

涉密论文 公开论文

浙江大学

本科生毕业论文（设计）



题目 多机器人任务调度与路径规划问题研究

姓名与学号 钱尘涤 3150100820

指导教师 郑荣濠

年级与专业 15 级 自动化（电气）

所在学院 电气工程学院

提交日期 2019 年 5 月 24 日

浙江大学本科毕业生毕业设计（论文）装订内容检查表

(此表请装订在论文首页,由指导教师在答辩结束后填写)

学生姓名		学号		专业年级小班	
指导教师		职称		学生联系电话	
毕业设计题目				评定成绩	
“卓越计划”等实习单位：(没有填“无”)					

检 查 内 容		评价栏（打√）	
第一部分 毕 业 设 计 (论 文)	1. 封面（使用统一封面，建议选用蓝色）	较好	一般
	2. 毕业设计（论文）承诺书		
	3. 致谢		
	4. 中文摘要、英文摘要（均不超过 300 字）		
	5. 目录（每项内容要对应标注页码）		
	6. 正文（从正文开始标注页码）		
	7. 参考文献		
	8. 附录（可根据需要）		
	9. 作者简介		
	10. 毕业设计（论文）任务书		
	11. 毕业设计（论文）考核表		
	12. 毕业设计（论文）专家评阅意见表		
	13. 毕业设计（论文）现场答辩记录表		
第二部分 文 献 综 述 和 开 题 报 告	1. 指导教师对文献综述和开题报告具体内容要求		
	2. 文献综述		
	3. 开题报告		
	4. 外文翻译		
	5. 外文原文		
	6. 《浙江大学本科生文献综述和开题报告考核表》		

检查人（签名）: _____

检查日期: _____

浙江大学本科生毕业论文（设计）承诺书

1. 本人郑重地承诺所呈交的毕业论文（设计），是在指导教师的指导下严格按照学校和学院有关规定完成的。
2. 本人在毕业论文（设计）中除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。
3. 与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。
4. 本人承诺在毕业论文（设计）工作过程中没有伪造数据等行为。
5. 若在本毕业论文（设计）中有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。
6. 本人完全了解浙江大学有权保留并向有关部门或机构送交本论文（设计）的复印件和磁盘，允许本论文（设计）被查阅和借阅。本人授权浙江大学可以将本论文（设计）的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编本论文（设计）。

作者签名：

导师签名：

签字日期： 年 月 日 签字日期： 年 月 日

致 谢

经过了半年的努力，我的毕业设计总算尘埃落定，也预示着我的本科生涯即将告一段落。在这过程中，我要感谢许多直接或间接帮助过我的人。

首先我要感谢我的导师，郑荣濠老师，没有您的指导，我就不能明确自己毕设的研究方向和技术路线；没有您的帮助，我在实施过程中会遇到不少麻烦。同时，您教会了我做研究的态度，这也是我未来学习和科研道路上的宝贵财富。

其次，我要感谢我的室友，你们为我创造了良好的寝室环境，和互帮互助的学习氛围。

我还要感谢我的父母，你们支持我完成了本科的学习，生活中你们无私帮助，使我潜心学习，快乐生活。

最后我要感谢努力的自己，不辜负自己的四年大学时光。

摘要

多机器人任务调度与路径规划问题在许多领域都有着广泛的研究。本文就仓储环境下对该问题进行研究，并提出一定的划分方法将地图划分成子区域进行路径规划。本文完成了以下几个部分：1. 基于整数线性规划(ILP)的单子区域内的最优无碰撞路径规划；2. 全局环境下的任务分配与动态任务管理；3. 使用两种不同方案，实现多个子区域之间机器人的无碰撞交互，并对两者进行比较。本文提出的总体方案简单可行，在仓储环境下有一定的研究意义和应用价值。

关键词：多机器人系统；整数线性规划；路径规划

Abstract

Multi-robot task scheduling and path planning problem has a wide range of research in many fields. In this paper, the problem is studied under the warehouse environment, and certain partition methods are put forward to divide the map into subregions for path planning. In this paper, the following parts are completed: 1. Optimal and collision-free path planning within a subregion based on integer linear programming(ILP); 2. Task scheduling and dynamic task management under the global environment; 3. Two different methods to realize collision-free robot commuting among the subregions, and comparison between them. The overall scheme proposed in this paper is simple and feasible, and has certain research significance and application value in warehouse environment.

Keywords: multi-robot system; integer linear programming; path planning

目 录

第一部分 毕业论文（设计）

1	绪论	1
1.1	研究背景.....	1
1.2	本文主要工作.....	2
2	仓储环境模型的建立	2
2.1	地图模型.....	2
2.2	机器人模型.....	3
2.3	任务模型.....	4
2.4	子区域的设定.....	5
2.5	小结.....	6
3	基于 ILP 的多机器人路径规划	6
3.1	机器人位置的时序状态表示.....	6
3.2	整数线性规划的约束条件.....	7
3.2.1	起始位置和终点位置约束.....	7
3.2.2	路径连续性约束.....	8
3.2.3	固定障碍物的防碰撞约束.....	8
3.2.4	机器人碰撞约束.....	9
3.3	整数线性规划的目标函数.....	11
3.4	求解器 GUROBI 简介与整体模型的构建.....	12
3.5	仿真用例与结果分析.....	13
3.5.1	模型的完备性和正确性分析.....	13
3.5.2	不同模型对于求解速度的影响分析.....	16
3.5.3	障碍物数量对于求解速度的影响分析.....	18
3.6	小结.....	21
4	多任务管理	22
4.1	多机器人多任务分配.....	22
4.2	动态任务管理.....	23

4.3 仿真.....	24
4.4 小结.....	28
5 多区域之间的协同工作	28
5.1 方案一：子区域密集型.....	28
5.1.1 模型构建.....	28
5.1.2 当前区域到下一区域的出口选取.....	29
5.1.3 进入下一子区域的过程.....	31
5.1.4 仿真.....	33
5.2 方案二：子区域与单行线结合型.....	36
5.2.1 模型构建.....	36
5.2.2 离开和进入子区域的过程.....	36
5.2.3 碰撞及其防止措施.....	38
5.2.4 仿真.....	40
5.3 两种方案的对比.....	42
5.4 小结.....	46
6 总结与展望	46
6.1 总结.....	46
6.2 不足与展望.....	47
参考文献.....	49
附录.....	50
作者简历.....	51
本科生毕业设计（论文）任务书	
毕业论文（设计）考核表	
浙江大学本科生毕业设计（论文）专家评阅意见表	
浙江大学本科生毕业设计（论文）现场答辩记录表	

第二部分 文献综述和开题报告

指导教师对文献综述和开题报告具体要求

一、 文献综述.....	1
1 背景介绍.....	1
2 国内外研究现状.....	2
2.1 研究方向及进展.....	2
2.2 存在问题.....	4
3 研究展望.....	5
二、 开题报告.....	6
1 问题提出的背景.....	6
1.1 背景介绍.....	6
1.2 本研究的意义和目的.....	6
2 论文的主要内容和技术路线.....	7
2.1 主要研究内容.....	7
2.2 技术路线.....	7
2.3 可行性分析.....	11
3 研究计划进度安排及预期目标.....	12
3.1 进度安排.....	12
3.2 预期目标.....	12
三、 外文翻译.....	15
四、 外文原文.....	33
参考文献.....	49

浙江大学本科生文献综述和开题报告考核表

第一部分

毕业论文（设计）

1 絮论

1.1 研究背景

多机器人任务调度与路径规划问题在机器人领域一直是一个热门的研究方向，并且在诸多领域都有着广泛的应用，如仓储智能机器人、交通实时规划等。其中，仓储智能机器人集群有着广阔的应用前景和多种研究方向。近年来，电子商务的蓬勃发展，给物流行业带来了巨大的机遇和挑战。物流货物品种多、数量多、配送周期短，传统的人工分拣货物的方式不再适用于新型的物流仓库，因此新型的仓储智能机器人应运而生。

在2003年，意大利学者Raffaello D'Andrea便开始研究物流机器人Kiva systems。Kiva systems使用数百个移动机器人，利用控制系统完成整个仓储机器人集群的调度。产品不是存储在静态货架、流架或传送带上，而是存储在仓库中心的库存舱中，而操作员则站在周边的库存站[1]。亚马逊作为电商及物流企业的代表，在2012年收购了Kiva systems，并投入使用。

除此之外，国内各大企业也竞相发展仓储智能机器人集群。比如海康威视的仓储智能机器人集群，采用海康威视多种先进算法，将仓储地图转换成AGV能够识别的模型数据，在保证不出现拥堵的基础上，提供最短路径、避让控制、路径重新规划控制等多种处理机制[2]。另外，HRG智玲机器人是哈工大机器人集团全资子公司，其自主研发的Bee Robot智能仓储系统，由一台服务器和多台Bee Robot组成，可集成于既有的WMS系统，实现多台机器人协同工作，完成上架、拣货、补货、退货等流程[3]。

移动机器人路径规划方法可以概括为：基于模版匹配路径规划技术、基于人工势场路径规划技术、基于地图构建路径规划技术和基于人工智能的路径规划技术[4]。地图构建可以分为路标法和栅格法。其中栅格法比较适用于仓储环境下的移动机器人建模。将环境分解为相邻且不重复的单元(即栅格)，根据障碍物和可行路段的位置，构造出一定的仓储环境对应的栅格地图，在栅格地图上可以规划出机器人对应的最优路径。

在路径规划方面，比较传统的算法有A*算法，Dijkstra算法等，近年来比较流行的有蚁群算法、遗传算法等智能算法。在仓储环境中，需要一种简便的路

径规划算法，能够在满足机器人能够完成任务且机器人之间无碰撞的前提下，使得多机器人各自的路径最优化。

多机器人路径规划是NP-Hard问题，该方面研究还尚未成熟[5]。尤其是多最优目标的规划，目前的算法复杂度比较高。同时，如果考虑机器人运动学和远程通信等问题，仓储机器人的发展还有很长的路要走。

1.2 本文主要工作

本文基于MATLAB平台，就仓储环境下多机器人的路径规划做了仿真。实现功能如下：

1. 可以构建不同大小的地图，并设定不同密集程度的障碍物。自定义机器人和任务点的数量，并随机产生其位置，模拟各种可能的情况。
2. 将地图划分成多个子区域，利用整数线性规划(ILP)实现单个子区域内多机器人的无冲突且最优的路径规划。
3. 子区域之间使用两种不同的策略，实现子区域之间的无冲突路径规划。

利用MATLAB仿真结果，比较两种方法并证明其可行性。

2 仓储环境模型的建立

2.1 地图模型

栅格法是一种常见的地图建模方法，利用多边形或矩形栅格，对地图的障碍物作一定程度的近似。栅格越小，则地图细节越清晰，但是规划的复杂度会增加，反之亦然。由于仓储环境下，障碍物（即货架）本身接近矩形，因此栅格法非常适用于此类地图的构建。



图 2.1 亚马逊 Kiva systems[6]

对于矩形栅格，可以用一个二维数组表示地图，数组中用 0 表示机器人可以行走的路段，用 1 表示货架，用 0.5 表示人工拣货区。利用 MATLAB 的绘图功能，地图构建如下图 2.2，图中的黑色部分为空货架，作为障碍物，白色部分为可以自由行走的道路，灰色的方格为人工拣货区域。

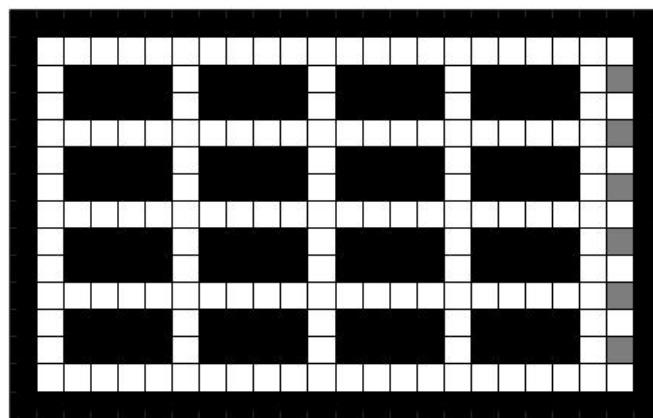


图 2.2 仓储地图环境的构建

2.2 机器人模型

机器人的数量可以自定义，对于机器人模型，作出以下假设：

1. 机器人的位置可以随机生成，但是不能与其他机器人的位置冲突。
2. 单个机器人在一个时间点只占据一个栅格。每一栅格不允许同时被两个及以上的机器人占据。

3. 单个机器人移动时，一个单位时间只能移动到相邻栅格，不允许走栅格的对角线。

4. 不考虑机器人加速、减速、转向等因素，认为机器人可以简单地在栅格之间进行位移。

5. 每个机器人的行为是等价的，即接到任务之后，移动到任务点，把货物连同货架一起移动到人工拣货区，再把空的货架放回原始位置。每个机器人有以下几种状态：空闲、前往任务点、前往拣货区、送回空货架，每次执行一个任务都依次变更以上几种状态。

6. 没有运送货架的机器人，允许其从货架下方通过，正在运送货架的机器人则只能走无障碍的道路（即空白区域），参考下图 Kiva systems 单个机器人的结构。



图 2.3 运送货架的单个机器人[7]

2.3 任务模型

任务的数量可以自定义，对于任务模型，作出以下假设：

1. 任务的位置可以随机生成，但是必须生成在货架上，即图 2.2 中除了最外圈的其他黑色部分。
2. 任务的数量不能过多，应小于或等于空货架的数量。
3. 一个任务只能由一个机器人执行，且一个机器人一次性只能执行一个任务，不允许机器人中途接管其他任务。

4. 任务所在的货架被机器人搬走之后，该位置立即成为可通行区域，即在地图上变为白色。

2.4 子区域的设定

对于较大的地图而言，需要将地图分割成若干个子区域。本文采用两种不同的划分方法，如下图 2.4 和图 2.5，下文会着重介绍。每个子区域内部对机器人进行规划，规划多个子区域比全局规划更加节省计算力。

每个子区域存储的信息包括：该子区域在全局地图上的坐标范围，该子区域内的机器人信息，该子区域到其他子区域的出口位置。

如果子地图过大，则进行规划的计算仍然比较费时；如果子地图过小，则机器人会更加频繁地进入新的子地图，导致全局环境下所有子地图重新规划的次数增加，同时子地图过小会使机器人最优路径的范围变小。因此需要合理选择子地图的大小。

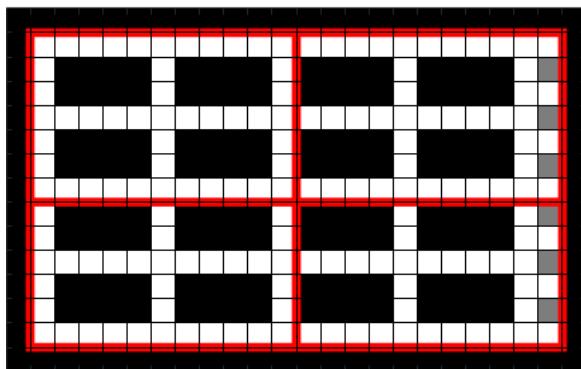


图 2.4 子地图划分例一

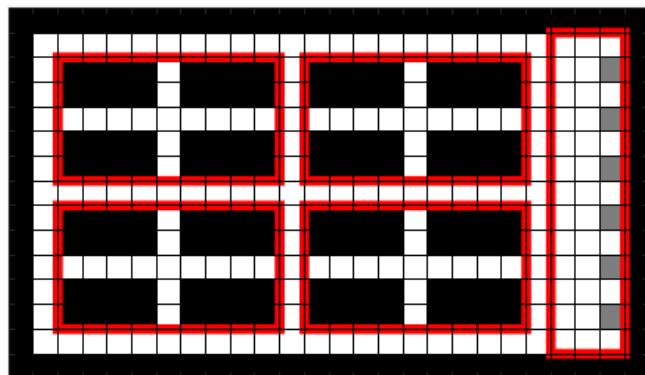


图 2.5 子地图划分例二

2.5 小结

本文设定了简单的栅格地图模型，并规定了机器人和货物的行为，为后续软件平台的仿真作了必要的铺垫。

但是，实际场景下的机器人和货物的行为并不会如此简单，考虑到机器人的加速、减速、转向、导航等因素，机器人的运动学和动力学模型等因素必须加以考虑[8]。且机器人的定位等因素需要实时更新，在控制上会有一些挑战。

