计算将p添加到现有集群中时总距离的增加：

|  |  |  |
| --- | --- | --- |
|  |  | （3） |

其中表示点p和q之间的欧几里得距离，是点p可能分配到的集群。

点将会分配到最小增量集群：

|  |  |  |
| --- | --- | --- |
|  |  | （4） |

一旦选择了最佳集群，点p被添加到集群，并且集群的总距离相应地更新：

|  |  |  |
| --- | --- | --- |
|  |  | （5） |

基于改进的最小增量方法的分区算法具体步骤如算法2所示。该算法首先初始化种子点集。，其中初始的两个点，和，之间具有最大欧几里得距离（第3行）。然后，对于每个附加种子（第4-7行），算法使用最大最小策略来选择下一个最大化平均距离的点。

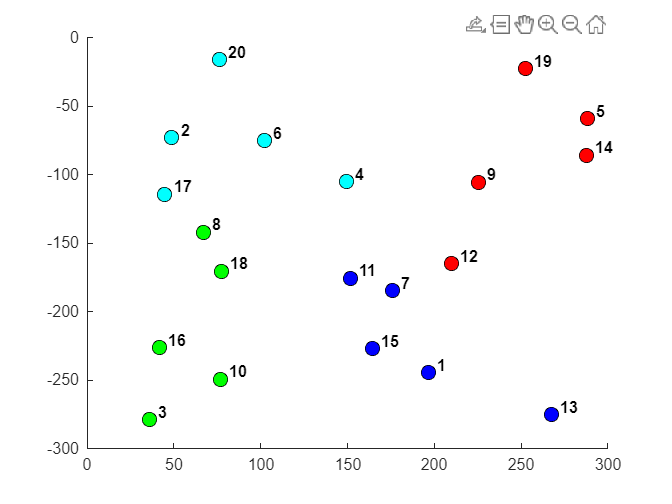
在选择种子之后，每个集群用种子点初始化，并且每个集群的总距离设置为零（第8行）。分配点集A初始化为种子集（第9行）。计算所有点之间的成对距离矩阵（第10行）。

主循环（第11-22行）运行，直到所有点都被分配到集群。对于每个未分配点p（第13行），算法检查将p添加到每个集群的效果。它计算每个集群的增量（第15行），如果增量小于当前最佳，则更新最佳集群和点（第16行）。

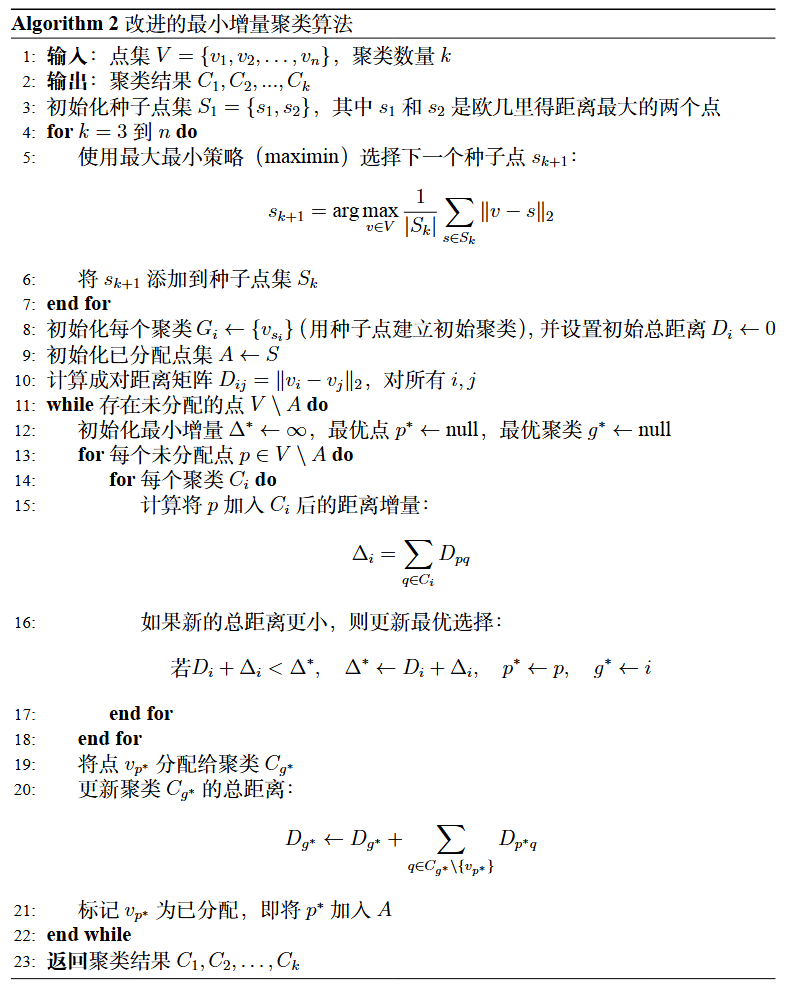
一旦找到最佳点和集群，点被分配到集群（第19行）。集群的总距离通过添加和集群中其他点之间的距离来更新（第20行）。最后，被标记为分配并添加到集合A（第21行）。

算法继续此过程，直到所有点都被分配到集群。最后，算法返回集群（第23行）。

动态编程方法确保至少有63%的近似比率与最优NP-hard解决方案进行比较。（"贪婪"启发式总是产生至少倍于最优值的解决方案[[38]](#ref10)。）图2.2展示了最终的分区结果。



**图2.2 多机器人分区可视化**



2.2 巡逻规划

巡逻规划涉及确定机器人分配区域内的运动策略。目标是确保机器人高效访问所需节点，同时避免碰撞并最大化每个节点之间的访问频率。

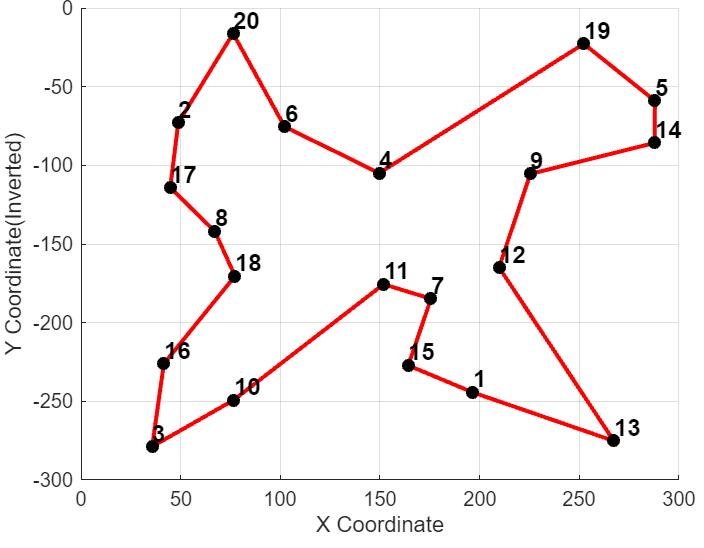
2.2.1 基于TSP的方法

为了高效地解决TSP问题，我们使用动态规划算法维护状态表示，其中每个访问节点子集与到达该子集的最短路径一起存储。递归关系如下：

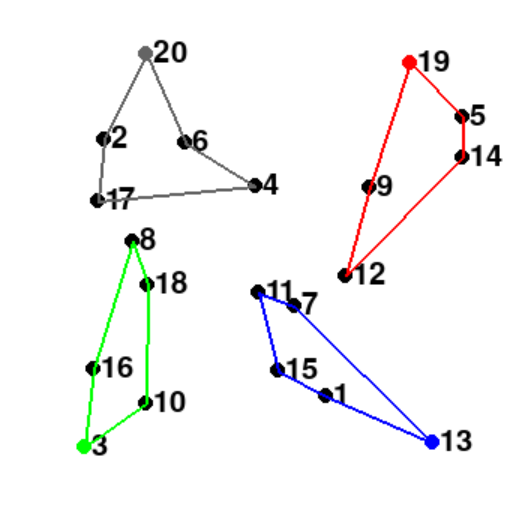
|  |  |  |
| --- | --- | --- |
|  |  | （6） |