因此，本文仅在理想化模型的基础上对多机器人的路径进行规划。

3 基于 ILP 的多机器人路径规划

3.1 机器人位置的时序状态表示

假设多机器人路径规划任务在 T 时间内完成，则整个栅格地图可以拓展为时长为 T 的扩张网络。扩张网络见下图 3.1，所有位置（即状态）经过时域的平移，形成了类似的时域扩张网络。

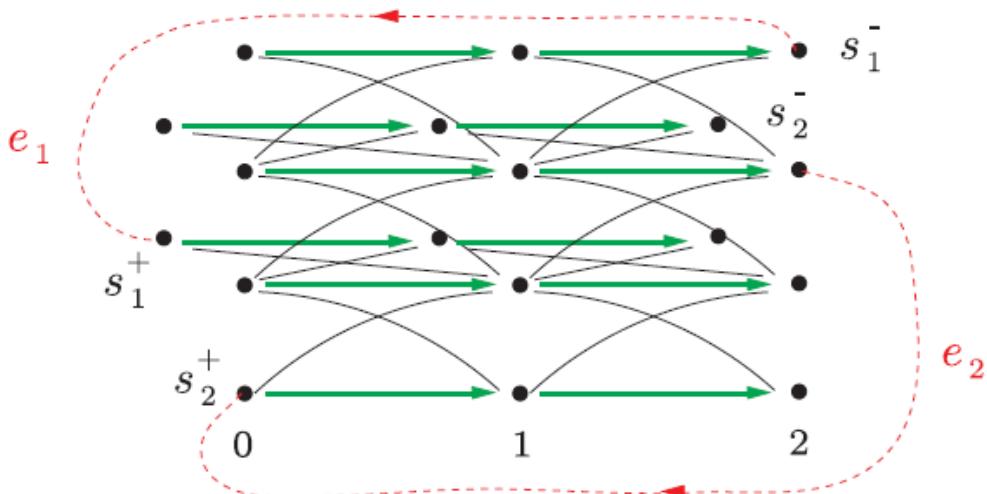


图 3.1 简洁的扩张网络 [9]

借助该方法，机器人的位置状态可以由以下两种方法表示：

1. 设定一个四维矩阵，假设某一机器人 r 在 t 时刻处于栅格地图的 $\{i, j\}$ 位置，则 $x_{i,j,r,t} = 1$ ，而对于整数对 $\{m, n\} \neq \{i, j\}$ ，必然有 $x_{m,n,r,t} = 0$ ，因为相同机器人不能在同一时间出现在其他位置。该矩阵的四个维度分别表示地图的行 i 、列

j 、机器人编号 r 和时间节点 t 。数组的元素取值范围只有 0 和 1，是一种 0-1 整数规划。该方法的优点在于比较简洁、直观，缺点在于冗余变量较多。

2. 设定一个三维矩阵，假设某一机器人 r 在 t 时刻处于栅格地图的 $\{i, j\}$ 位置，则 $x_{r,t} = \{i, j\}$ ，即 $x_{1,r,t} = i$ ，且 $x_{2,r,t} = j$ 。 $x_{1,r,t}$ 表示机器人 r 在 t 时刻的行坐标， $x_{2,r,t}$ 表示机器人 r 在 t 时刻的列坐标。即该三维矩阵的三个维度分别为机器人 r 的行列位置信息(该维度的大小必然为 2，用于存储行和列的位置)、机器人编号 r 和时间节点 t 。该方法同样比较直观，较于上一种方法大大减少了变量数量，但是不再是 0-1 规划。

为了减少程序的空间复杂度，本文采用第二种方法。

3.2 整数线性规划的约束条件

考虑多机器人的起始位置、终点位置、机器人碰撞约束、机器人路径连续等条件，本节详细介绍线性规划模型的约束。

3.2.1 起始位置和终点位置约束

起始位置约束：对于机器人 $r=1$ ，若规定其起始位置为 $\{s_x, s_y\}$ ，即在时间 $t=1$ 时处于 $\{s_x, s_y\}$ 位置，则根据上述的时序状态表示法，可以表示为

$$x_{1,1} = \{s_{1x}, s_{1y}\}$$

若推广到所有机器人，则对于总共 R 个机器人，对于 $\forall r \in \{1, 2, 3, \dots, R\}$ ，有

$$x_{r,1} = \{s_{rx}, s_{ry}\} \quad (1)$$

终点位置约束：假设完成任务所需的时间长度为 T (T 由人为规定，且 T 足够大，以保证完成任务)，则对于机器人 $r=1$ ，若其终点位置为 $\{e_x, e_y\}$ ，则

$$x_{1,T} = \{e_{1x}, e_{1y}\}$$

同样，推广到所有机器人，则对于总共 R 个机器人，对于 $\forall r \in \{1, 2, 3, \dots, R\}$ ，有

$$x_{r,T} = \{e_{rx}, e_{ry}\} \quad (2)$$

实际上，考虑到多个机器人的终点可能是同一个位置，而机器人碰撞避免的约束必须满足，因此本文的仿真程序中放弃终点位置约束，而把它写入线性规划模型的目标函数中，下文会详细介绍。

3.2.2 路径连续性约束

对于每一个机器人，其路径必须是连续的，并且只能走上下左右四个方向，不允许走左上、左下等斜的方格。

假设机器人在某一时间节点较于上一时间节点的横纵坐标的改变量分别为 dx 和 dy ，则 $dx, dy \in \{-1, 0, 1\}$ ，且 $dx \cdot dy = 0$ 。该约束可以转化为线性约束，即

$$\begin{aligned} -1 &\leq dx + dy \leq 1 \\ -1 &\leq dx - dy \leq 1 \end{aligned}$$

因此，对于总共 R 个机器人，对于 $\forall r \in \{1, 2, 3, \dots, R\}$ ，对于时间节点 $\forall t \in \{1, 2, 3, \dots, T-1\}$ ，以上约束可以表示为

$$\begin{aligned} -1 &\leq (x_{1,r,t+1} - x_{1,r,t}) + (x_{2,r,t+1} - x_{2,r,t}) \leq 1 \\ -1 &\leq (x_{1,r,t+1} - x_{1,r,t}) - (x_{2,r,t+1} - x_{2,r,t}) \leq 1 \end{aligned} \tag{3}$$

3.2.3 固定障碍物的防碰撞约束

对于固定的障碍物，一般情况下，规定机器人不得处于固定障碍物所处的位置。该约束的线性表示的思想可借鉴参考文献[10]。

对于一个地图(或子区域)中的所有障碍物，可对其进行标号，一个编号表示一个栅格内的障碍物，且不对多个障碍物进行合并。假设总共有 K 个障碍物， R 个机器人，时间长度为 T ，假定障碍物 k 的位置坐标用 $\{obs_{k,1}, obs_{k,2}\}$ 表示，则

对于 $\forall r \in \{1, 2, 3, \dots, R\}$ ，对于 $\forall k \in \{1, 2, 3, \dots, K\}$ ，对于 $\forall t \in \{1, 2, 3, \dots, T\}$ ，约束如下

$$\begin{aligned} x_{1,r,t} &\leq obs_{k,1} - 1 + M \cdot t_{r,k,t,1} \\ -x_{1,r,t} &\leq -obs_{k,1} - 1 + M \cdot t_{r,k,t,2} \\ x_{2,r,t} &\leq obs_{k,2} - 1 + M \cdot t_{r,k,t,3} \\ -x_{2,r,t} &\leq -obs_{k,2} - 1 + M \cdot t_{r,k,t,4} \\ \sum_{i=1}^4 t_{r,k,i} &\leq 3 \end{aligned} \tag{4}$$

假定 M 是一个足够大的数，且 $t_{r,k,i} \in \{0,1\}$ 。若某一个 t 为 1，则该项约束为软约束，否则为硬约束。上面四项约束至少有一项为硬约束，由此可以保证指定机器人和指定障碍物的 x 和 y 坐标至少有一项不同。另外， $t_{r,k,1}$ 和 $t_{r,k,2}$ ， $t_{r,k,3}$ 和 $t_{r,k,4}$ 不可能同时取 0。

事实上，根据第二章对机器人模型的规定，没有运送货架的机器人可以从货架底下穿过，因此对于此类机器人，不存在该约束。另外，每次规划时机器人的起点或终点可能为目标任务点，因此起点和终点也不考虑该约束。

3.2.4 机器人碰撞约束

机器人的碰撞有多种形式，包括相遇碰撞和迎面碰撞[9]。

相遇碰撞指的是两个或以上机器人在同一时间点出现在同一栅格。几种可能的情况如下：

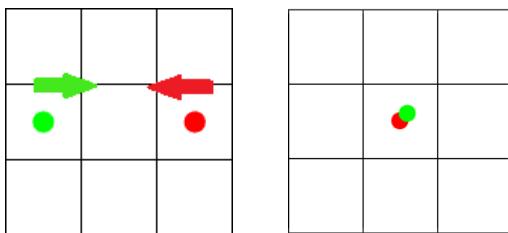


图 3.2 相向而行的相遇碰撞例

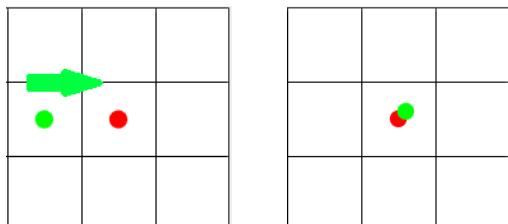


图 3.3 追尾的相遇碰撞例

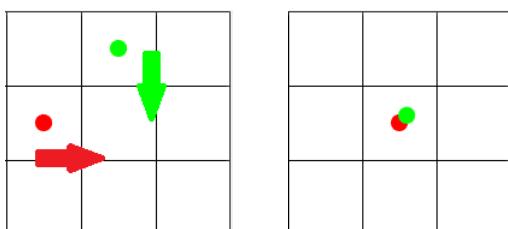


图 3.4 方向正交的相遇碰撞例

相遇碰撞一般有以上几种情况。为了避免多机器人之间的相遇碰撞，需要添加以下约束，同样用 $\{x_{1,r,t}, x_{2,r,t}\}$ 表示机器人 r 在时间 t 的坐标，则对于机器人 $\forall p, q \in \{1, 2, 3, \dots, R\}$ ，且 $p \neq q$ ，对于 $\forall t \in \{1, 2, 3, \dots, T\}$ ，有

$$\begin{aligned} x_{1,p,t} &\leq x_{1,q,t} - 1 + M \cdot t_{p,q,t,1} \\ -x_{1,p,t} &\leq -x_{1,q,t} - 1 + M \cdot t_{p,q,t,2} \\ x_{2,p,t} &\leq x_{2,q,t} - 1 + M \cdot t_{p,q,t,3} \\ -x_{2,p,t} &\leq -x_{2,q,t} - 1 + M \cdot t_{p,q,t,4} \\ \sum_{i=1}^4 t_{p,q,i} &\leq 3 \end{aligned} \quad (5)$$

M 是一个足够大的数，且 $t_{p,q,i} \in \{0, 1\}$ 。公式的物理意义同固定障碍物的防碰撞约束，只是把任意机器人与任意固定障碍物的关系改为任意两个机器人之间的关系。

除了相遇碰撞外，还有一种碰撞形式为迎面碰撞，见下图 3.5。本质是两个机器人在路径连续的前提下互换位置。这类碰撞不属于上一种碰撞，机器人不会在相同时间节点出现在同一栅格，但是机器人仍然产生碰撞，在仿真及现实中应当避免。

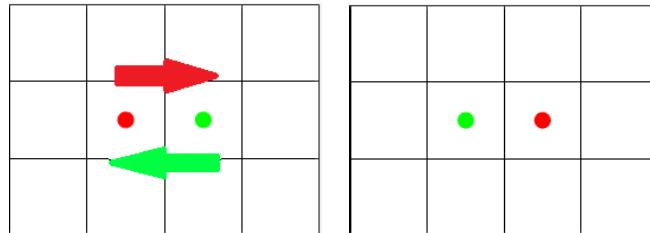


图 3.5 迎面碰撞例

解决迎面碰撞，其本质是不允许任意两个相邻的机器人交换位置。添加以下约束：对于机器人 $\forall p, q \in \{1, 2, 3, \dots, R\}$ ，且 $p \neq q$ ，对于 $\forall t \in \{1, 2, 3, \dots, T\}$ ，有

$$\begin{aligned}
x_{1,p,t} &\leq x_{1,q,t+1} - 1 + M \cdot t_{p,q,t,1} \\
-x_{1,p,t} &\leq -x_{1,q,t+1} - 1 + M \cdot t_{p,q,t,2} \\
x_{2,p,t} &\leq x_{2,q,t+1} - 1 + M \cdot t_{p,q,t,3} \\
-x_{2,p,t} &\leq -x_{2,q,t+1} - 1 + M \cdot t_{p,q,t,4} \\
x_{1,q,t} &\leq x_{1,p,t+1} - 1 + M \cdot t_{q,p,t,1} \\
-x_{1,q,t} &\leq -x_{1,p,t+1} - 1 + M \cdot t_{q,p,t,2} \\
x_{2,q,t} &\leq x_{2,p,t+1} - 1 + M \cdot t_{q,p,t,3} \\
-x_{2,q,t} &\leq -x_{2,p,t+1} - 1 + M \cdot t_{q,p,t,4} \\
\sum_{i=1}^4 t_{q,p,t,i} + t_{p,q,t,i} &\leq 7
\end{aligned} \tag{6}$$

该公式规定了机器人 p 在 t 时刻的位置不是机器人 q 在 $t+1$ 时刻的位置，或机器人 q 在 t 时刻的位置不是机器人 p 在 $t+1$ 时刻的位置，即防止相邻的机器人互换位置，从而避免迎面碰撞。

3.3 整数线性规划的目标函数

多机器人路径规划的目标可以是多样的，如完成时间最短、总路径最短等 [9]。本文采用的是总路径最短。

同时，在上文中已经提到，多个机器人的终点可能是同一栅格（如机器人都要经过该栅格前往另一子区域的情况），因此终点约束不考虑，而是在目标函数中得以实现。因此，要满足机器人尽可能到达自己的终点的情况下，多机器人总的路径尽可能短的条件，可以如下设定目标函数。

假设完成时间长度为 T ，有 R 个机器人，单个机器人 r 的终点表示为 $\{dest_{r,1}, dest_{r,2}\}$ ，则对于 $\forall t \in \{1, 2, 3, \dots, T\}$ ， $\forall r \in \{1, 2, 3, \dots, R\}$ ，目标函数为

$$\min \left[\sum_{r=1}^R \sum_{t=1}^T (x_{1,r,t} - dest_{r,1})^2 + (x_{2,r,t} - dest_{r,2})^2 \right]$$