其中S是节点子集，是当前节点，表示节点和之间的距离。最终的最优路径是通过回溯通过父矩阵重建的。

这种方法确保了接近最优的路径，但具有较高的计算复杂性，不适用于大规模环境。图2.3显示了没有分区的TSP方法的路径，图2.4展示了带有分区的TSP方法的路径。



**图2.3无分区TSP路线**



**图2.4 分区TSP路线**

2.2.2 基于马尔可夫的多机器人巡逻算法

为克服TSP巡逻策略的可预测性问题，本研究提出了一种基于马尔可夫决策过程（MDP）的巡逻算法。该算法通过概率方式选择下一个巡逻点，使机器人能够动态适应环境变化，尤其是对抗性行为。具体而言，算法融合了两类策略：一种是基于马尔可夫转移矩阵的区域分区方法，另一种是不进行分区的策略，包括带通信和不带通信两种子方法。上述策略均基于马尔可夫转移矩阵决定巡逻点的转移概率，该矩阵综合考虑空间分布与时间动态因素。

分区方法与马尔可夫转移矩阵

在分区方法中，环境被分成单独的区域，每个机器人分配巡逻一个特定区域。在每个区域中，机器人使用马尔可夫转移矩阵选择其下一个巡逻点。转移概率取决于两个因素：到候选点的距离和该点距离上次被访问的时间。从点到点的转移权重定义为：

|  |  |  |
| --- | --- | --- |
|  |  | （7） |

• ：自上次访问节点 j 以来的时间间隔

• ：节点 i 和 j 之间的欧几里得距离

• ：时间权重系数

归一化产生转移概率矩阵：

|  |  |  |
| --- | --- | --- |
|  |  | （8） |

机器人然后根据这个分布选择其下一个巡逻点。它更倾向于最近没有访问或更接近自己的点。这种方法确保了高效覆盖，同时适应环境变化。

在这种分区马尔可夫方法中，每个机器人的转移矩阵是独立计算的，不受其他机器人影响。允许每个机器人仅根据其自己的巡逻历史和本地环境做出移动决策。

**表2.1 马尔可夫转移矩阵**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P2 | P3 | P8 | P10 |
| P2 | 0.0000 | 0.1982 | 0.5727 | 0.2291 |
| P3 | 0.1524 | 0.0000 | 0.2239 | 0.6237 |
| P8 | 0.4607 | 0.2341 | 0.0000 | 0.3052 |
| P10 | 0.1614 | 0.5713 | 0.2673 | 0.0000 |



**图2.6 机器人从 P2 移动到 P3 示意图**

图2.6展示了我们环境中的一个小子区域。绿色圆点（没有序号）表示机器人当前正在从节点移动到节点。到的转移概率可以在表2.1中找到，该表显示了0.1982的概率。这表明机器人根据马尔可夫转移矩阵选择此巡逻路径的可能性。

非分区方法

非分区的方法消除了区域约束，允许所有机器人巡逻整个环境。此策略中实现了两个子方法：

无通信

在第一个子方法中，机器人独立操作，没有机器人通信。由于此方法不涉及分区，每个机器人都有可能访问任何巡逻点。因此，所有机器人共享相同的转移矩阵，确保团队中移动概率的一致性。然而，决策过程保持独立，这意味着虽然机器人依赖相同的转移模型，但它们的个体巡逻路线是自主确定的，没有直接来自其他机器人的影响。虽然此方法更简单，不需要通信开销，但它可能导致冗余覆盖或巡逻间隙，因为缺乏协调。

有通信

第二个子方法引入了机器人通信，使机器人能够共享下一个意图巡逻点的信息。转移矩阵计算保持相似，但决策过程被调整以稍微偏好最近被其他机器人访问的点。此"双重检查"机制旨在防止立即在机器人离开点后发动攻击的对抗攻击。通过纳入对其他机器人运动的认识，此方法增强了安全性并提高了覆盖效率，尽管它增加了计算和通信复杂性。概率选择确保灵活性，而偏向最近访问点的偏差加强了系统对对抗利用的抵抗力。

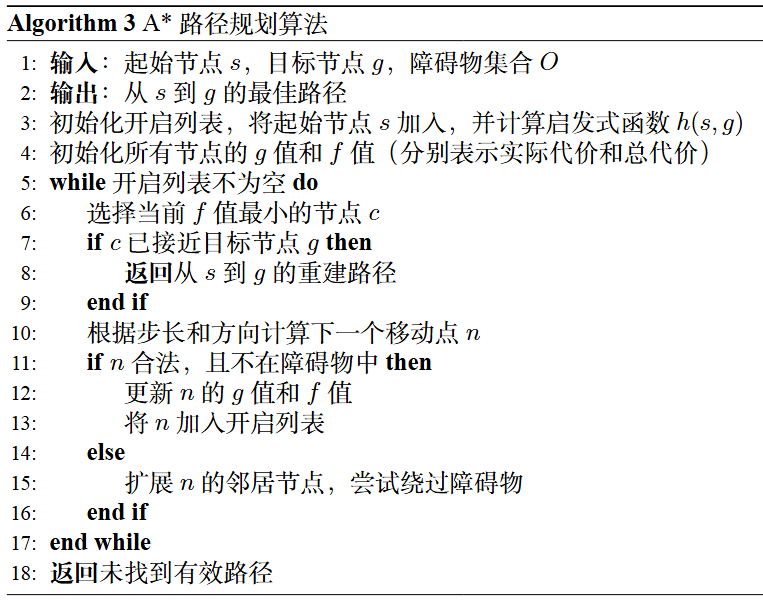
2.3 路径优化使用A\*进行障碍物避免

本节解释了使用A\*算法进行路径优化。A\*算法帮助机器人导航障碍物环境，确保高效路径规划。它考虑障碍物以找到最佳路径，避免碰撞并引导机器人到达其目标节点

2.3.1 A\*算法实现

A\*算法旨在找到从起始节点到目标节点的最短路径，同时需要避免与障碍物发生碰撞。由于模拟环境的连续性，用两个点之间的欧几里得距离作为启发函数。

|  |  |  |
| --- | --- | --- |
|  |  | （9） |

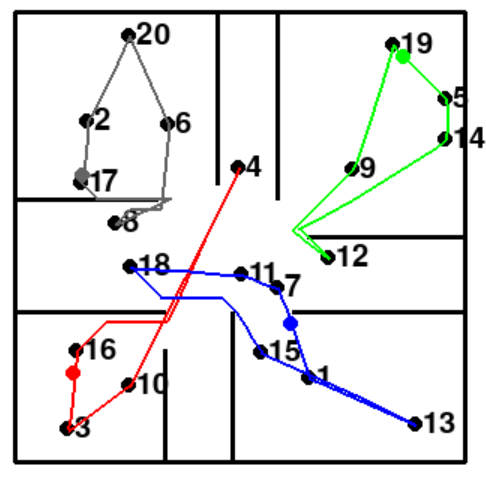
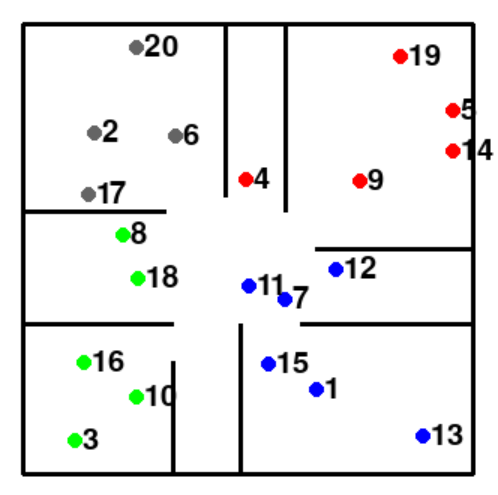


A\*路径规划算法从初始化开放集开始，该开放集包含起始节点并计算启发以估计到目标节点的距离（第3行）。然后初始化所有节点的-scores和-scores（第4行）。然后连续选择-score最低的节点（第6行）。如果节点接近目标，则返回重建路径（第8行）。否则，算法根据步长和方向计算下一个移动（第10行）。如果移动是有效的并且不在障碍物中，则更新-score和-score，并将添加到开放集中（第12-14行）。如果被障碍物阻挡，则算法将扩展到邻近节点以找到有效路径（第15行）。如果开放集为空且未找到路径，则算法返回"未找到有效路径"（第18行）。

2.3.2 障碍物避免策略

当直接移动到目标被阻挡时，算法探索邻近点以重新路由绕过障碍物。它考虑有效的移动方向，包括主要方向，如上、下、左、右，以及对角线方向，如左上、右上、左下、右下。

算法动态选择替代路径以防止不必要的回溯并确保即使在复杂环境中也能高效导航。图2.7显示了生成的房间形状障碍物图像。图2.8说明了使用分区TSP方法生成的路径。



**图3.8 A\*路径规划图**

**图3.7 障碍物地图**

2.4 攻击者建模

在本节中，我们展示了攻击者在模拟中的建模。考虑了两种对抗行为：

• 随机攻击者：这些攻击者随机攻击环境中巡逻点的位置，而不考虑机器人巡逻行为。

• 等待攻击者：这些攻击者监控机器人运动并等待正确时刻发动攻击。他们等待攻击机器人离开某节点后，直接攻击该节点，利用巡逻时间表中的间隔来实现攻击。

# 3. 实验和评估

在本节中，我们设计实验以评估所提出的方法并分析模拟结果。

3.1 实验设置

在此实验中，考虑了一个大小为的区域，其中随机分布了20个巡逻点。从实验开始，每2.5分钟出现一次攻击者。这些攻击者要么全部是随机攻击者，要么全部是等待攻击者。每次攻击持续10分钟。如果机器人在此10分钟窗口期间通过相应点，则防御被认为是成功的。否则，防御被标记为失败。

在整个实验中进行了300次攻击。这些攻击的结果用于评估巡逻策略的性能。

为了比较IMIC方法与传统k均值聚类方法的性能，本研究对两种方法进行了对比实验。实验使用三个机器人巡逻20个点，在区域中使用相同的TSP巡逻方法。使用TSP巡逻方法来比较的原因是它是一种确定性方法，允许更清晰的分析分区对实验结果的影响。

此外，基于此分区方案，比较确定性和概率性巡逻策略的性能。

3.2 实验结果

实验比较了传统k均值分区方法和改进的最小增量聚类方法。此外，通过模拟实验比较了各种巡逻方法，包括旅行商问题（TSP）、分区TSP、分区马尔可夫方法、非分区非通信马尔可夫方法和非分区通信马尔可夫方法。以下列出了三个主要实验的结果。

3.2.1 实验1：聚类方法比较

本实验对两个聚类方法，K均值聚类方法和改进的最小增量聚类（IMIC）方法，进行了比较。

聚类性能分析

图3.1和图3.2显示了两种方法在巡逻点被分为3组情况下的聚类结果。图3.3和图3.4显示了方法在巡逻点被分为4组情况下的聚类结果。表3.1和表3.2显示了方法在3组和4组情况下的数据比较。

改进的最小增量方法在聚类一致性方面优于K均值聚类方法。尽管两者在总体平均距离上相近，改进的最小增量方法展现出更低的方差，生成了更加稳定且均匀分布的分区。在3组和4组的场景中，K均值聚类方法表现出较高的方差，导致分区更加分散且负载不均。相比之下，改进的最小增量方法能够实现更紧凑和平衡的区域划分。

|  |  |
| --- | --- |
| **图3.1 K均值聚类（3组）** | **图3.2 IMIC聚类（3组）** |
| **图3.3 K均值聚类（4组）** | **图3.4 IMIC聚类（4组）** |

**表3.1 聚类方法比较（3 组）**

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 总平均距离 | 组内平均距离 | 方差 |
| K均值 | 102.3 | [115.8, 63.4, 77.1] | 493.1 |
| IMIC | 107.6 | [127.9, 97.2, 93.9] | 233.8 |

**表3.2 聚类方法比较（4 组）**

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 总平均距离 | 组内平均距离 | 方差 |
| K均值 | 75.6 | [82.0, 63.4, 48.4, 77.1] | 170.5 |
| IMIC | 79.0 | [83.3, 77.3, 77.6, 77.6] | 6.3 |

TSP性能与聚类方法比较

无障碍物

图3.6和图3.7显示了两种方法的路径。

在无障碍物环境中，如图3.8所示，改进的最小增量聚类（IMIC）比k均值提高了5%的性能。

|  |  |
| --- | --- |
| **图3.6 无障碍环境K均值分区方法TSP路线** | **图3.7 无障碍环境IMIC分区方法TSP路线** |
| **图3.8 聚类方法比较无障碍物** | **图3.9 聚类方法比较有障碍物** |

有障碍物

|  |  |
| --- | --- |
| **图3.10障碍环境K均值分区方法TSP路线** | **图3.11 障碍环境IMIC分区方法TSP路线** |

图3.10和图3.11显示了两种方法的路径。

在存在障碍物的情况下，分区方法不再基于欧几里得距离。相反，两个点之间的距离使用A\*算法计算，该算法找到最短路径，同时避免障碍物。这确保了聚类和后续TSP模拟考虑实际可导航路径，而不是直线距离，提供了更真实的场景表示。

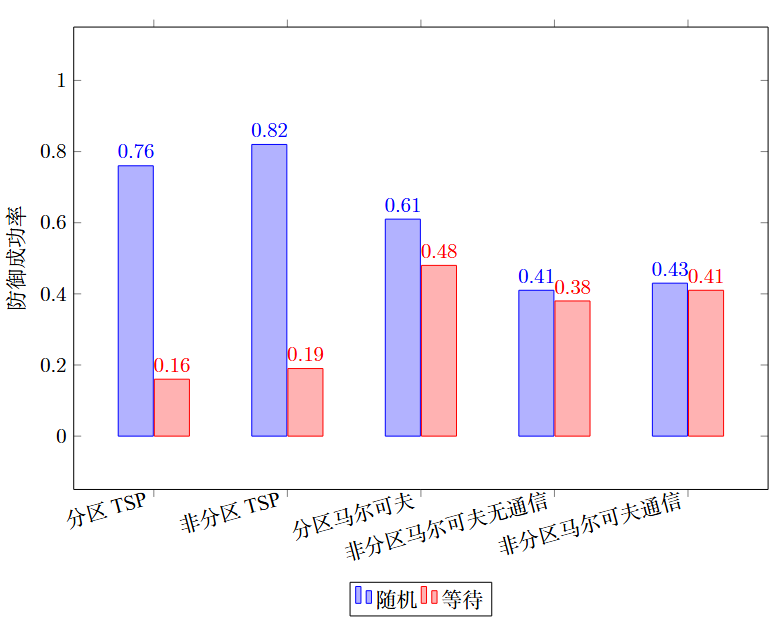
如图3.9所示，可以观察到改进的最小增量聚类方法比k均值方法的路线优化大约提高了17%。

3.2.2 实验2：防御成功率

无障碍物

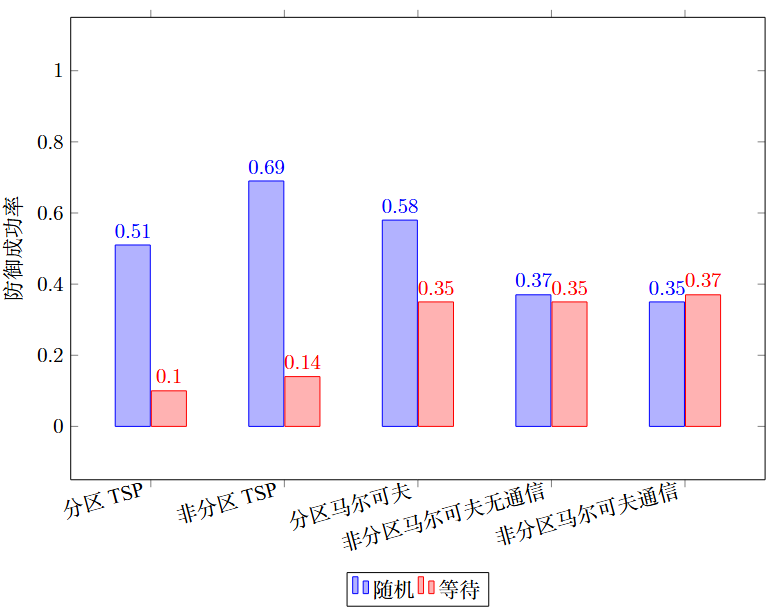
图3.12显示了不同方法的防御成功率。对于随机攻击者，非分区TSP（0.82）表现最佳，略高于分区TSP（0.76），因为它们的固定路线提供了对巡逻点更好的覆盖性。分区马尔可夫（0.61）效果较差，而无通信（0.41）和通信（0.43）方法效果最差，突出了非分区马尔可夫策略中的覆盖效率不足。然而，对于等待攻击者，分区马尔可夫（0.48）实现了最高的成功率，显示了其在处理策略型攻击者方面的优势。非分区马尔可夫无通信（0.38）和非分区马尔可夫通信（0.41）也优于TSP方法，由于TSP方法的可预测巡逻路线，防御成功率较低分别为0.16和0.19。