即所有机器人在所有时间节点到对应目标位置的距离平方的和最小。这样的目标函数可以保证机器人朝目标位置运动，又可以避免与多个机器人目标位置相同的情况冲突，即机器人可以尽可能地靠近目标位置，但是如果多个机器人的目标位置相同，允许某一个或多个机器人不到达该位置。

事实上，这种目标函数具有二次项，求解器虽然可以求解非线性问题，但是效率会在一定程度上低于线性问题。对于本文的应用场景而言，规划的次数会比较多，因此以上目标函数需要尽可能线性化。

考虑到平方与绝对值在一定程度上可以等价，该目标函数的平方部分可以用绝对值代替。假定对于机器人 r 的目标位置 $\{dest_{r,1}, dest_{r,2}\}$ ，定义 2 个新的变量 $\{\omega_{r,1}, \omega_{r,2}\}$ ，使得对于 $\forall t \in \{1, 2, 3, \dots, T\}$ ， $\forall r \in \{1, 2, 3, \dots, R\}$ ，满足

$$\begin{aligned} |x_{1,r,t} - dest_{r,1}| &\leq \omega_{r,1} \\ |x_{2,r,t} - dest_{r,2}| &\leq \omega_{r,2} \end{aligned}$$

即

$$\begin{aligned} -\omega_{r,1} &\leq x_{1,r,t} - dest_{r,1} \leq \omega_{r,1} \\ -\omega_{r,2} &\leq x_{2,r,t} - dest_{r,2} \leq \omega_{r,2} \end{aligned} \tag{7}$$

即 $\{\omega_{r,1}, \omega_{r,2}\}$ 是机器人 r 在所有时间段距离目标位置的上限。目标函数只需所有机器人的上限的和最小，即

$$\min \left[\sum_{r=1}^R (\omega_{r,1} + \omega_{r,2}) \right] \tag{8}$$

就可以在保证效果相同的情况下将目标函数线性化，提高求解器的求解速度。

3.4 求解器 Gurobi 简介与整体模型的构建

Gurobi 是美国 Gurobi Optimization 公司开发的新一代大规模数学规划优化器，支持多平台，包括 Windows、Linux、Mac OS 等，并且与 C++、Python、MATLAB 等编程语言都有便捷接口。Gurobi 对在校师生提供免费版本。由下图 3.4 可见，Gurobi 在第三方优化器中有着很高的精度和求解速度，因此本文的仿真程序选用 Gurobi 作为求解器。

线性混合整数规划 MILP								
1 线程	CBC	CPLEX	GUROBI	SCIPC	SCIPS	XPRESS	MATLAB	SAS
速度比例	39	1.74	1	5.75	7.94	2	72.2	2.9
解决问题数量	53	87	87	83	76	86	32	84
4 线程	CBC	CPLEX	FSCIPC	FSCIPS	GUROBI	XPRESS	MIPCL	SAS
速度比例	34.8	1.5	9.9	12.1	1	1.66	7.29	3
解决问题数量	66	86	80	79	87	87	84	85
12 线程	CBC	CPLEX	FSCIPC	FSCIPS	GUROBI	XPRESS	MIPCL	SAS
速度比例	27	1.49	9.8	13	1	1.57	6.53	3.39
解决问题数量	69	87	78	76	87	87	82	82

速度比例为1是最快的速度，其他数值为该速度的倍数。

图 3.6 各种大规模优化器的速度比较[11]

在 MATLAB 的 Gurobi 接口中，求解整数线性规划问题需要构建如下的模型

$$\begin{aligned}
 & \min c^T x + \alpha \\
 & s.t. \\
 & Ax \leq b \\
 & l \leq x \leq u \\
 & x \in \mathbb{Z}
 \end{aligned} \tag{9}$$

因此需要构建的参数有优化目标(min 或 max 等)，目标函数的系数向量 c ，变量名 x ，常数向量 α ，系数矩阵 A ，约束向量 b ，变量的下限向量 l 和上限向量 u ，以及变量类型(整数、实数、0-1 变量等)。

参考线性规划结构(9)，根据上述约束(1)(2)(3)(4)(5)(6)(7)，以及目标函数(8)，可以构建完整的模型。

3.5 仿真用例与结果分析

3.5.1 模型的完备性和正确性分析

为了验证以上约束的完备性和正确性，此处列举了几个例子，来检验单个子区域内的多机器人路径规划算法。

例 1：设机器人个数为 2，起点分别为 [3,1] 和 [3,7]，终点分别为 [3,7] 和 [3,1]。地图构建如下：

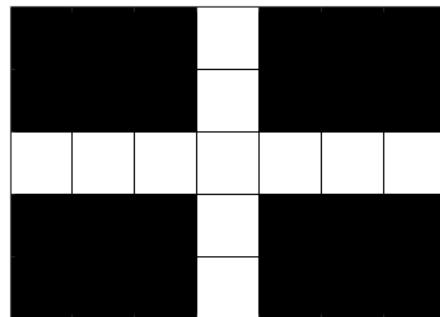


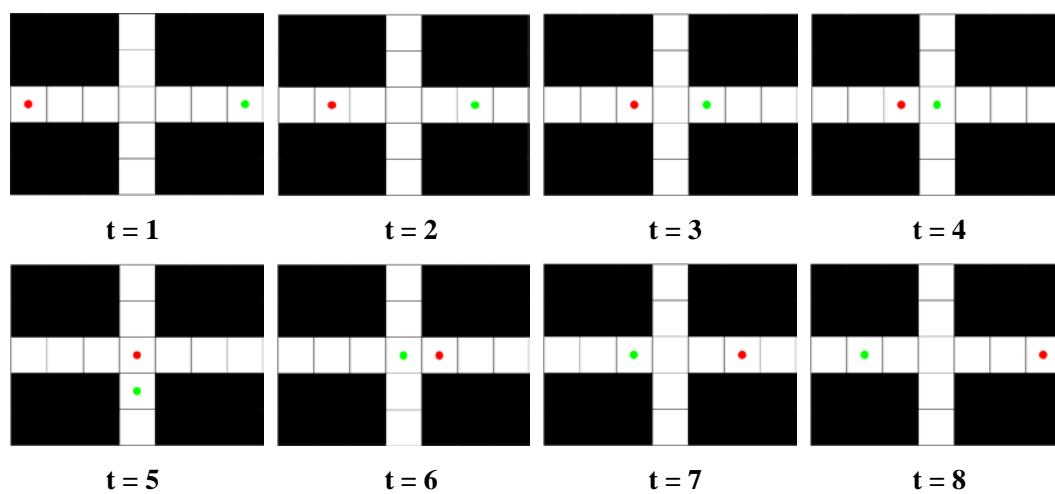
图 3.7 例 1、2 地图构建

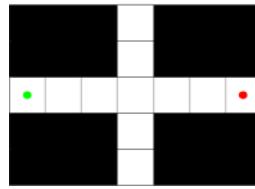
利用 MATLAB 构建模型并用 Gurobi 接口进行求解。求解结果如下：

status	'OPTIMAL'
versioninfo	1x1 struct
runtime	0.1037
objval	178
x	1900x1 double
slack	2421x1 double
poolobjbound	178
pool	1x2 struct
mipgap	0
objbound	178
objboundc	178
itercount	1009
baritercount	0
nodecount	56

图 3.8 例 1 求解结果

可见求解结果为“optimal”，大小为 5×7 的栅格地图求解时间为 0.1037 秒，求解速度比较理想。如果将解 x 经过处理，绘制到栅格地图上，则机器人路径如下：





$t = 9$

图 3.9 例 1 机器人路径

红色和绿色分别表示两个机器人，黑色的是不允许通过的障碍物，白色的是道路。从图 3.5.3 中可以直观地看到，机器人的路径满足连续且无碰撞，并且不会与障碍物进行碰撞。如果没有防止相遇碰撞的机制，则机器人会在点 [3,4] 上相撞。因此，本例可以证明机器人防相遇碰撞的机制具有可行性。

例 2：设机器人个数为 2，起点分别为 [3,1] 和 [3,6]，终点分别为 [3,6] 和 [3,1]。求解结果如下：

ch	status	'OPTIMAL'
E	versioninfo	1x1 struct
E	runtime	0.0917
E	objval	107
E	x	1688x1 double
E	slack	2151x1 double
E	poolobjbound	107
E	pool	1x2 struct
E	mipgap	0
E	objbound	107
E	objboundc	107
E	itercount	609
E	baritercount	0
E	nodecount	24

图 3.10 例 2 求解结果

由求解结果可知求解时间约为 0.0917 秒。机器人路径的绘制如下：

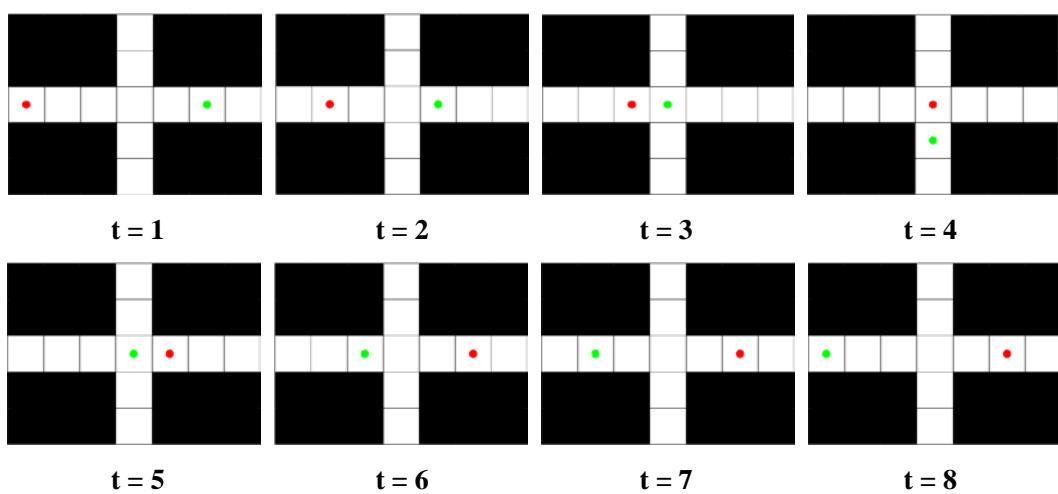


图 3.11 例 2 机器人路径

同样可见，机器人的路径是不冲突的。在本例中，如果没有机器人迎面碰撞的约束，机器人会在位置[3,3]和[3,4]上产生迎面碰撞，即简单的交换位置。

从以上两个例子中可以看到，机器人的行为能够满足路径连续、起点和终点约束、不与固定障碍物碰撞，以及两种机器人之间的碰撞的约束。

3.5.2 不同模型对于求解速度的影响分析

我们可以通过仿真程序比较有无二次项对求解速度的影响。以例 1 中的机器人数量及其初始/目标位置为例，如果目标函数有二次项，即目标函数采用(10)，求解结果如下：

ch	status	'OPTIMAL'
	versioninfo	1x1 struct
	runtime	0.5680
	objval	-2476
x	x	1864x1 double
	slack	2349x1 double
	poolobjbound	-2476
pool	pool	1x1 struct
	mipgap	0
	objbound	-2476
	objboundc	-2476
	itercount	44589
	baritercount	0
	nodecount	2435

图 3.12 例 1 的带二次项的求解结果

可见，求解时间大约 0.5680 秒，比例 1 的求解结果慢。

如果可以扩大地图，并增加机器人数，求解速度的差距会更加明显。

例 3：若设置机器人数为 4，起点为[3,1], [3,9], [1,10], [6,5]；终点为[3,9], [6,1], [1,5], [9,15]，地图构建如下：

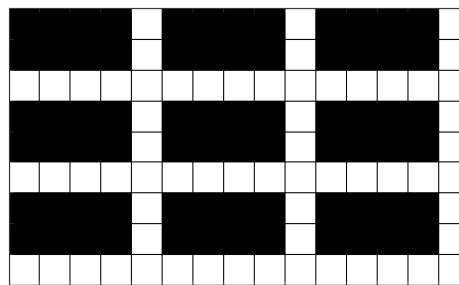


图 3.13 例 3 地图构建

带二次项的仿真结果如下：

ch	status	'OPTIMAL'
versioninfo	1x1 struct	
runtime	153.6580	
objval	-43684	
x	17200x1 double	
slack	21506x1 double	
poolobjbound	-4.3686e+04	
pool	1x4 struct	
mipgap	4.5783e-05	
objbound	-4.3686e+04	
objboundc	-4.3686e+04	
itercount	2292508	
baritercount	0	
nodecount	80572	

图 3.14 例 3 带二次项模型的仿真结果

机器人的路径如下：

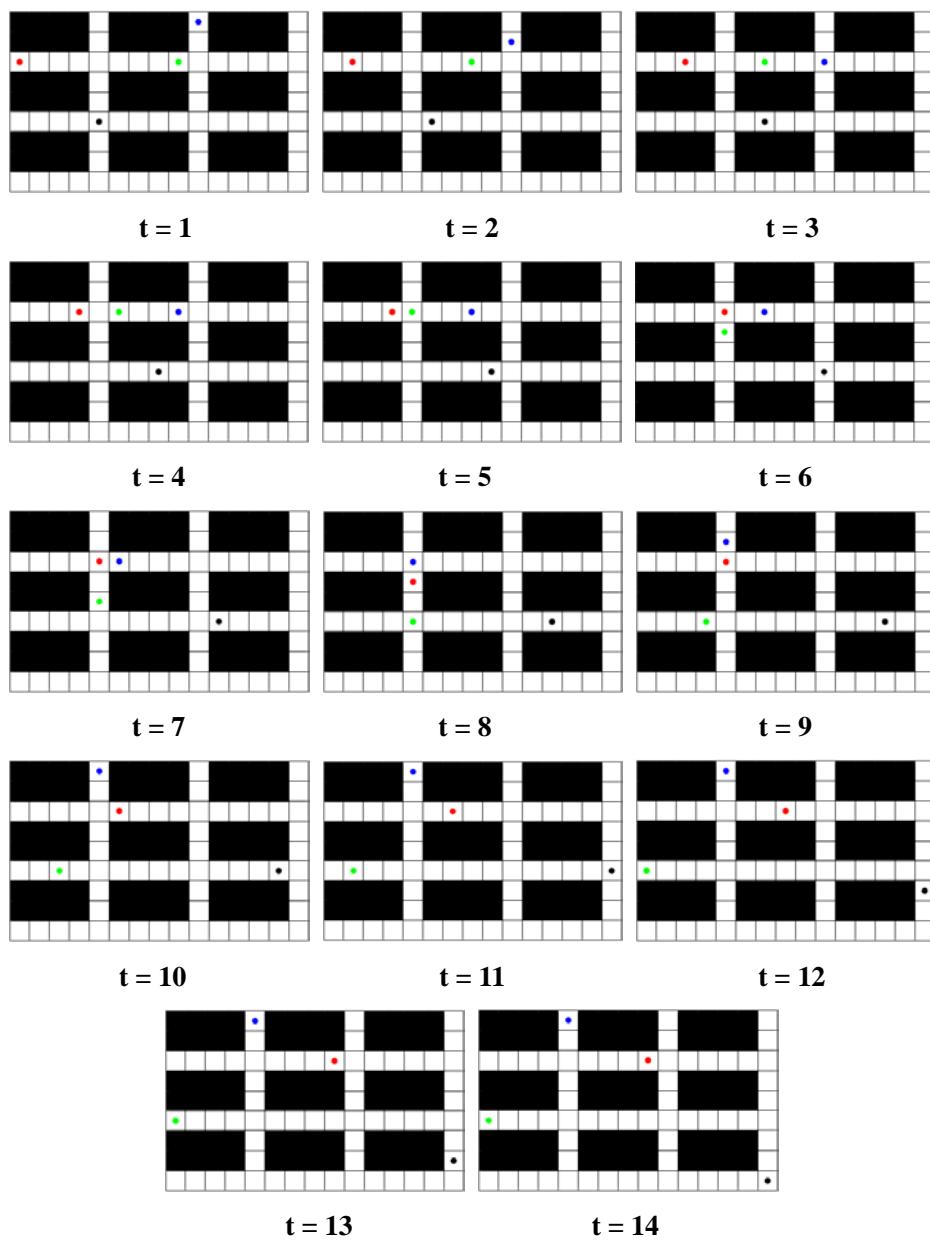


图 3.15 例 3 机器人路径

同样的数据，如果使用目标函数不带二次项的模型进行仿真，则结果如下：

ch	status	'OPTIMAL'
E	versioninfo	1x1 struct
	runtime	29.3474
	objval	1175
	x	17312x1 double
	slack	21730x1 double
	poolobjbound	1175
E	pool	1x5 struct
	mipgap	0
	objbound	1175
	objboundc	1175
	itercount	149876
	baritercount	0
	nodecount	5962