总体而言，非分区TSP在随机攻击者方面表现最佳，而分区马尔可夫在等待攻击者方面表现最佳。此外，分区马尔可夫在两种攻击者类型方面表现都较好，提供了比TSP更平衡的防御，TSP表现出极端结果——要么非常有效，要么非常弱，取决于攻击类型。通信在非分区马尔可夫方法中提供了微小的性能改进，但不会显著改变其有效性。

**图3.12 无障碍物不同模型和攻击者类型的防御成功率**

有障碍物

在存在障碍物的环境中，实验结果如图3.13所示，与无障碍物环境中的结果相似。对于随机攻击者，非分区TSP方法表现最佳，其次是分区马尔可夫方法。此外，对于等待攻击者，马尔可夫方法显示了更好的性能。总体而言，分区马尔可夫方法展示了更稳定的性能，可以有效地处理不同类型的攻击者。



**图3.13 有障碍物不同模型和攻击者类型的防御成功率**

3.2.3 实验3：最小机器人数量以实现70%防御成功率

第三个实验评估了不同方法以实现70%防御成功率所需的最小机器人数量。在此实验中，未考虑障碍物。

**表3.3 不同方法和攻击类型的策略达到70%防御成功率机器人数量要求**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 攻击类型 | TSP | | 马尔可夫 | | |
| IMIC | 无分区 | |
| IMIC | 无分区 | 无通讯 | 通讯 |
|
| 随机攻击者 | 3 | 3 | 4 | 7 | 6 |
|
| 等待攻击者 | 4 | 4 | 4 | 6 | 6 |
|

表3.3展示了在不同方法和攻击类型下，实现70%防御成功率所需的最小机器人数量。结果表明，对于马尔可夫方法，分区策略通常能够减少机器人需求。具体而言，在应对随机攻击者时，分区TSP和分区马尔可夫方法分别只需3个和4个机器人，而对应的非分区的方法则需4个和6个机器人，体现了分区策略在提升巡逻效率方面的优势。

在应对等待攻击者时，分区方法同样表现更优：分区TSP和分区马尔可夫均需4个机器人，而非分区的方法需要6个机器人。这进一步说明，分区能够有效降低实现目标防御率所需的机器人数量。

总体来看，分区方法在不同攻击类型下均能减少机器人需求，尤其在面对随机攻击者时，优势更为明显。

3.3 结果讨论

本节讨论了不同分区方法、巡逻方法及其适用场景的优点和缺点。

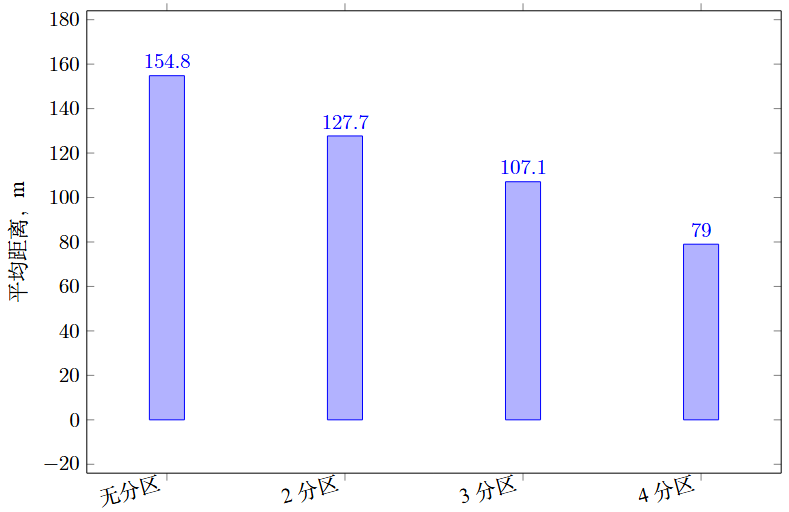
3.3.1 分区方法比较

如果追求最低的总平均距离（不考虑组内平衡），建议选择k均值，但需要注意的是，k均值可能会将太多紧密聚集的巡逻点分配到同一组中，导致不均匀的巡逻工作量，然而最小增量方法并不会有这个问题。如果需要在组间工作量平衡和组内低平均距离之间取得平衡，改进的最小增量方法表现更好。

3.3.2 确定性和概率性方法比较

确定性方法（TSP）在随机攻击者方面表现出色，因为它提供了高效覆盖所有点的巡逻路线。然而，当面对等待攻击者（在机器人离开点后立即攻击的攻击者）时，TSP的防御成功率会显著下降。这是因为TSP遵循固定路径，导致攻击者很容易预测并利用它。相比之下，概率方法（马尔可夫），它们概率性地确定下一个巡逻点，引入了不可预测性并在等待攻击者的情况下表现很好。然而，它们在随机攻击者方面不如TSP有效。图4.12说明了这种比较。

3.3.3 分区方法与非分区的方法比较

****与非分区巡逻策略相比，分区马尔可夫方法在多个方面展现出更优越的性能。该方法通过将巡逻区域划分为若干子区域，并指定每台机器人仅在所属区域内执行巡逻任务，有效减少了每台机器人需要负责的巡逻点数量，从而显著缩短了点与点之间的平均距离。这种区域内的紧凑布局使得机器人在单位时间内能够更高频地访问每一个点，降低了单个点在连续两次被访问之间的时间间隔，因此在面对攻击者时提高了巡逻系统的响应能力和防御成功率。图3.14进一步验证了这一趋势，表明分区马尔可夫方法在巡逻点数量较多的场景中尤为有效。

然而，这一策略的优势并不适用于所有类型的巡逻算法。以旅行商问题（TSP）为基础的巡逻方法为例，其核心在于对路径的全局最优排序，无论点的数量如何，TSP算法都会寻求一条总巡逻距离最短的路径。在这种情况下，路径的全局最优性已经内建于算法之中，分区操作反而可能限制路径的整体优化空间，从而降低效率。因此，分区策略在提升马尔可夫方法性能方面具有显著意义，而对于TSP方法则作用有限，甚至可能适得其反。

**图3.14 不同 IMIC 分区的马尔可夫方法的平均距离**

3.3.4 非分区通信与非分区无通讯的比较

结果表明，虽然通信略微提高了性能，但差异仍然很小。这主要是由于不同攻击者类型的性质。对于随机攻击者，由于攻击位置是随机选择的，通信在机器人巡逻模式方面没有优势，因为攻击是独立的。因此，两种通信和非通信方法都产生了类似的防御成功率。

对于等待攻击者，通信的有效性取决于机器人优先重新访问以前巡逻的点。如果双重检查权重增加，机器人将更可能重新访问最近巡逻的点，提高防御等待攻击者的能力。然而，这导致了选择巡逻点时的低灵活性。在场景中，攻击者策略从等待切换到随机时，过度优先考虑双重检查可能会减少整体巡逻覆盖，使系统更容易受到随机攻击。这种权衡解释了为什么通信的好处仍然有限。

# 4. 总结

本论文提出了一种基于分区的多机器人协作探索方法。核心贡献在于改进的最小增量聚类（IMIC）算法，该方法通过最远点初始化并逐步最小化组内平均距离的增加，有效平衡了组内距离最小化与方差控制，克服了K均值算法在任务分配不均方面的局限性。在此基础上，将马尔可夫决策过程（MDP）应用于分区结果，实现更合理的巡逻策略。本文在三台机器人巡逻20个点的场景下，对比分析了改进方法与传统TSP方法、非分区MDP方法的性能。实验结果表明，IMIC结合MDP能够在更少机器人数量下实现更高的防御成功率，尤其在面对不同类型攻击者时表现出更强的鲁棒性和效率。

# 5. 未来工作方向

**5**.1 基于视觉的巡逻

当前巡逻策略需要机器人物理访问巡逻点以检测对抗攻击者，而不包括视觉检测。未来的改进可以通过整合视觉传感器，使机器人能够检测其视野内的威胁，消除物理访问每个巡逻点的需要。这将产生更灵活的巡逻策略，机器人可以优化路线以实现视觉覆盖，而不是访问固定点。

然而，采用基于视觉的方法会带来挑战。需要新的路径规划算法，以确保在最大化视觉覆盖的同时高效导航。此外，虽然更大的视野可能会减少持续运动的需要，但它可能会增加计算复杂性。

**5**.**2** 障碍物感知和覆盖优化

另一个潜在的研究方向是增强障碍物丰富的环境中的巡逻策略。墙壁或房间可以在机器人视野中创建盲点，使巡逻更具挑战性。为了保持有效覆盖，机器人必须绕过障碍物并识别阻塞区域。在多机器人系统中，通讯对于最小化盲点至关重要，各机器人之间需要协作巡逻。

此外，机器人可以实时动态调整其巡逻路径，以确保更灵活和适应性策略。

**参考文献**

[1] Parker L E. Current research in multi-robot systems[J]. International Journal of Advanced Robotic Systems, 2003, 1(1): 5-15.

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

[3] Parker L E. ALLIANCE: An architecture for fault tolerant multi-robot cooperation[J]. IEEE Transactions on Robotics and Automation, 1998, 14(2): 220-240.

[4] Thrun S, Burgard W, Fox D. Probabilistic Robotics[M]. Cambridge: MIT Press, 2005: 1-647.

[5] Portugal D, Rocha R P. Distributed multi-robot patrol: A scalable and fault-tolerant framework[J]. Robotics and Autonomous Systems, 2013, 61(12): 1572-1587.

[6] Xie J, Zhou R, Luo J, et al. Hybrid Partition-Based Patrolling Scheme for Maritime Area Patrol with Multiple Cooperative Unmanned Surface Vehicles[J]. Journal of Marine Science and Engineering, 2020, 8(11): 936.

[7] Simmons R, Apfelbaum D, Burgard W, et al. Coordinated deployment of multiple, heterogeneous robots[A]. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems[C]. Takamatsu: IEEE, 2000: 2254-2260.

[8] Elmaliach Y, Agmon N, Kaminka G A. Multi-robot area patrol under frequency constraints[J]. Annals of Mathematics and Artificial Intelligence, 2009, 57(3): 293-320.

[9] Sea V, Sugiyama A, Sugawara T. Frequency-Based Multi-agent Patrolling Model and Its Area Partitioning Solution Method for Balanced Workload[A]. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research[C]. Cham: Springer, 2018: 530-545.

[10] Sugiyama A, Sea V, Sugawara T. Effective Task Allocation by Enhancing Divisional Cooperation in Multi-Agent Continuous Patrolling Tasks[A]. In: 2016 IEEE 28th International Conference on Tools with Artificial Intelligence[C]. San Jose: IEEE, 2016: 33-40.

[11] Choset H, Lynch K M, Hutchinson S, et al. Principles of Robot Motion: Theory, Algorithms, and Implementations[M]. Cambridge: MIT Press, 2005: 1-603.

[12] LaValle S M. Planning Algorithms[M]. Cambridge: Cambridge University Press, 2006: 1-842.

[13] Burgard W, Moors M, Stachniss C, et al. Coordinated multi-robot exploration[J]. IEEE Transactions on Robotics, 2005, 21(3): 376-386.

[14] Xiang Z, Chen Z, Gao X, et al. Solving Large-Scale TSP Using a Fast Wedging Insertion Partitioning Approach[J]. Mathematical Problems in Engineering, 2015, 2015: 854218.

[15] Liu J, Xi B, Chen S, et al. The Path Planning Study of Autonomous Patrol Robot based on Modified Astar Algorithm and Genetic Algorithm[A]. In: 2022 34th Chinese Control and Decision Conference[C]. Hefei: IEEE, 2022: 4713-4718.

[16] Nikolaev A, Batsyn M. Branch-and-Bound Algorithm for Symmetric Travelling Salesman Problem[M]. Cham: Springer, 2018: 1-15.

[17] Sak T, Wainer J, Goldenstein S K. Probabilistic Multiagent Patrolling[A]. In: Advances in Artificial Intelligence - SBIA 2008[C]. Berlin: Springer, 2008: 124-133.

[18] Thrun S. Robotic mapping: A survey[A]. In: Exploring Artificial Intelligence in the New Millennium[C]. San Francisco: Morgan Kaufmann, 2002: 1-35.

[19] LaValle S M, Kuffner J J. Randomized kinodynamic planning[J]. The International Journal of Robotics Research, 2001, 20(5): 378-400.

[20] Gerkey B P, Matarić M J. Sold! Auction methods for multi-robot coordination[J]. IEEE Transactions on Robotics and Automation, 2002, 18(5): 758-768.

[21] Zlot R, Stentz A, Dias M B, et al. Multi-robot exploration controlled by a market economy[A]. In: IEEE International Conference on Robotics and Automation[C]. Washington: IEEE, 2002: 3016-3023.

[22] Dias M B, Zlot R, Kalra N, et al. Opportunistic optimization for market-based multi-robot control[A]. In: IEEE/RSJ International Conference on Intelligent Robots and Systems[C]. Beijing: IEEE, 2006: 2710-2717.

[23] Kuipers B, Byun Y T. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations[J]. Robotics and Autonomous Systems, 1991, 8(1-2): 47-63.

[24] Borenstein J, Koren Y. The vector field histogram-fast obstacle avoidance for mobile robots[J]. IEEE Transactions on Robotics and Automation, 1991, 7(3): 278-288.

[25] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots[J]. The International Journal of Robotics Research, 1986, 5(1): 90-98.

[26] Agmon N, Hazon N, Kaminka G A. Multi-robot perimeter patrol in adversarial settings[A]. In: IEEE International Conference on Robotics and Automation[C]. Pasadena: IEEE, 2008: 2339-2345.

[27] Sariel S, Balch T, Erdogan N. Efficient multi-robot search for a moving target[J]. The International Journal of Robotics Research, 2008, 27(2): 201-219.

[28] Michael N, Zavlanos M M, Kumar V, et al. Distributed multi-robot coordination using auction-based task allocation[A]. In: IEEE International Conference on Robotics and Automation[C]. Shanghai: IEEE, 2011: 3862-3867.

[29] Echefu L, Alam T, Newaz A A R. Randomized Multi-Robot Patrolling with Unidirectional Visibility[A]. In: 2024 21st International Conference on Ubiquitous Robots[C]. New York: IEEE, 2024: 324-329.

[30] Hollinger G A, Singh S. Multi-robot coordination with periodic connectivity[J]. IEEE International Conference on Robotics and Automation, 2012, 48(6): 852-865.

[31] Thrun S, Burgard W, Fox D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping[A]. In: IEEE International Conference on Robotics and Automation[C]. San Francisco: IEEE, 2000: 321-328.

[32] Latombe J C. Robot Motion Planning[M]. Boston: Kluwer Academic Publishers, 1991: 1-538.

[33] Kavraki L E, Svestka P, Latombe J C, et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces[J]. IEEE Transactions on Robotics and Automation, 1996, 12(4): 566-580.

[34] Fiorini P, Shiller Z. Motion planning in dynamic environments using velocity obstacles[J]. The International Journal of Robotics Research, 1998, 17(7): 760-772.

[35] Choset H, Lynch K M, Hutchinson S, et al. Principles of Robot Motion: Theory, Algorithms, and Implementations[M]. Cambridge: MIT Press, 2005: 1-603.

[36] Nemhauser G L, Wolsey L A, Fisher M L. An analysis of approximations for maximizing submodular set functions—I[J]. Mathematical Programming, 1978, 14(1): 265-294.

[37] Hazra T K, Hore A. A comparative study of Travelling Salesman Problem and solution using different algorithm design techniques[A]. In: 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference[C]. Vancouver: IEEE, 2016: 1-7.