图 3.16 例 3 线性模型的仿真结果

机器人路径相同，不再展示图片。如果对于此例地图比较大，机器人数量偏多的情况，如果目标函数带有二次项，则求解时间达到了 153.7 秒，而不带二次项的求解时间只有 29.3 秒，差距相当明显。

为了进一步确认模型对于求解时间的影响，使用例 3 的地图布局和机器人设定，分别用二次项和线性的模型求解 10 次，统计求解时间如下

表 3.1 模型类型与求解时间关系统计表

模型 \ 试验次数	1	2	3	4	5	6	7	8	9	10
非线性	158.0730	212.7319	205.5099	204.3836	215.6798	215.2063	215.5280	212.0794	212.5854	207.0677
线性	18.9695	20.1737	21.2374	20.9344	20.5264	22.4971	23.4472	22.8009	23.1232	23.9026

由表中可以看到，带有非线性项的模型的求解速度明显大于线性模型。对于本文这种对于实时性要求比较高的仿真，对于求解时间也比较敏感，因此在构建模型时，应尽量线性化模型，使求解速度更快。

3.5.3 障碍物数量对于求解速度的影响分析

由 3.5.2 分析可知，地图较大、机器人数量较多时，求解速度会相对小地图、少量机器人而言更慢。如果随机减少一部分障碍物数量，求解速度可能会因此受影响。

例 4： 使用例 3 的机器人数量及其起始/目标位置，随机减少 40% 的障碍物数量，地图如下：

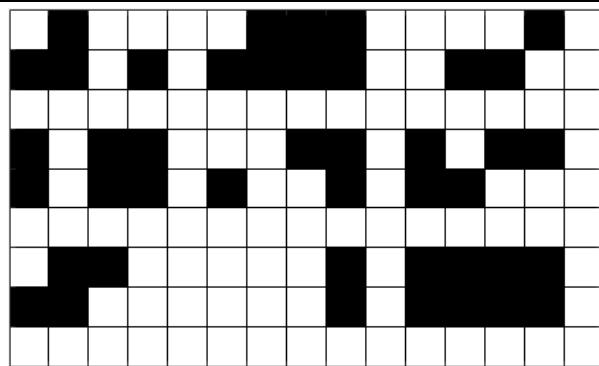


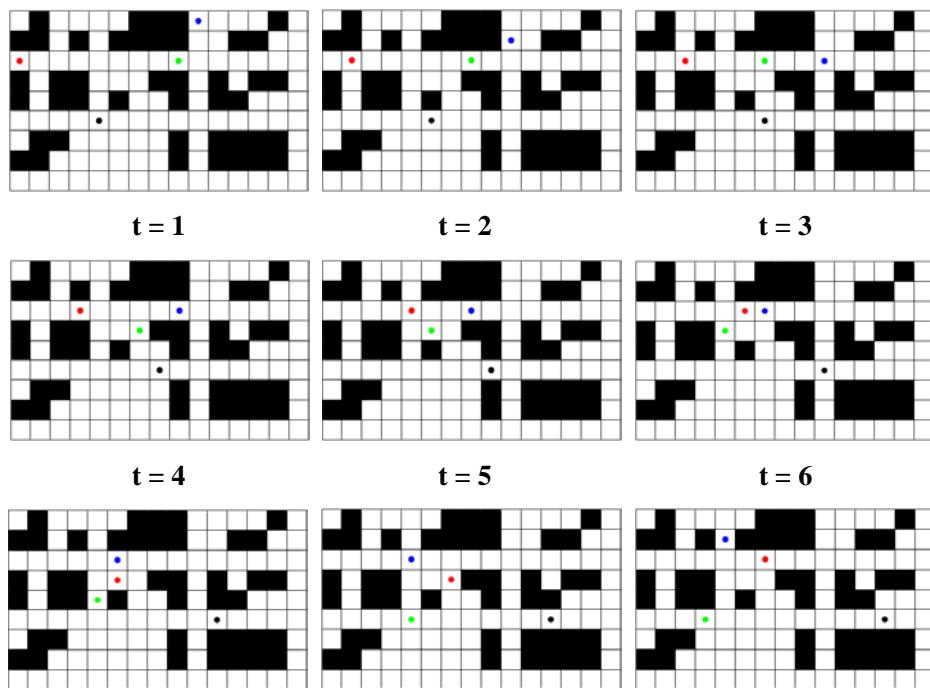
图 3.17 例 4 地图构建

使用线性化的目标函数的模型进行求解，求解结果如下：

ch	status	'OPTIMAL'
E	versioninfo	1x1 struct
runtime	6.1934	
objval	1076	
x	10816x1 double	
slack	13610x1 double	
poolobjbound	1076	
pool	1x4 struct	
mipgap	0	
objbound	1076	
objboundc	1076	
itercount	10301	
baritercount	0	
nodecount	168	

图 3.18 例 4 的仿真结果

机器人的路径如下：



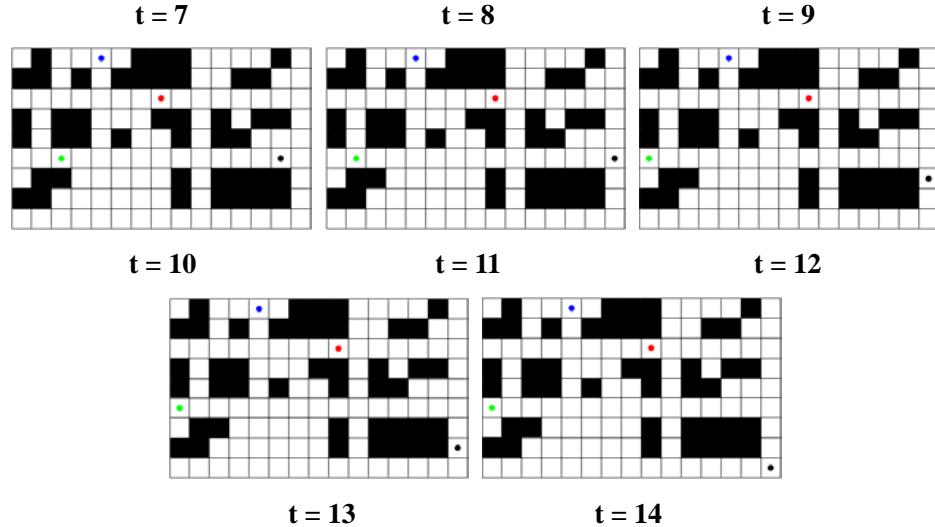


图 3.19 例 4 机器人路径

机器人的路径会较例 3 的结果有所不同。但是在数据一样的前提下，明显可见障碍物的数量对仿真结果有影响：随机减少 40% 的障碍物，求解时间只有 6.2 秒左右。

为了进一步验证障碍物数量对求解时间的影响，使用图 3.13 作为初始地图，机器人数量与起点、终点均与例 3、例 4 相同，使用线性化的模型，障碍物减少量为 0%，10%，20%，…，100%，每次各运行十次程序，障碍物减少的部分均随机。统计求解时间如下：

表 3.2 障碍物数量与求解时间关系统计表

障碍物减少比例 \ 试验次数	1	2	3	4	5	6	7	8	9	10
0%	20.6780	24.5294	21.9942	22.3137	23.0037	24.3048	29.1346	26.0795	24.2548	23.8283
10%	22.7416	20.5279	24.7226	15.5231	17.6022	21.9529	12.1830	12.4392	20.9521	27.7705
20%	20.6629	14.9251	20.7722	11.5295	23.6846	22.5727	9.8948	12.0868	24.6515	20.1043
30%	20.0777	15.7474	7.6469	12.6502	14.4805	19.2470	8.0080	12.1215	13.6105	14.1507
40%	7.7316	7.6486	5.8547	5.9328	3.4252	7.0738	7.4058	7.1347	4.4785	6.1889
50%	1.8745	4.6677	7.2828	3.6584	5.8265	3.2540	4.2821	5.6220	5.6723	2.9173
60%	6.6005	3.1722	3.9492	1.2726	1.0686	0.9995	3.6672	0.2663	4.3721	2.5946
70%	0.7562	0.3814	2.0474	0.7952	3.0001	0.5692	1.0446	7.8680	0.4697	0.9047
80%	0.3901	0.8303	0.1566	0.2344	0.1705	0.1985	0.3541	0.9625	1.8770	0.1994
90%	0.0708	0.0788	0.0878	0.7051	0.0708	0.1067	0.0608	0.0748	0.0449	0.0588
100%	0.0399	0.0369	0.0379	0.0399	0.0349	0.0389	0.0389	0.0389	0.0369	0.0369

对应的均值与方差统计图如下：

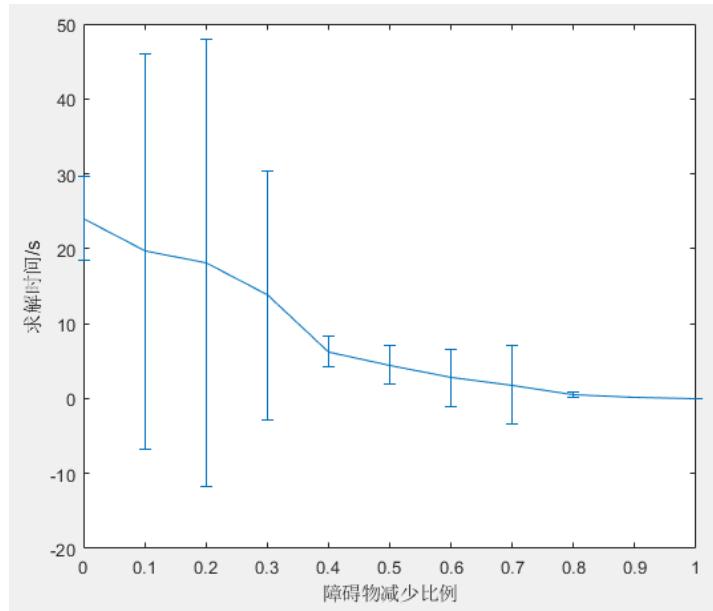


图 3.20 障碍物数量与求解时间关系统计图

从图中可以看到，障碍物越少，求解时间越短。在障碍物随机减少 40% 的时候，求解时间下降较快，方差较小。因此在保证功能的前提下，减少障碍物(即货架)的密集程度，有利于提高求解速度，并可以增加可行解的数量，有一定的实际意义。

3.6 小结

本章主要讨论了单个子区域内的多机器人路径规划算法。由于每个机器人在每一时间点的位置状态可以用栅格的整数坐标表示，因此可以用整数线性规划解决机器人的路径规划问题。

为了实现这一应用场景，整数线性规划的约束包括：1. 机器人有各自的起始与目标位置；2. 机器人路径必须连续；3. 不得与固定障碍物碰撞；4. 机器人之间不得以相遇碰撞或迎面碰撞等形式冲突。同时，整数线性规划的目标函数为所有机器人的总路径最短。

本文利用仿真程序实现了这一功能，验证了以上约束的完备性和正确性。同时讨论了该规划问题模型构造、地图大小、机器人数和障碍物数量等因素对求解速度的影响，得出结论：线性模型、尽量小的地图、尽量少的机器人和障碍物可以加快求解速度。

4 多任务管理

4.1 多机器人多任务分配

上文中提到，本文拟定的多机器人和多任务的位置随机生成。对于本文的应用场景，仿真程序的运行流程为：多任务分配给每个空闲机器人，随后机器人前往任务点搬取货架，获取任务之后前往指定的人工拣货区，再把空的货架送回原位。

本文拟定一种简单的任务分配算法，以任务点为中心，以任务点的编号为其优先级，按顺序给任务分配对应的执行机器人。流程图如下：

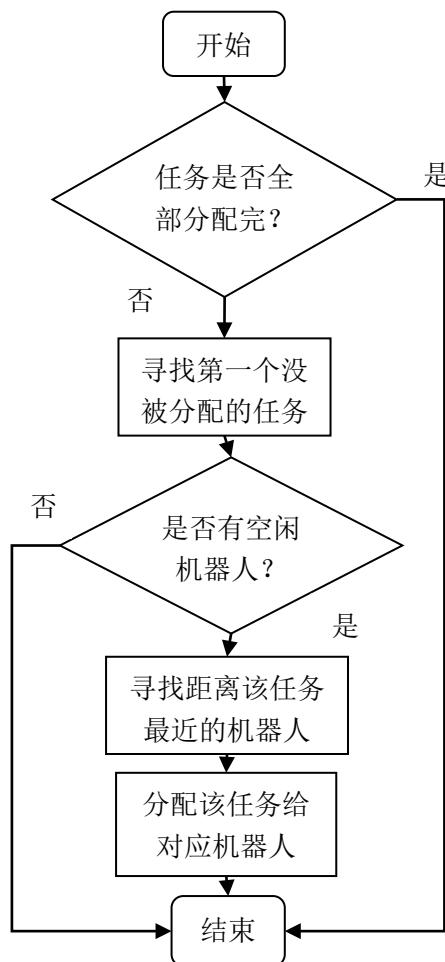


图 4.1 任务分配流程图

在 MATLAB 平台可以使用定时器功能，每隔一定时间进入一次回调函数，执行如上的程序，一旦有机器人空闲，就把未被执行的任务中优先级最高的分配给该机器人。由于机器人前往任务点的路径需要后期规划，因此在分配任务

的时候只能参考每个机器人距离任务点的曼哈顿距离，分配给参考距离最短的机器人。

4.2 动态任务管理

本文的仿真程序实现了动态的任务添加与删除功能。流程图如下：

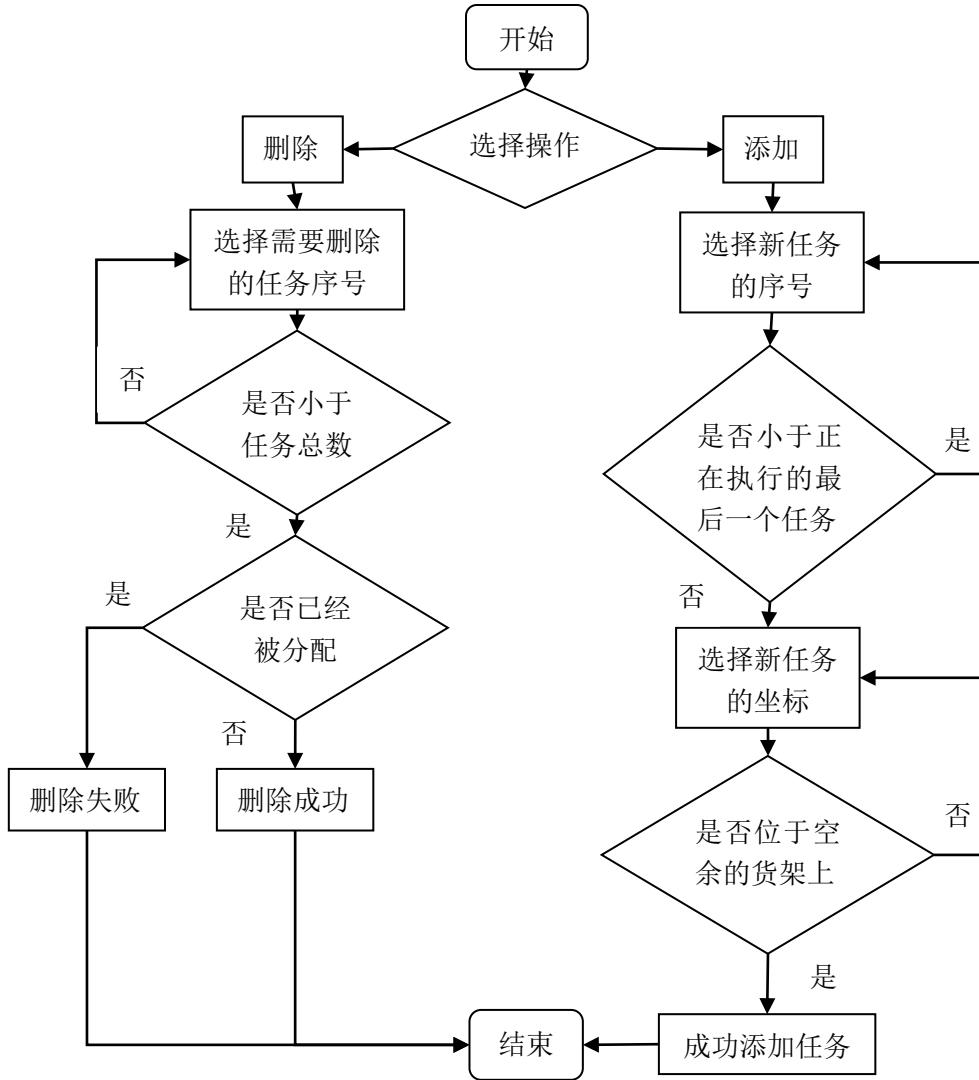


图 4.2 动态管理任务流程图

利用 MATLAB 的键盘 GUI 功能，用户可以自己添加或删除任务。原则上，添加的任务编号不能小于正在执行的最大任务编号。因此每个机器人会存储正在运行的任务的编号，添加新的任务会引起编号错乱，同时，逻辑上机器人也不能放下正在执行的任务，去执行新添加的任务。另外，要删除的任务必须不能已经被分配(或正在执行，或已经完成)。

4.3 仿真

本节采用几个简单的多任务到多机器人的任务分配用例，以及动态管理任务的用例，检验任务管理模块的可行性。

例 1：5 个任务，3 个机器人。初始任务分配如下：

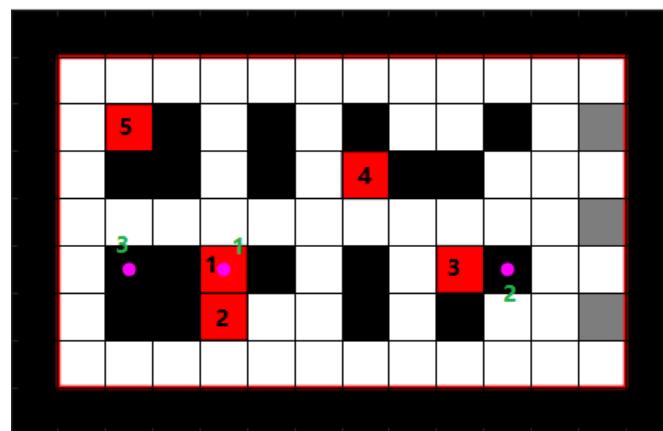


图 4.3 例 1 任务分配

图中黑色编号为任务编号，绿色编号为机器人编号。工作区内任务的信息如下：

字段	str	tstate	position	exit	zone	exit_zone	executer	edge
1	"assigned"	[6,5]	[5,13]	[1,1]	[1,1]		1	0
2	"assigned"	[7,5]	[7,13]	[1,1]	[1,1]		3	0
3	"assigned"	[6,10]	[5,13]	[1,1]	[1,1]		2	0
4	"new"	[4,8]	[3,13]	[1,1]	[1,1]		0	0
5	"new"	[3,3]	[3,13]	[1,1]	[1,1]		0	0

图 4.4 例 1 任务信息

可见任务 1 分配给了机器人 1，任务 2 分配给了机器人 3，任务 3 分配给了机器人 2。任务是由优先级依次分配机器人，每个任务分配给离它最近的机器人。结合图中信息可以看到，本例在任务多于机器人数量时，分配正确。

例 2：3 个任务，5 个机器人。初始任务分配如下：

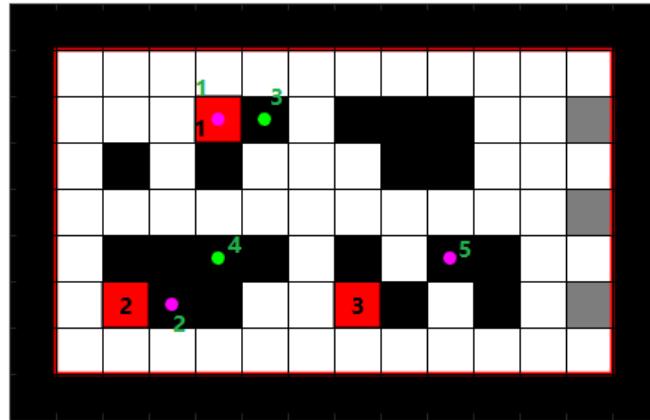


图 4.5 例 2 任务分配

工作区内任务的信息如下：

字段	str	tstate	position	exit	zone	exit_zone	executer	edge
1	"assigned"	[3,5]	[3,13]	[1,1]	[1,1]		1	0
2	"assigned"	[7,3]	[7,13]	[1,1]	[1,1]		2	0
3	"assigned"	[7,8]	[7,13]	[1,1]	[1,1]		5	0

图 4.6 例 2 任务信息

可见，任务 1 分配给了机器人 1，任务 2 分配给了机器人 2，任务 3 分配给了机器人 5。同样，本例也是任务按优先级分配，当前任务分配给距离最近的机器人。本例在任务数少于机器人数时，分配正确。因此，该分配方案具有可行性。

例 3：动态任务管理。初始数量设置为任务 5 个，机器人 3 个。初始地图布局如下：

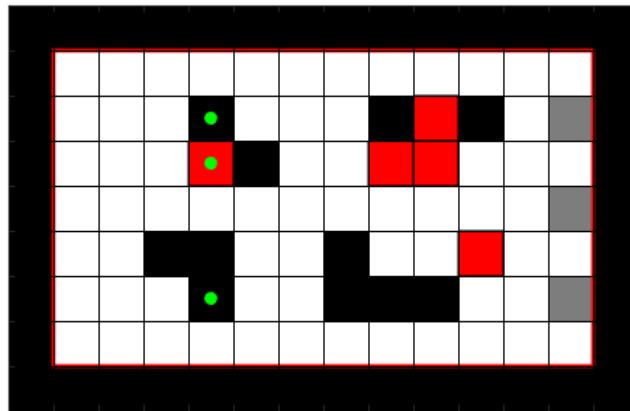


图 4.7 例 3 初始地图布局

添加任务：在 GUI 界面按下字母 a 表示 add(添加任务)，命令行窗口会显示所有可以供添加任务的货架位置，并提示选择优先级，选定位置和优先级后，任务添加完毕。

```
命令行窗口
=====
Add new tasks=====
Available positions as follows:
 6   4
 3   5
 6   5
 7   5
 4   6
 6   8
 7   8
 3   9
 7   9
 7   10
 3   11

Select a position for a new task:
[3,5]
Available priority: 4 - 6
4
Successfully added a new task!
fx >> |
```

图 4.8 例 3 添加任务

添加任务之后的地图如下：

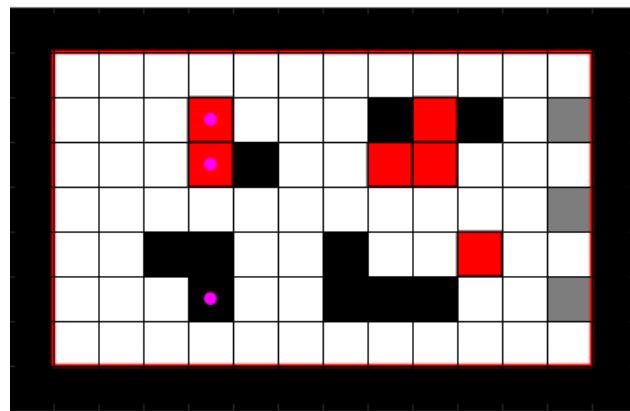


图 4.9 例 3 添加任务后的地图布局

对比图 4.7 可以看到地图上 [3,5] 的位置多了新的任务点。同时，在工作区可以查看任务信息，编号为 4 的就是刚刚添加的任务 [3,5]。

字段	str	tstate	position	exit	zone	exit_zone	executer	edge
1	"assigned"	[4,9]	[3,13]	[1,1]	[1,1]		2	0
2	"assigned"	[4,5]	[3,13]	[1,1]	[1,1]		3	0
3	"assigned"	[6,11]	[5,13]	[1,1]	[1,1]		1	0
4	"new"	[3,5]	[3,13]	[1,1]	[1,1]		0	0
5	"new"	[4,10]	[3,13]	[1,1]	[1,1]		0	0
6	"new"	[3,10]	[3,13]	[1,1]	[1,1]		0	0

图 4.10 例 3 添加任务后的任务信息

删除任务：在 GUI 界面按下 d 表示 delete(删除任务)，命令行窗口会提示可供删除的任务编号，即未被处理的任务：

```
命令行窗口
Unassigned tasks as follows:
4      5      6

Select a new task to delete:
5
Successfully deleted a task!
fx >> |
```

图 4.11 例 3 删除任务

删除任务之后的地图如下：

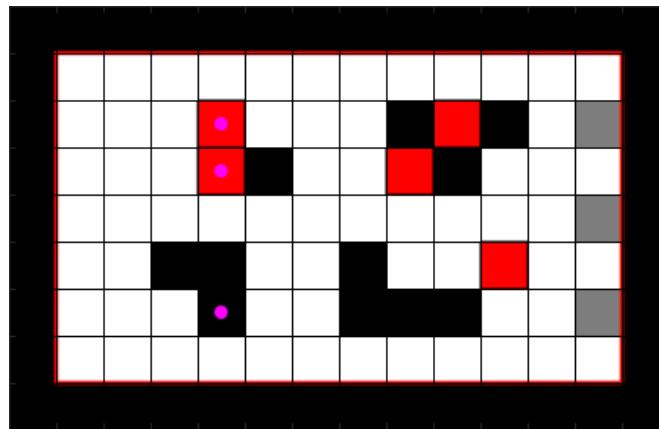


图 4.12 例 3 删除任务后的地图布局

对比图 4.7 可以看到，任务点[4,10]已经不存在。在工作区可以查看任务信息，之前图 4.10 中编号为 5 的任务点不存在。

task							
1x5 struct 包含 7 个字段							
字段	tstate	position	exit	zone	exit_zone	executer	edge
1	"assigned"	[4,9]	[3,13]	[1,1]	[1,1]	2	0
2	"assigned"	[4,5]	[3,13]	[1,1]	[1,1]	3	0
3	"assigned"	[6,11]	[5,13]	[1,1]	[1,1]	1	0
4	"new"	[3,5]	[3,13]	[1,1]	[1,1]	0	0
5	"new"	[3,10]	[3,13]	[1,1]	[1,1]	0	0

图 4.13 例 3 删除任务后的任务信息

4.4 小结

本节详细介绍了一种简单的任务分配方案，和仿真程序动态管理任务的功能。通过几个仿真用例可以证明分配方案和动态管理任务功能的可行性。

5 多区域之间的协同工作

对于多机器人在全局的运动问题，本文采用的策略是将整个区域划分成多个子区域。对于单个子区域内的多机器人路径规划，在上文中已经做了详细的介绍。本章的主要内容是介绍多个子区域之间，机器人如何调度。

对于不同子区域之间机器人的调度，本文提出了两种解决方案，接下来将着重介绍这两种不同的方案，并将其做对比。

5.1 方案一：子区域密集型

5.1.1 模型构建

对于子区域密集型的方案，子区域的分割方式可以参考第二章图 2.4。即子区域之间紧密相邻。

由图 2.4 可以看到，每个区域内都有货架，因此也都存在任务点，而人工拣货区（灰色部分）仅存在于整个地图的最右侧。如果子区域的数量大于 1，则在程序运行过程中，必定需要机器人跨区域协作。

如果机器人的目标位置不在当前区域，本方案采用的一般策略是，先确定机器人的目标区域，确定到达目标区域的方向。到达目标区域的方向只有一个或两个（例如：上、下、左、右、左上、右下等），如果方向只有一个，则只能进入这个方向的相邻子区域；如果方向有两个，则判断与当前区域这两个方向

的相邻的子区域，哪个区域内部机器人数量比较少，选择该区域为下一区域。

这一方法适用于机器人的各种状态：前往任务点、前往拣货区、送回空货架等。

对于某些位置的任务点则存在例外，如下图 5.1：

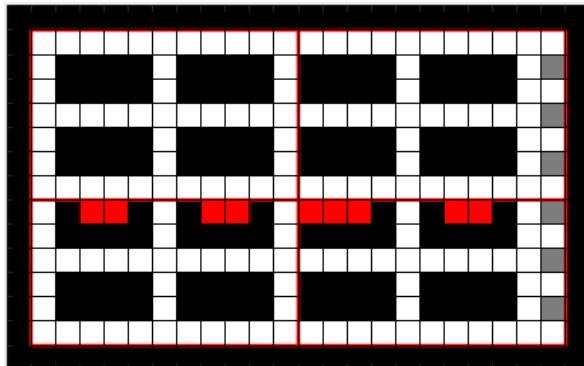


图 5.1 一些特殊的任务点

对于图中所示的红色的任务点，被取走之后的下一位置只能是往上移动一个栅格，即下一区域必定是上方区域。同样，这些位置的空货架被送回时，可能会先到达本区域，但是无法直接在本区域内送到原来的位置，而是必须送到上方区域的对应位置，再往下移动一个栅格，到达指定位置。