[38] Nemhauser, G.L., Wolsey, L.A., Fisher, M.L．An analysis of approximations for maximizing submodular set functions—I[J]．Mathematical Programming，1978，14：265-294．

致谢

在本研究的整个过程中，我有幸得到多位老师的悉心指导和支持，在此谨向他们表示诚挚的感谢。

首先，衷心感谢我的指导老师郑裕基老师。在课题选题、研究推进、论文撰写等各个阶段，郑老师都给予了我非常宝贵的指导。他总能准确地指出问题的关键所在，帮助我不断优化研究方法、理清研究思路。在每一次讨论中，我都能从郑老师的建议中获得启发，进一步提升我的学术表达和逻辑思维能力。

同时，我也要感谢我的副指导老师张晴老师。张老师在整个研究过程中始终给予我关心和支持，尤其在实验设计和阶段性成果的分析过程中，她提出了许多细致而务实的建议，对本研究的完善起到了积极的推动作用。

此外，我还要特别感谢新加坡国立大学的指导老师 Ong Chong Jin 教授。在我赴新加坡交流学习期间，Ong 教授每周都组织例会，耐心听取我的研究汇报，并提出了许多具有方向性的意见。他对方法论的理解与把握，对我构建研究框架起到了重要的引导作用。他严谨又开放的科研态度，对我影响深远。

我还要感谢南方科技大学和新加坡国立大学提供的联合培养与交流学习机会。这段跨文化的学习经历不仅拓展了我的视野，也让我对科研工作有了更深入的理解。两所高校良好的科研环境和丰富的资源为本研究的顺利开展提供了有力保障。

感谢实验室的师兄师姐和同学们，在研究过程中给予我帮助与鼓励。在面对挑战与压力时，是你们的支持让我能够坚持下去，顺利完成各项任务。

最后，感谢我的家人一直以来的理解与支持，是他们在背后默默给予我鼓励和力量，让我能够安心投入到学术研究中。

本研究的完成凝聚了许多人的智慧与帮助，在此一并致以诚挚的谢意。

附录

符号列表

|  |  |  |
| --- | --- | --- |
|  | = | 节点i和j之间的转移概率 |
|  | = | 自上次访问节点j以来的时间间隔 |
|  | = | 节点i和j之间的欧几里得距离 |
|  | = | 时间权重系数 |
|  | = | 点a和b之间的启发式距离 |
|  | = | 路径规划的动态规划成本函数 |
|  | = | 从节点i到j的转移权重因子 |
|  | = | 巡逻点i的坐标 |
|  | = | A\*路径规划中的起点和终点节点 |
|  | = | 障碍物位置集合 |
|  | = | K均值聚类的目标函数 |
|  | = | K均值聚类中的聚类 |
|  | = | 聚类的质心 |
|  | = | 点x和质心之间的欧几里得距离平方 |
|  | = | 数据点集， |
|  | = | 聚类数量 |
|  | = | 聚类的选定种子集 |
|  | = | 具有最大欧几里得距离的两个初始种子点 |
|  | = | 最大最小策略中选择的下一个种子 |
|  | = | 距离矩阵中点和之间的欧几里得距离 |
|  | = | 已分配点集 |
|  | = | 为下一个簇分配选择的最佳未分配点 |
|  | = | 为点分配而选择的最佳簇 |
|  | = | 分配过程中观察到的最小总距离增加 |
|  | = | 分配新点前聚类的总距离 |

代码实现

以下代码是包含巡逻点的生成，两种TSP巡逻的实现（nearest\_neighbor和dynamic方法）。传统k-means分区方法实现，提出的最小增量聚类方法实现以及分区优化实现。此外还包含图像绘制以及分区结果的数据分析。修改机器人数量以及顶点数量并运行，可以直接生成对应的分区实现图像。

clear all;

% 参数设置

m = 4; % 机器人数量

n = 20; % 顶点数量

MAXc = 300; % 2D空间大小

vertices = [196.529401, 244.285448;

48.783521, 73.057491;

35.699304, 278.779087;

149.509216, 104.99513;

287.923188, 58.978575;

102.115718, 75.325157;

175.580325, 184.813403;

67.143582, 141.986655;

225.380118, 105.497852;

76.528535, 249.248588;

151.787115, 175.579227;

209.723017, 164.917082;

267.270976, 275.158099;

287.787428, 85.751706;

164.164659, 227.160069;

41.587333, 226.118728;

44.788202, 114.133754;

77.252476, 170.346492;

252.215177, 22.756287;

76.284654, 16.185036

];

max\_factor = 150;

dist\_matrix = squareform(pdist(vertices));

best\_partition = partition\_points\_kmeans(vertices, m)

% 使用 tsp\_nearest\_neighbor

% 2. 对每个分区使用TSP求解最优路径

bestPaths\_nn = cell(m, 1);

initialDistances\_nn = zeros(1, m);

for i = 1:m

subgraph = dist\_matrix(best\_partition{i}, best\_partition{i});

[bestPaths\_nn{i}, initialDistances\_nn(i)] = tsp\_nearest\_neighbor(subgraph);

end

disp(bestPaths\_nn);

% 3. 使用 tsp\_dynamic\_programming

% 2. 对每个分区使用TSP求解最优路径

bestPaths\_dp = cell(m, 1);

initialDistances\_dp = zeros(1, m);

for i = 1:m

subgraph = dist\_matrix(best\_partition{i}, best\_partition{i});

[bestPaths\_dp{i}, initialDistances\_dp(i)] = tsp\_dynamic\_programming(subgraph);

end

clear figure;

figure;

hold on;

colors = ['r', 'g', 'b','c','m']; % 预定义颜色序列

axis([0 300 -300 0]); % 固定坐标轴范围

set(gca, 'YDir', 'normal'); % 保持y轴方向（原始数据y∈0-300）

for i = 1:m

% 获取当前分区的点集（注意MATLAB索引转换）

partition\_points1 = best\_partition{i} ; % 原始索引0-based转1-based

sub\_vertices = vertices(partition\_points1, :);

% 选择颜色（循环使用预定义颜色）

color = colors(mod(i-1, length(colors)) + 1);

% 绘制当前分区的所有点

scatter(sub\_vertices(:,1), -sub\_vertices(:,2), ... % y坐标取反

80, ... % 点大小

'Marker', 'o', ...

'MarkerFaceColor', color, ...

'MarkerEdgeColor', 'k'); % 黑色边框

% 标注点序号（显示原始0-based索引）

for k = 1:size(sub\_vertices, 1)

text(sub\_vertices(k,1) + 5, -sub\_vertices(k,2) + 5, ... % 偏移量增大

num2str(best\_partition{i}(k)), ... % 原始0-based索引

'FontSize', 10, ...

'Color', 'k', ...

'FontWeight', 'bold', ...

'HorizontalAlignment', 'left');

end

end

clear figure

figure;

hold on;

colors = ['r', 'g', 'b','c','m'];

for i = 1:m

path = bestPaths\_dp{i};

sub\_vertices = vertices(best\_partition{i}, :);

for j = 1:length(path)-1

plot([sub\_vertices(path(j), 1), sub\_vertices(path(j+1), 1)], ...

-[sub\_vertices(path(j), 2), sub\_vertices(path(j+1), 2)], ...

colors(i), 'LineWidth', 2);

end

for k = 1:size(sub\_vertices, 1)

% 绘制黑色小圆圈

plot(sub\_vertices(k, 1), -sub\_vertices(k, 2), 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 6);

% 标注点的序号，添加偏移量避免与圆圈重叠

text(sub\_vertices(k, 1) + 0.05, -(sub\_vertices(k, 2) + 0.05), num2str(best\_partition{i}(k)), ...