5.1.2 当前区域到下一区域的出口选取

在确定了下一个区域之后，就应当确定如何从当前区域前往下一个区域，即选择一个位置作为当前区域到下一区域的出口。

对于这一任务，需要考虑以下三种类型的冲突：

1. 机器人在进入下一区域时不得与下一区域内部原有的机器人碰撞。如下

图 5.2：

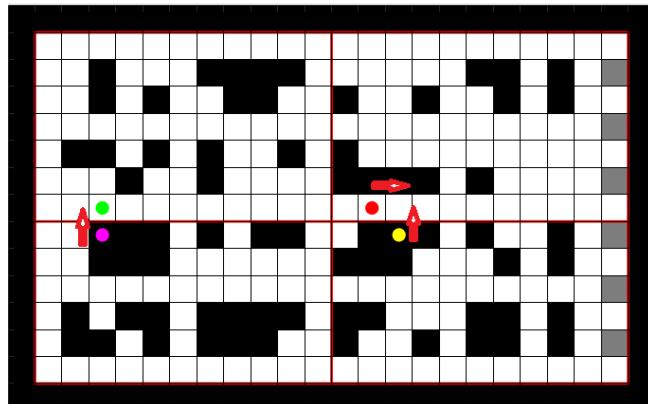


图 5.2 冲突例 1

当上方子区域有机器人，如果图中的洋红色机器人在前往上方子区域时会与静止的绿色机器人碰撞，则不得进入；如果图中黄色机器人在前往上方子区域时会与运动中的红色机器人碰撞，也不得进入。具体的措施会在下一节详细介绍。

2. 两个不同区域的机器人进入同一区域时，不得产生碰撞。如下图 5.3：

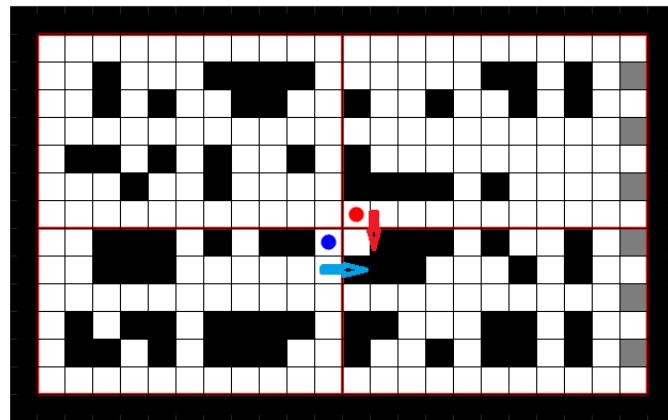


图 5.3 冲突例 2

如图中，如果蓝色和红色机器人都要进入右下方的子区域，此时会产生碰撞，这种情况应当避免。

3. 两个相邻子区域互有机器人来往时，不得产生迎面碰撞。如下图 5.4：

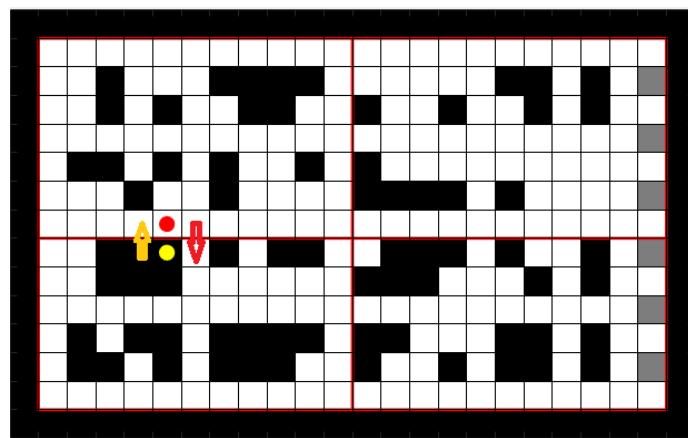


图 5.4 冲突例 3

迎面碰撞的定义在单个子区域路径规划部分已经提到过。如上图，如果荒色机器人要前往上方子区域，红色机器人要前往下方子区域，则会发生迎面碰撞。这种情况应当避免。

对于一般情况下的任务点，本文采用的方法为类单行线法则。如下图：

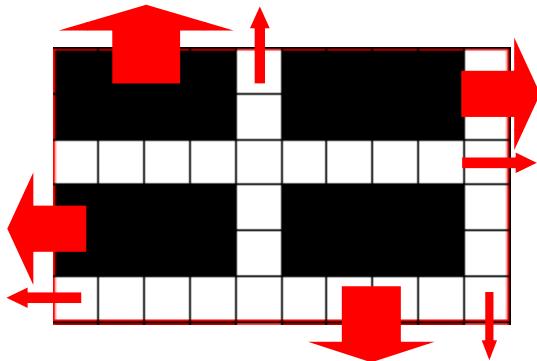


图 5.5 方案一子区域的出口设置

对于每一个子区域，规定前往四个方向邻近子区域的出口。由于带着货架运行的机器人不允许从另外货架底下经过，而不带货架的机器人可以，因此上图中四个粗箭头代表不带货架的机器人前往四个方向的出口，细箭头代表带货架的机器人前往四个方向的出口。

对于上述的三种冲突情况，这种方案可以避免 2、3 两种。不会有两个机器人从不同区域前往同一子区域时发生碰撞；两个相邻子区域在机器人互相来往时也不会发生迎面碰撞。且此方案可以发挥不带货架的机器人可以在货架底下运行的特点。但是此方案的缺点是，机器人前往下一子区域的路径未必最短。而对于上述第一种冲突情况，下一节会详细介绍。

上一节中提到，有一些位置的任务点属于特殊任务点。由图 5.1 可见，对于这种区域划分策略，这些任务点必定在某一子区域的最上面一排。对于这些特殊任务点，下一子区域必然是上面邻近的子区域，而出口就是该任务点的位置本身。

5.1.3 进入下一子区域的过程

上一节中提到，机器人进入下一子区域时不得与该子区域内部原有的机器人产生碰撞冲突，且上一节中采用的出口设置方案并没有成功解决这一问题。

对于一个机器人，其进入下一个子区域之前，需要先到达出口位置。在出口处进入下一子区域时，需要判断该子区域的对应栅格是否会有机器人。如果没有，则直接可以进入；如果有，则需要判断该机器人会不会移开，如果是空闲的机器人，则其不会主动移开，需要将其移动一下，让需要进去的机器人进

去。另外，如果在下一子区域的该位置出现一个机器人，该机器人的任务属于特殊任务点，需要往下移动，则同样会出现迎面碰撞的情况，见下图 5.6，此时可以设定让搬运特殊任务点的机器人让路：

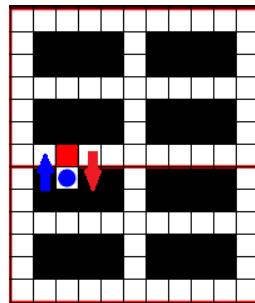


图 5.6 特殊情况

如图，蓝色圆形机器人要前往上方的子区域，红色方形机器人要前往蓝色机器人所在位置放置货架，此时属于特殊情况。

程序流程图如下：

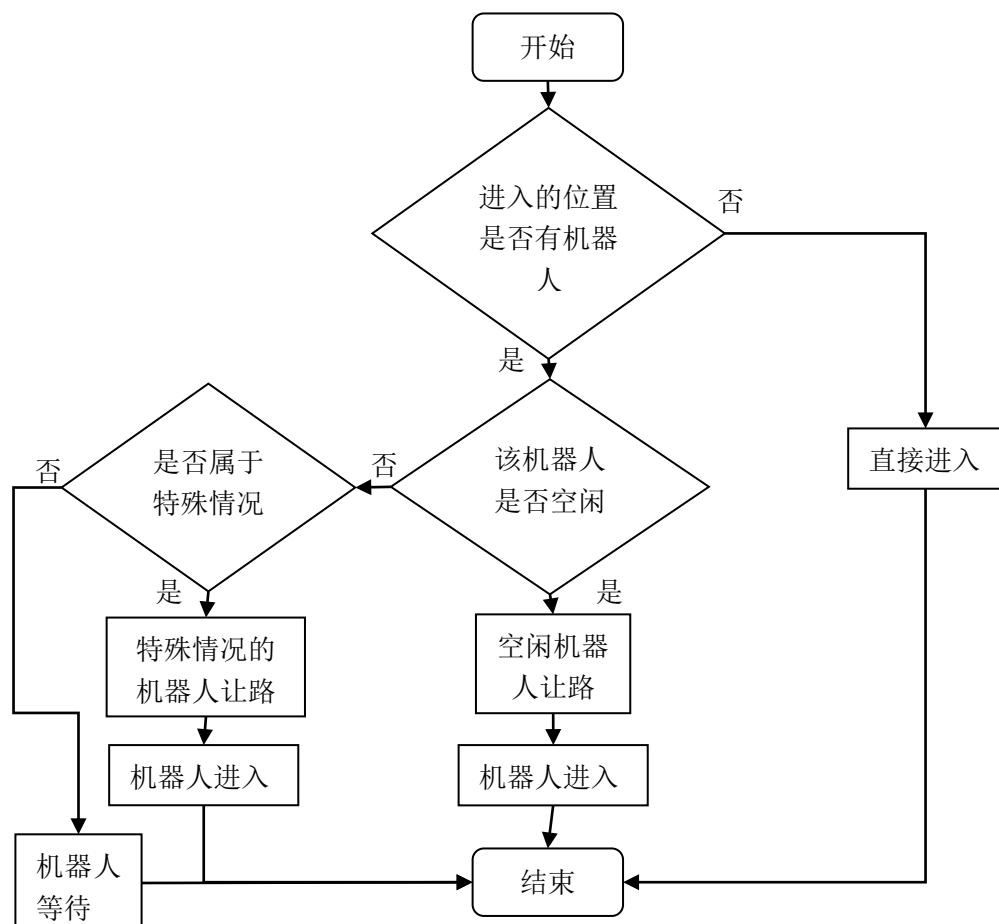


图 5.7 机器人进入子区域流程图

5.1.4 仿真

本节用几个仿真案例，验证多区域之间机器人交互的功能是否完善。

例 1：检验特殊任务点的工作流程。取货路径如下：

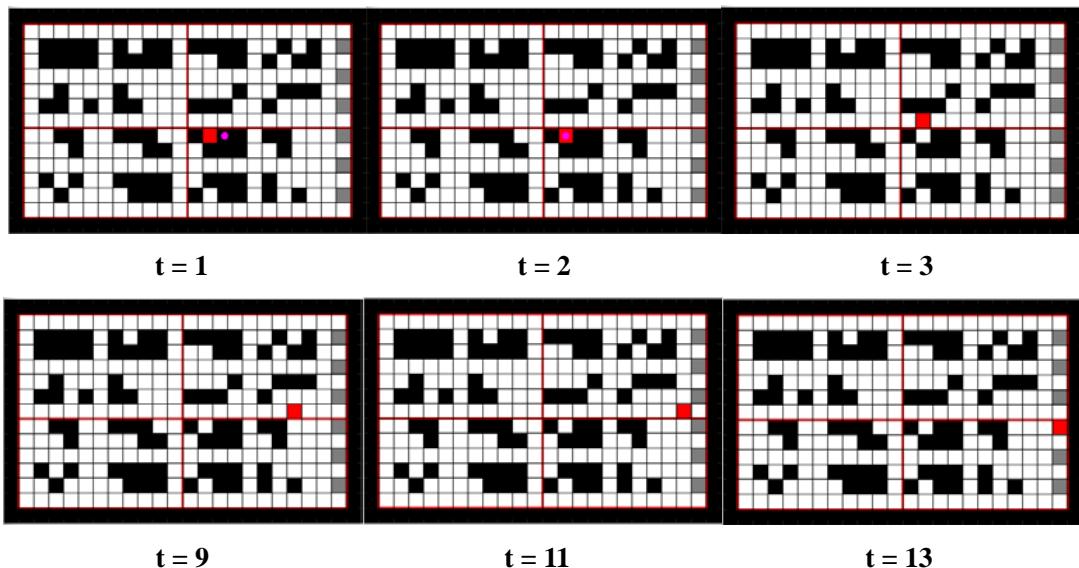


图 5.8 例 1 机器人取货架路径

由于版面有限，此处略去部分移动路径。可见，本节对于机器人在子区域之间往来的设想基本得以实现。虽然任务点和目标拣货区处于同一子区域，但是由于任务点处于特殊位置，机器人会将其搬到上方子区域，再规划路径，返回到右下方的子区域。

机器人运回货架的流程如下：

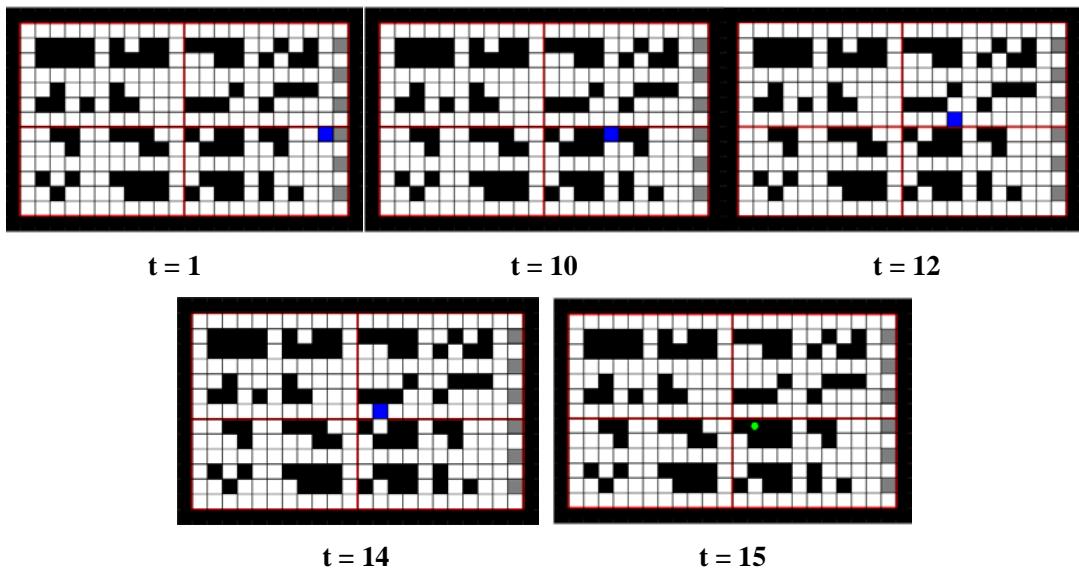


图 5.9 例 1 机器人送回货架路径

在本例中，机器人不能在右下角子区域内直接把任务点送回原处，而是需要送到上方子区域的对应位置，再往下移动一个栅格送到任务原始位置。本例可以验证机器人能够在不同子区域内往返，并且特殊位置的任务点也可以得到特殊处理。

例 2：验证机器人能否在不与其他机器人碰撞的情况下进入另一个子区域。

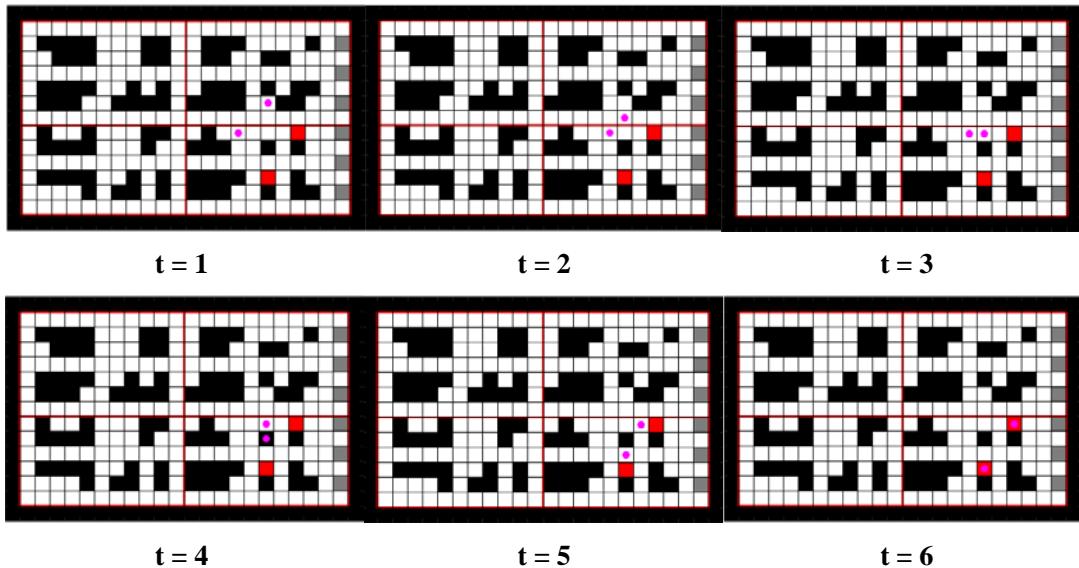
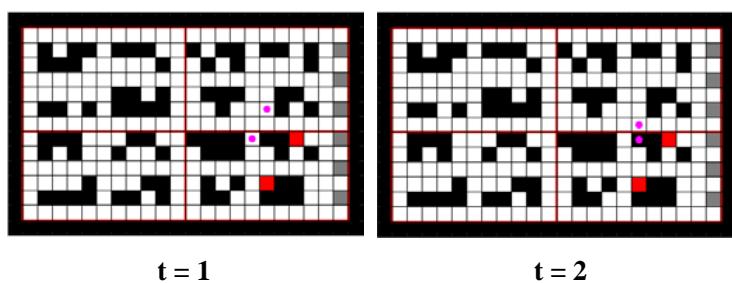


图 5.10 例 2 机器人取货架路径

图中可见，当原本处于上方子区域的机器人到达边缘时，下一栅格没有机器人，因此该机器人可以直接进入右下方的子区域，而下方子区域内原有的机器人等待了一个时间单位。待该机器人进入子区域后，右下方的子区域立即重新规划路径。

例 3：改变一下机器人位置，再次验证机器人能否在不与其他机器人碰撞的情况下进入另一个子区域。



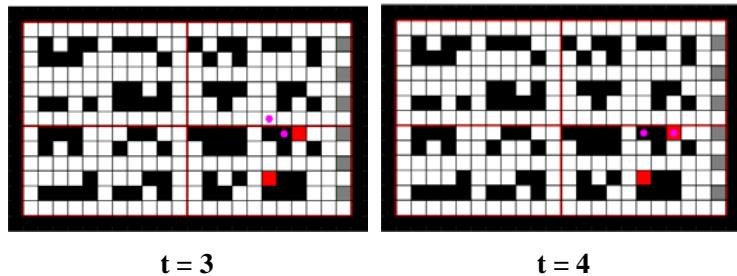


图 5.11 例 3 机器人取货架路径

在本例中，欲进入下方子区域的机器人到达边缘时，该进入位置有机器人，此时该机器人会在原来的子区域等待一个时间单位，直到下方挡路的机器人离开为止。

例 4：假设待进入的子区域内部有个静止的机器人挡路，验证进入子区域是否无碰撞。

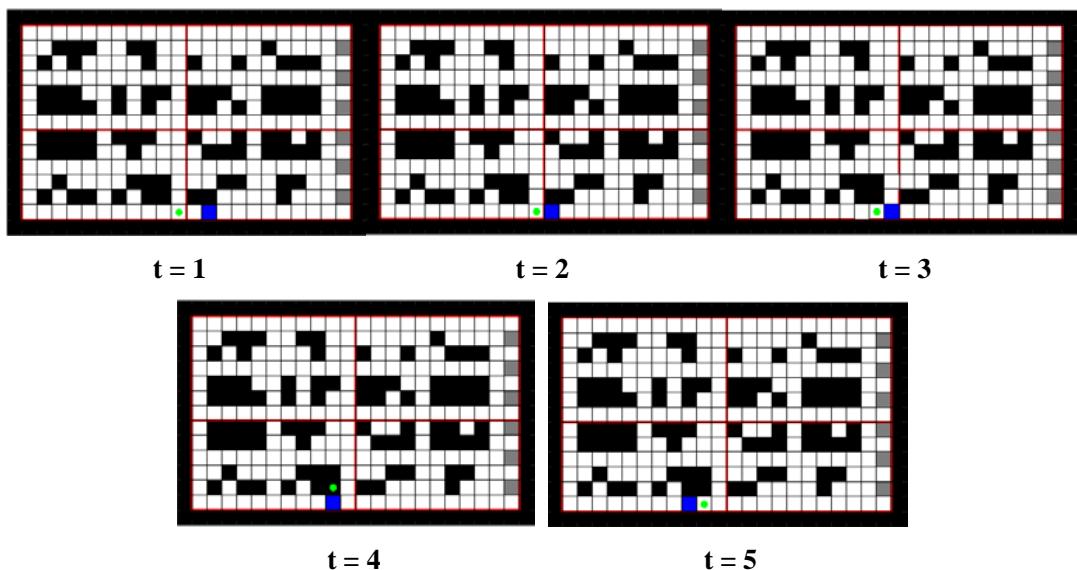


图 5.12 例 4 机器人取货架路径

由上图可见，由于静止的(绿色)机器人一般不会移动，因此不能让蓝色机器人一直等待，而是蓝色机器人进入的时候要求绿色机器人让路。

根据以上几个例程可以看出，子区域紧密型的方案用于解决机器人在不同子区域之间往返具有可行性。

5.2 方案二：子区域与单行线结合型

5.2.1 模型构建

对于本方案，子区域的划分可以参考第二章图 2.5。

由图中可以看出，子区域之间并不紧密相连，而是不同子区域之间有独立于区域之外的道路。

人工拣货区集群单独成为特殊的一个子区域，且人工拣货区作为一个子区域，并不使用 ILP 进行规划，而是同样根据单行线，简单地对机器人进行下一位置的规划，保证其能够到达拣货口，并离开人工拣货区所在子区域。

道路的行走规则为单行线，单行的方向如下：

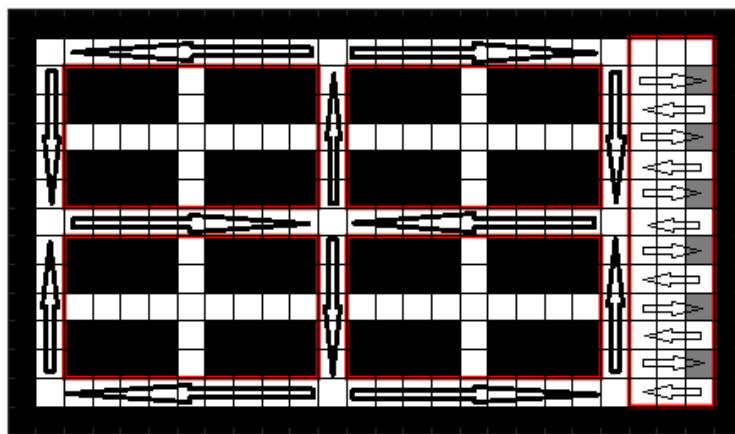


图 5.13 单行线的规定

图中规定了 4×4 数量的货架时无冲突单行线的方向。该单行线的方向设置可以保证子区域外公共道路上的任意一点的能达性，可以满足机器人途中运送任务和在拣货区的行为。同理可以规定任意地图大小的无冲突单行线方向。

5.2.2 离开和进入子区域的过程

类似于上一种方案，这种方案也有一些特殊的任务点：

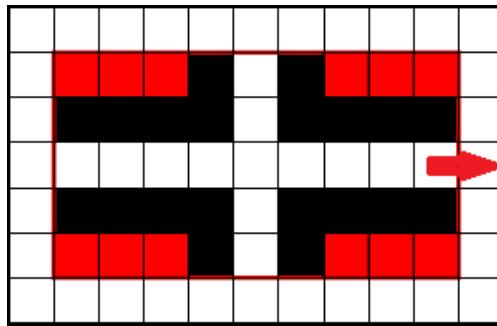


图 5.14 方案二的特殊任务点

图中红色区域所示的任务点，可能会被黑色的空货架所阻挡，从而无法通过图中红色箭头的位置离开该子区域。对于此类任务点，规定其下一步直接前往周围的公共道路即可。在机器人完成出货，将货架送回时，也需要单独考虑。如果由图中子区域的白色道路进入，则有可能无法到达原来的位置。因此，规定其从道路上与原来位置相邻的地方直接到达该位置，放回空货架。

不同于上一方案，此方案四个角落均有特殊任务点。对于此方案，规定出口和入口如下：

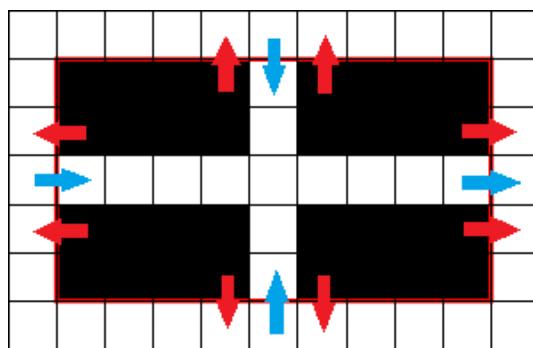


图 5.15 方案二子区域的出入口设置

比较特殊的是，在这种方案下，机器人带着货物从一个子区域出来，必定是前往人工拣货区，不会中间穿过另一个子区域。因此，对于带着货物的机器人，出口只需设定为子区域右边缘的中间位置(蓝色箭头)；而对于不带着货物的机器人，其前往取货的子区域可能位于各种方位，同时为了不与特殊任务点冲突，不带着货物的机器人沿着图中红色箭头所示的位置走出当前子区域。

当机器人要进入该子区域时，可能是前来取货或者把空货架送回来的情况。规定只能由上方、下方和左方的蓝色箭头进入，不管机器人是否携带空货架。对于带着空货架的机器人，由于不能与其他货架冲突，因此只能沿着道路走；

而不携带货架的机器人理应可以在货架底下运行，但是为了避免与特殊任务点以及刚才规定的出口冲突，也规定只能从子区域的白色道路进入。

5.2.3 碰撞及其防止措施

除了单个子区域内部的碰撞外，本方案需要考虑的碰撞有以下几种：

1. 两个机器人进出子区域产生的迎面碰撞。

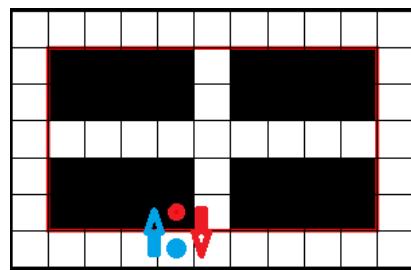


图 5.16 方案二碰撞情况例 1

2. 同一条单行线上的相遇(追尾)碰撞。

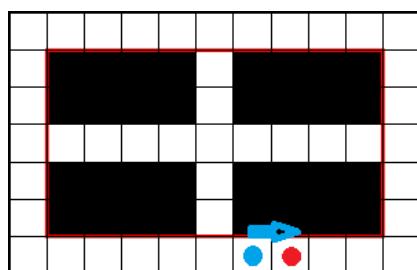


图 5.17 方案二的碰撞情况例 2

3. 交叉路口的相遇碰撞。

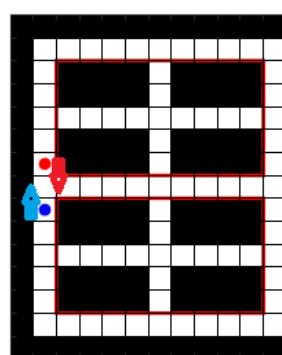


图 5.18 方案二的碰撞情况例 3

4. 机器人前往道路产生的相遇碰撞。

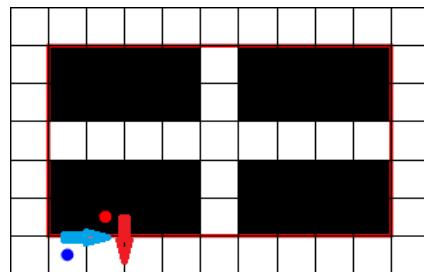


图 5.19 方案二的碰撞情况例 4

5. 机器人进入子区域产生的相遇碰撞。

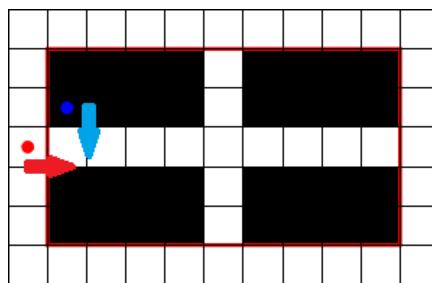


图 5.20 方案二的碰撞情况 5

对于情况 1，可以通过上一节所述的出口方案解决。其他所有情况都有至少一个机器人在单行线上，都可以通过在单行线算法中添加以下算法解决，对于当前在单行线上需要移动的机器人：