'FontSize', 12, 'Color', 'k', 'FontWeight', 'bold', 'HorizontalAlignment', 'left', 'VerticalAlignment', 'bottom');

end

end

% 图形美化

grid on;

xlabel('X Coordinate');

ylabel('Y Coordinate(Inverted)');

title('TSP with partition');

hold off;

[optimizedPartition\_nn, min\_smax\_nn] = optimizePartition(best\_partition, dist\_matrix);

[optimizedPartition\_dp, min\_smax\_dp] = optimizePartition(best\_partition, dist\_matrix);

% 3. 使用 tsp\_dynamic\_programming

% 2. 对每个分区使用TSP求解最优路径

% 显示优化后的路径总距离

disp('Total Distance After Improvement using Nearest Neighbor:');

for i = 1:m

subgraph = dist\_matrix(optimizedPartition\_nn{i}, optimizedPartition\_nn{i});

[~, dist] = tsp\_nearest\_neighbor(subgraph);

fprintf('Total Distance of Group %d : %f\n', i, dist);

end

real\_min\_smax\_dp=0;

disp('Total Distance After Improvement using Dynamic Programming:');

for i = 1:m

subgraph = dist\_matrix(optimizedPartition\_dp{i}, optimizedPartition\_dp{i});

[~, dist] = tsp\_dynamic\_programming(subgraph);

fprintf('Total Distance of Group %d : %f\n', i, dist);

if dist>real\_min\_smax\_dp

real\_min\_smax\_dp = dist;

end

end

paths\_m = cell(1, m);

distances\_m = cell(1, m);

probabilities\_m = cell(1, m);

for i = 1:m

% 提取子图的距离矩阵

subgraph = dist\_matrix(best\_partition{i}, best\_partition{i});

% 获取符合条件的路径、距离和概率

[paths\_m{i}, distances\_m{i}, probabilities\_m{i}] = tsp\_nearest\_neighbor\_with\_markov(subgraph, max\_factor);

end

disp(distances\_m);

function [best\_path, best\_dist] = tsp\_nearest\_neighbor(dist\_matrix)

num\_cities = size(dist\_matrix, 1);

visited = false(1, num\_cities); % 记录是否访问过

best\_path = zeros(1, num\_cities); % 初始化路径

current\_city = 1; % 从第一个城市开始

best\_path(1) = current\_city;

visited(current\_city) = true;

best\_dist = 0;

for i = 2:num\_cities

% 找到距离最近的未访问的城市

distances = dist\_matrix(current\_city, :);

distances(visited) = inf; % 忽略已经访问过的城市

[min\_dist, next\_city] = min(distances);

best\_path(i) = next\_city;

best\_dist = best\_dist + min\_dist;

visited(next\_city) = true;

current\_city = next\_city;

end

% 返回到起点，闭合回路

best\_dist = best\_dist + dist\_matrix(current\_city, best\_path(1));

best\_path = [best\_path, best\_path(1)]; % 添加起点以形成闭合回路

end

function [best\_path, best\_dist] = tsp\_dynamic\_programming(dist\_matrix)

num\_cities = size(dist\_matrix, 1);

num\_subsets = 2^num\_cities;

% 初始化 DP 表

inf\_val = realmax; % 用于初始化距离矩阵的无穷大值

dp = inf(num\_subsets, num\_cities); % DP表，dp(S, i) 表示从起点到达子集 S 中的城市 i 的最短路径

parent = zeros(num\_subsets, num\_cities); % 用于存储路径的前驱城市

% 初始化

dp(2, 1) = 0; % 从起点城市 1 出发的距离为 0

% 填充 DP 表

for subset = 1:(num\_subsets - 1)

for u = 1:num\_cities

if bitget(subset, u) % 如果城市 u 在子集 subset 中

for v = 1:num\_cities

if v ~= u && bitget(subset, v) % 如果城市 v 在子集 subset 中且 v 不等于 u

prev\_subset = subset - bitget(subset, u) \* 2^(u - 1);

if dp(prev\_subset + 1, v) + dist\_matrix(v, u) < dp(subset + 1, u)

dp(subset + 1, u) = dp(prev\_subset + 1, v) + dist\_matrix(v, u);

parent(subset + 1, u) = v;

end

end

end

end

end

end

% 计算最优解

final\_subset = num\_subsets - 1;

best\_dist = inf\_val;

best\_end\_city = -1;

for i = 2:num\_cities

if dp(final\_subset + 1, i) + dist\_matrix(i, 1) < best\_dist

best\_dist = dp(final\_subset + 1, i) + dist\_matrix(i, 1);

best\_end\_city = i;

end

end

% 重建最优路径

best\_path = zeros(1, num\_cities);

current\_city = best\_end\_city;

subset = final\_subset;

for i = num\_cities:-1:1

best\_path(i) = current\_city;

prev\_city = parent(subset + 1, current\_city);

subset = subset - bitget(subset, current\_city) \* 2^(current\_city - 1);

current\_city = prev\_city;

end

best\_path = [best\_path, best\_path(1)]; % 形成闭合回路

end

function [optimizedPartition, min\_smax] = optimizePartition(partition, dist\_matrix)

numPartitions = length(partition);

partitionDistances = zeros(1, numPartitions);

% Initial distances for each partition

for i = 1:numPartitions

subgraph = dist\_matrix(partition{i}, partition{i});

[~, partitionDistances(i)] = tsp\_nearest\_neighbor(subgraph);

end

min\_smax = max(partitionDistances);

optimizedPartition = partition;

improved = true;

while improved

improved = false;

% Track if any swap leads to an improvement

improved\_in\_this\_round = false;

for i = 1:numPartitions

for j = i+1:numPartitions

[newPartition, newDistances] = swapPoints(partition, i, j, dist\_matrix);

new\_smax = max(newDistances);

if new\_smax < min\_smax

min\_smax = new\_smax;

optimizedPartition = newPartition;

partitionDistances = newDistances;

improved = true;

improved\_in\_this\_round = true;

end

end

end

end

end

function [newPartition, newDistances] = swapPoints(partition, i, j, dist\_matrix)

points\_i = partition{i};

points\_j = partition{j};

if length(points\_i) > 1 && length(points\_j) > 1

idx\_i = randi(length(points\_i));

idx\_j = randi(length(points\_j));

temp = points\_i(idx\_i);

points\_i(idx\_i) = points\_j(idx\_j);

points\_j(idx\_j) = temp;

end

newPartition = partition;

newPartition{i} = points\_i;

newPartition{j} = points\_j;

newDistances = zeros(1, length(partition));

for k = 1:length(partition)

subgraph = dist\_matrix(newPartition{k}, newPartition{k});

[~, newDistances(k)] = tsp\_nearest\_neighbor(subgraph);

end

end

% 计算欧几里得距离

function d = calcDistance(p1, p2)

d = sqrt(sum((p1 - p2).^2));

end

% 计算闭合路径的总长度

function [minLength, bestRoute] = closedRouteLength(points)

n = size(points, 1);

minLength = inf; % 初始化为无穷大

bestRoute = []; % 初始化最佳路线

% 生成所有排列

perm = perms(1:n);

for i = 1:size(perm, 1)

route = points(perm(i,:), :);

routeLength = 0;

% 计算闭合路线长度

for j = 1:n

if j == n

routeLength = routeLength + calcDistance(route(j,:), route(1,:)); % 回到起点

else

routeLength = routeLength + calcDistance(route(j,:), route(j+1,:));

end

end

if routeLength < minLength

minLength = routeLength;

bestRoute = perm(i,:); % 保存最佳路线

end

end

end

function [paths, distances, probabilities] = tsp\_nearest\_neighbor\_with\_markov(dist\_matrix, max\_factor)

num\_cities = size(dist\_matrix, 1);

all\_paths = {}; % 存储所有符合要求的路径

all\_distances = []; % 存储所有符合要求的路径的距离

% 原始的最优路径和最优距离

[best\_path, best\_dist] = tsp\_nearest\_neighbor(dist\_matrix);

threshold\_dist = best\_dist \* max\_factor;

% 使用邻近算法生成所有可能路径

for start\_city = 1:num\_cities

visited = false(1, num\_cities);

path = zeros(1, num\_cities + 1); % 增加一个元素来存储闭合路径

current\_city = start\_city;

path(1) = current\_city;

visited(current\_city) = true;

total\_dist = 0;

for i = 2:num\_cities

distances = dist\_matrix(current\_city, :);

distances(visited) = inf;

[min\_dist, next\_city] = min(distances);

path(i) = next\_city;

total\_dist = total\_dist + min\_dist;

visited(next\_city) = true;

current\_city = next\_city;

end

% 闭合路径

total\_dist = total\_dist + dist\_matrix(current\_city, path(1));

path(end) = path(1);

new\_path = true;

for i = 1: length(all\_paths)

current\_path = all\_paths{i};

if checkCyclicExcludeLast(path,current\_path)

new\_path = false;

end

end

% 仅记录符合距离阈值的路径

if total\_dist <= threshold\_dist && new\_path == true

all\_paths{end+1} = path; % 追加路径到cell数组

all\_distances(end+1) = total\_dist; % 追加距离到数组

end

end

% 根据路径距离计算概率

distances = all\_distances;

inverse\_distances = 1 ./ distances; % 使用倒数作为权重

probabilities = inverse\_distances / sum(inverse\_distances); % 归一化概率

paths = all\_paths;

disp (paths);

disp(probabilities);

disp(all\_distances);

end

function isCyclic = checkCyclicExcludeLast(arr1, arr2)

% Check if two arrays are cyclic or reverse cyclic permutations of each other, excluding the last element

if length(arr1) ~= length(arr2)

isCyclic = false;

return;

end

% Exclude the last element from both arrays

arr1\_excluded = arr1(1:end-1);

arr2\_excluded = arr2(1:end-1);

% Concatenate arr1\_excluded with itself

doubledArr = [arr1\_excluded, arr1\_excluded];

% Check if arr2\_excluded or its reverse appears as a subarray in doubledArr

isCyclic = contains(num2str(doubledArr), num2str(arr2\_excluded)) ;

% || ...

% contains(num2str(doubledArr), num2str(flip(arr2\_excluded)));

end

function groupIndices = partition\_points(vertices, n)

numPoints = size(vertices, 1);

if n == 1

groupIndices = {1:numPoints};

return;

end

% Step 1: 选择 n 个距离最远的点作为初始种子

disp('seedIndices')

seedIndices = select\_farthest\_points(vertices, n)

colors = ['r', 'g', 'b','c','m'];

figure;

hold on;

seedIndices\_0based = seedIndices - 1;

xlim([0 300]);

ylim([-300 0]); % 因y坐标取反，实际对应原始y∈[0,300]

for k = 1:size(vertices, 1)

original\_idx = k - 1;

is\_seed = ismember(original\_idx, seedIndices\_0based);

if is\_seed

color\_idx = find(seedIndices\_0based == original\_idx);

color = colors(mod(color\_idx-1, length(colors)) + 1); % 循环颜色

marker = 'o';

marker\_size = 8;

else

color = 'k';

marker = 'o';

marker\_size = 6;

end

plot(vertices(k,1), -vertices(k,2), ...

'Marker', marker, ...

'MarkerFaceColor', color, ...

'MarkerEdgeColor', color, ...

'MarkerSize', marker\_size);

text(vertices(k,1)+3, -vertices(k,2)+3, ...

num2str(k), ... % 显示1-based编号

'FontSize', 12, ...

'Color', 'k', ...

'HorizontalAlignment', 'left');

end

% 图形美化

axis equal;

grid on;

xlabel('X Coordinate');

ylabel('Y Coordinate (Inverted)');

title('Seed Points Visualization');

hold off;

groupIndices = cell(n, 1);

groupTotalDists = zeros(n, 1); % 每组当前的总距离

for i = 1:n

groupIndices{i} = seedIndices(i); % 初始种子点索引

end

% 标记已分配的点

assigned = false(numPoints, 1);

assigned(seedIndices) = true;

% 距离矩阵

dist\_matrix = squareform(pdist(vertices)); % numPoints x numPoints

% Step 2: 分配剩余的点

while ~all(assigned)

bestIncrease = inf;

bestGroup = -1;

bestPoint = -1;

% 遍历所有未分配点

for pt = 1:numPoints

if assigned(pt)

continue;

end

% 遍历每个组

for g = 1:n

currentGroup = groupIndices{g};

% 只计算该点到组内每个点的距离和

increase = sum(dist\_matrix(pt, currentGroup));

totalAfterAdd = groupTotalDists(g) + increase;

if totalAfterAdd < bestIncrease

bestIncrease = totalAfterAdd;

bestGroup = g;

bestPoint = pt;

end

end

end

% 把最优点加入最优组

groupIndices{bestGroup} = [groupIndices{bestGroup}, bestPoint];

groupTotalDists(bestGroup) = groupTotalDists(bestGroup) + ...

sum(dist\_matrix(bestPoint, groupIndices{bestGroup}(1:end-1))); % 新增的边

assigned(bestPoint) = true;

end

end

function seedIndices = select\_farthest\_points(vertices, n)

numPoints = size(vertices, 1);

seedIndices = zeros(n, 1);

% 计算距离矩阵

dist\_matrix = squareform(pdist(vertices));

% 选择前两个点，选择距离矩阵中值最大的两个点

[~, idx] = max(dist\_matrix(:)); % 找到最大距离的线性索引

[row, col] = ind2sub(size(dist\_matrix), idx); % 将线性索引转换为行列索引

% 确保不选择相同的点

seedIndices(1) = row;

seedIndices(2) = col;

% 从第三个点开始，选择与已选点集的平均欧几里得距离最大

for i = 3:n

maxAvgDist = -inf;

bestIdx = -1;

for j = 1:numPoints

if any(seedIndices == j) % 如果点已选，跳过

continue;

end

% 计算当前点到已选点集的欧几里得距离并计算平均距离

distances = dist\_matrix(j, seedIndices(1:i-1));

avgDist = mean(distances); % 计算平均距离

if avgDist > maxAvgDist

maxAvgDist = avgDist;

bestIdx = j;

end

end

seedIndices(i) = bestIdx;

end

end

function groupIndices = partition\_points\_kmeans(vertices, n)

% 传统K-Means算法对vertices进行n个分区

numPoints = size(vertices, 1);

% Step 1: 随机选择 n 个点作为初始中心

randIndices = randperm(numPoints, n);

centroids = vertices(randIndices, :);

% 迭代参数

max\_iters = 100; % 最大迭代次数

prev\_assignments = zeros(numPoints, 1); % 记录上一次分配情况

groupIndices = cell(n, 1); % 初始化分组

for i = 1:n

groupIndices{i} = [];

end

for iter = 1:max\_iters

% Step 2: 计算每个点到所有中心的欧几里得距离，并分配到最近的中心

distances = pdist2(vertices, centroids); % 计算所有点到中心的距离

[~, assignments] = min(distances, [], 2); % 找到最近的中心

% 如果分配结果没有变化，提前终止

if isequal(assignments, prev\_assignments)

break;

end

prev\_assignments = assignments;

% Step 3: 更新每个簇的中心（计算均值）

for k = 1:n

clusterPoints = vertices(assignments == k, :);

if ~isempty(clusterPoints)

centroids(k, :) = mean(clusterPoints, 1);

end

end

end

% Step 4: 生成最终分区结果

for k = 1:n

groupIndices{k} = find(assignments == k)';

end

end

function groupIndices = partition\_points\_min\_incremental(vertices, n)

% Min Incremental Clustering Method

numPoints = size(vertices, 1);

% Step 1: 随机选择 n 个点作为初始种子

rng('shuffle'); % 不固定随机种子，确保多次运行结果不同

seedIndices = randperm(numPoints, n);

centroids = vertices(seedIndices, :);

% 初始化 n 个分组索引

groupIndices = cell(n, 1);

for i = 1:n

groupIndices{i} = seedIndices(i); % 初始种子点索引

end

% 标记已分配的点

assigned = false(numPoints, 1);

assigned(seedIndices) = true;

% Step 2: 计算所有点的欧几里得距离矩阵

dist\_matrix = squareform(pdist(vertices));

% Step 3: 依次分配剩余的点

for i = 1:numPoints

if assigned(i)

continue;

end

minBestDist = inf;

bestGroup = 1;

% 计算加入每个组的增量代价

for j = 1:n

newGroupIndices = [groupIndices{j}, i];

newDistMatrix = dist\_matrix(newGroupIndices, newGroupIndices);

% 计算新分组的最短路径

[~, best\_dist] = tsp\_nearest\_neighbor(newDistMatrix);

% 选择最优分配

if best\_dist < minBestDist

minBestDist = best\_dist;

bestGroup = j;

end

end

% 分配到最优组

groupIndices{bestGroup} = [groupIndices{bestGroup}, i];

assigned(i) = true;

end

end