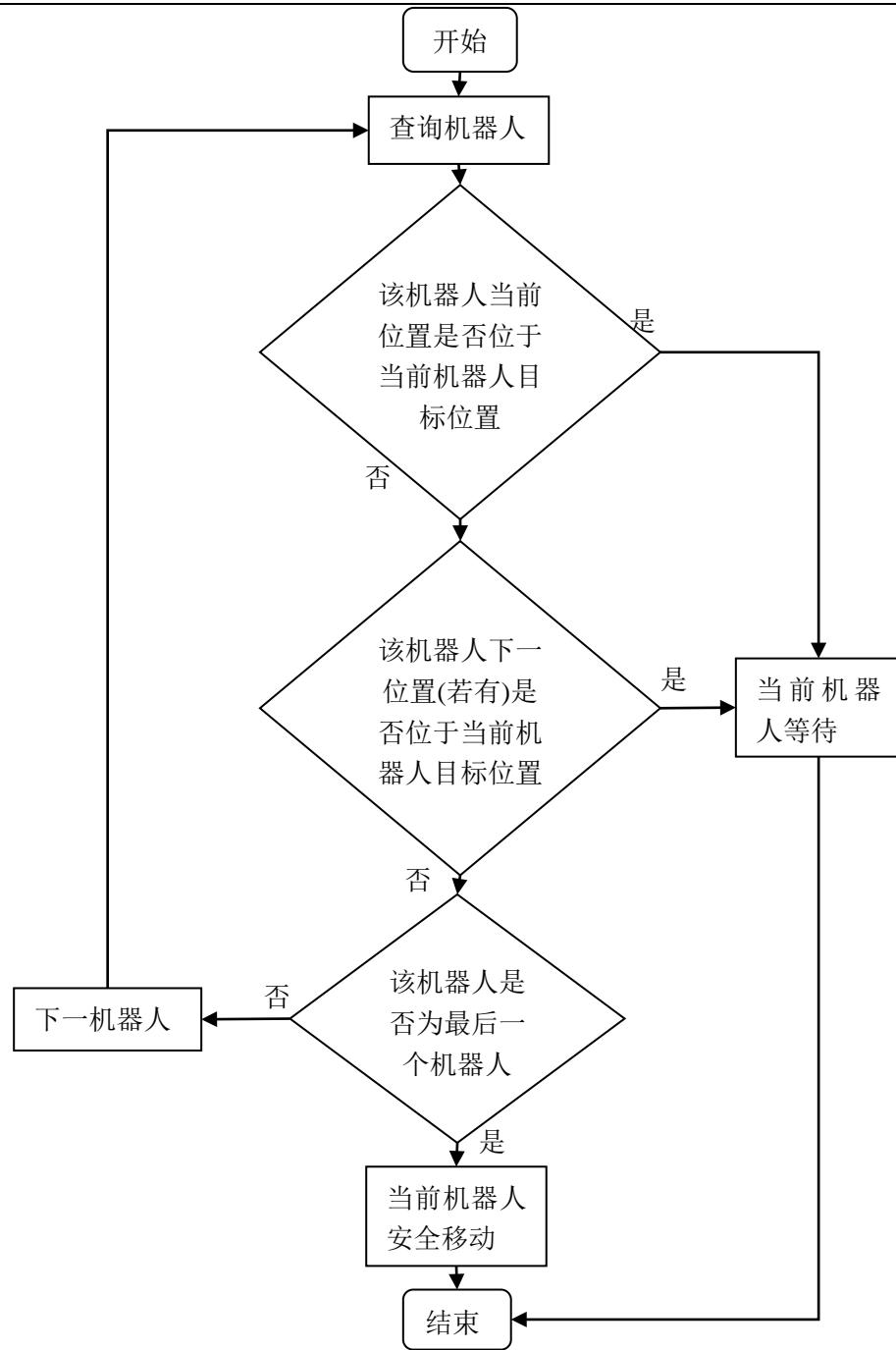


图 5.21 碰撞检测流程图

5.2.4 仿真

本节采用一些仿真案例，检验本方案的机器人多区域交互功能是否完善。

例 1：检验单个区域的单行线功能，同时检验特殊任务点是否可以特殊处理。

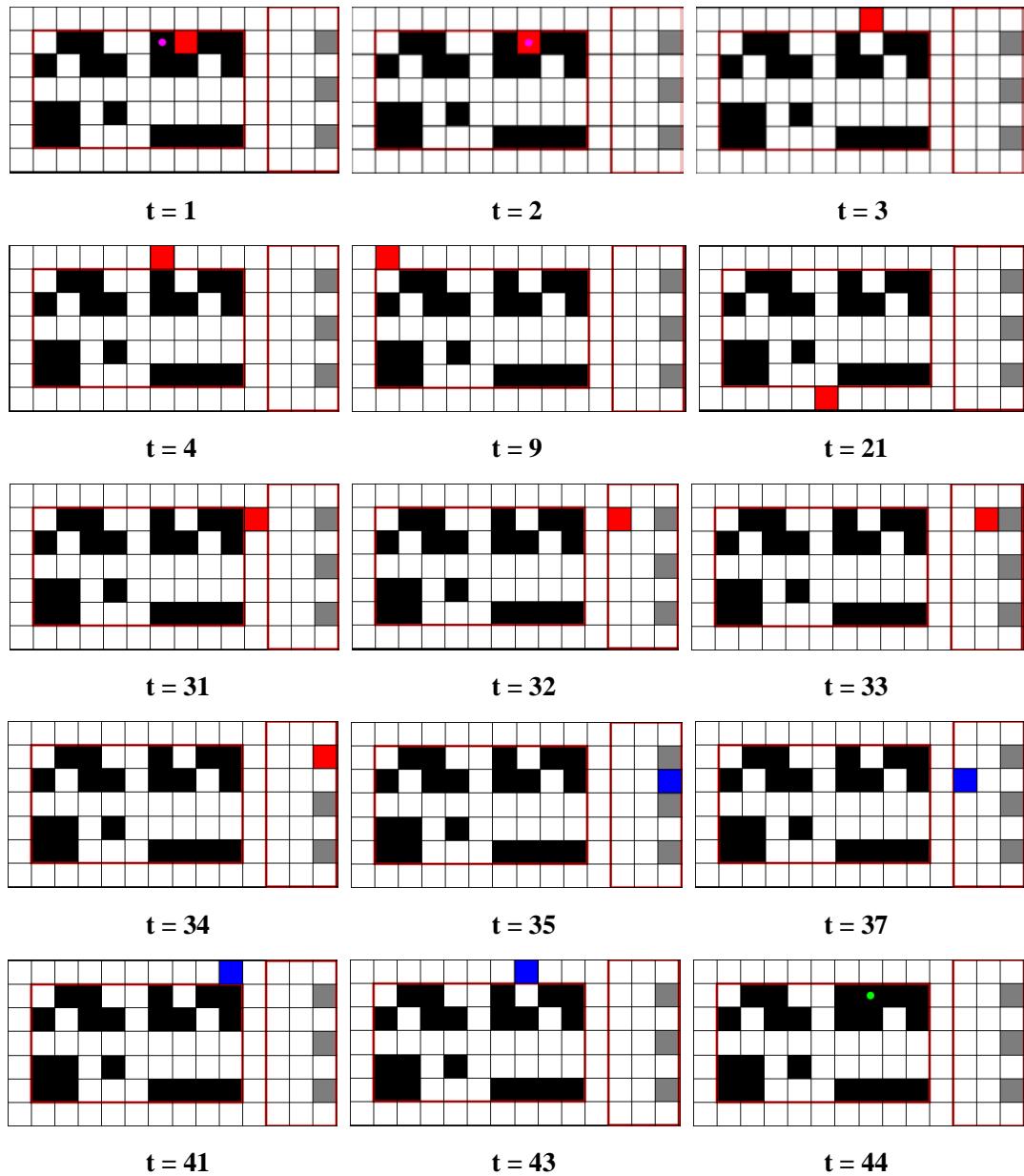


图 5.22 例 1 机器人路径

由于版面有限，省略了部分路径。从这一系列的路径图中，可以明确看到机器人对特殊任务点，直接送到区域之外的单行线上，送回时也是从单行线上直接送回。由于外围的单行线是逆时针，必须严格沿着单行线行走，机器人多走了一些路。

例 2：检验机器人在交叉路口的防相遇碰撞措施。

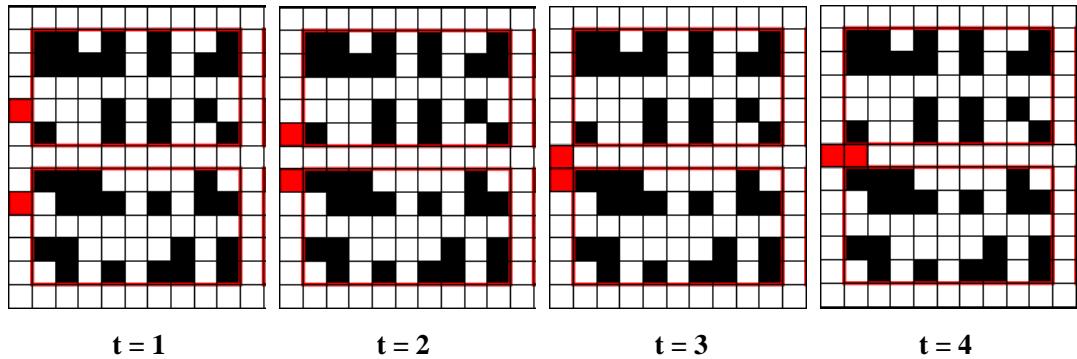


图 5.23 例 2 机器人路径

由图中可以看出，在交叉路口，如果没有防止相遇碰撞的措施，机器人会在[7,1]处相撞。

例 3：检验机器人在出入子区域时的防迎面碰撞措施。

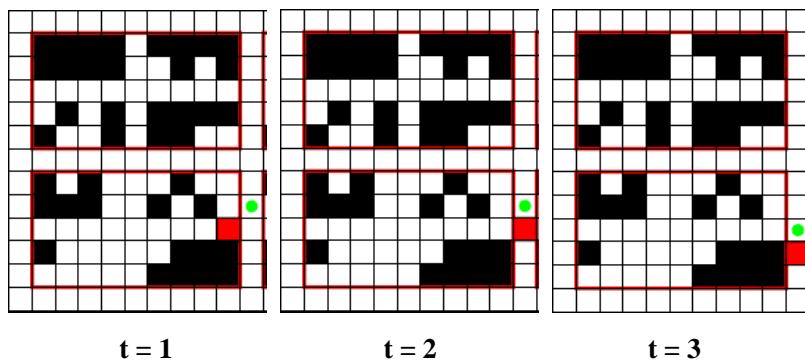


图 5.24 例 3 机器人路径

图中可见，在道路上行走的绿色机器人，由于优先级低于红色机器人，会等待一个时间单位，等红色机器人先走过。

根据以上的几个仿真用例可以看出，单行线与子区域的结合方案具有可行性。

5.3 两种方案的对比

本节采用几组初始条件(包括货架布局、任务数量和位置、机器人数量和初始位置)完全相同的仿真用例，通过在程序中添加一些观察变量(总路径长度、总规划时间、总规划次数)，对比上述提出的两种方案，比较各自的优点和不足之处。由于完成任务的总流程较长，本节不再展示机器人的具体路径。

例 1：共 2×2 排货架，单个机器人，单个任务。使用方案一进行求解，初始

地图布局如下：

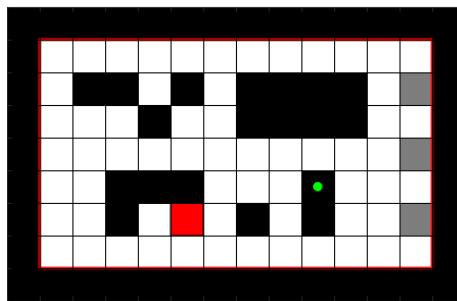


图 5.25 例 1 方案一的初始地图布局

求解结果如下：

名称	值
fig	1x1 Figure
last_data	1
map	9x14 double
robot	1x1 struct
run_mode	1
t_dyna_plot	1x1 timer
t_path_plan	1x1 timer
t_task_alloc	1x1 timer
task	1x1 struct
times_plan	6
total_path_len	24
total_time_plan	0.6003
zone	1x1 struct

图 5.26 例 1 方案一的完整求解结果

可以看到，调用求解器求解总次数为 6 次，总的求解时间为 0.6003 秒，机器人走过的总路径长度为 24。

相同的货架布局、任务数量和位置、机器人数量和位置，使用方案二进行求解，初始地图布局如下：

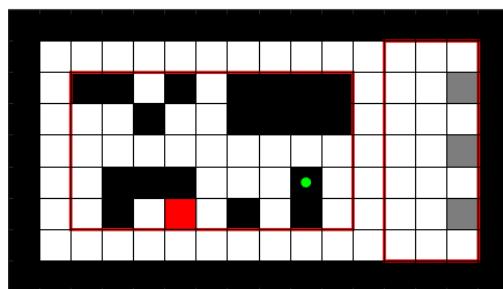


图 5.27 例 1 方案二的初始地图布局

结果如下：

名称	值
direction	9x16 cell
fig	1x1 Figure
last_data	1
map	9x16 double
robot	1x1 struct
run_mode	1
t_dyna_plot	1x1 timer
t_path_plan	1x1 timer
t_task_alloc	1x1 timer
task	1x1 struct
times_plan	5
total_path_len	67
total_time_plan	0.0518
zone	1x2 struct

图 5.28 例 1 方案二的完整求解结果

调用求解器求解总次数为 5 次，总的求解时间为 0.0518 秒，机器人走过的总路径长度为 67。

分析：

1. 求解时间方案二比方案一小了一个数量级，因为方案二的子区域划分与方案一不一样，子区域小很多，导致规划的总时间差距很大。
2. 方案二的总路径比方案一长很多，因为方案二子区域周围有严格的方向规定，单机器人也必须遵照单行线的方向，因此会绕着子区域多走一部分路。

例 2：共 4×4 排货架，3 个机器人，10 个任务。使用方案一求解，地图布局如下：

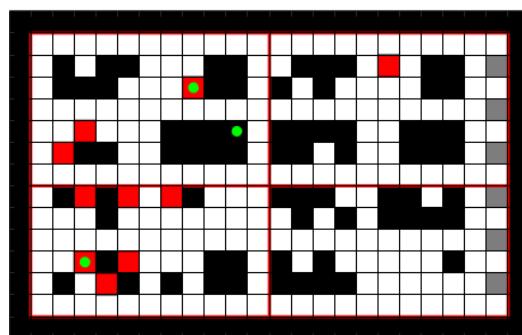


图 5.29 例 2 方案一的初始地图布局

求解结果如下：

工作区	
名称	值
fig	1x1 Figure
last_data	1
map	15x24 double
robot	3x1 struct
run_mode	1
t_dyna_plot	1x1 timer
t_path_plan	1x1 timer
t_task_alloc	1x1 timer
task	1x10 struct
times_plan	114
total_path_len	514
total_time_plan	110.2974
zone	2x2 struct

图 5.30 例 2 方案一的完整求解结果

调用求解器求解总次数为 114 次，总的求解时间为 110.2974 秒，机器人走过的总路径长度为 514。

相同的货架布局、任务数量和位置、机器人数量和位置，使用方案二进行求解，初始地图布局如下：

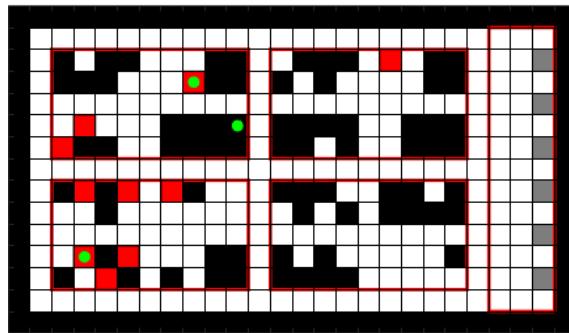


图 5.31 例 2 方案二的初始地图布局

求解结果如下：

工作区	
名称	值
direction	15x26 cell
fig	1x1 Figure
last_data	1
map	15x26 double
robot	3x1 struct
run_mode	1
t_dyna_plot	1x1 timer
t_path_plan	1x1 timer
t_task_alloc	1x1 timer
task	1x10 struct
times_plan	58
total_path_len	801
total_time_plan	0.8433
zone	2x3 struct

图 5.32 例 2 方案二的完整求解结果

调用求解器求解总次数为 58 次，总的求解时间为 0.8433 秒，机器人走过的总路径长度为 801。

由此可见，地图较大，任务和机器人数量较多的时候，差距比较明显：

1. 方案一的求解时间为 110.2974 秒，而方案二只有 0.8433 秒。原因是方案一需要不停地规划路线，而且机器人每时每刻都在某一子区域内，因此规划时间会特别长，而方案二机器人在前往人工拣货区、送回货架的时候并不会经过其他不相关的子区域，所以规划次数也少、每次规划的时间也很少。

2. 方案一的总路径长度为 514，方案二为 801。原因是方案一能够保证机器人在单个子区域内的路径是最优的，而方案二严格的单行线方案会导致机器人多走一些路。

5.4 小结

本章详细介绍了两种方案用于解决机器人在不同子区域之间交互，且可以实现碰撞避免的情况。

一种方案子区域相邻，机器人在完成任务时会穿过多个子区域，每个子区域有机器人进入或离开时都需要重新规划；另一种方案是子区域之间隔了公共的单行线，机器人不会进入无关的子区域，而是在离开子区域后直接进入公共的单行线。

两种方案各有优点和缺点，方案一的全局路径更优，但是求解时间花费较长，对求解器的要求较高；方案二的全局路径更长，但是求解次数少、求解时间短，给求解器的压力小。

总的来讲，在地图较小、任务和机器人较少的时候，选择方案一更加合适，而在地图大、机器人和任务数都较多的情况下，选择方案二更加合适。

6 总结与展望

6.1 总结

多机器人的任务调度与路径规划在各领域都有广泛的研究和应用。随着物流行业的发展，物流仓库的订单数量剧增，人力形式已经不能满足装卸货物的需求，仓储机器人应运而生。但是仓储机器人任务调度与路径规划方面的研究

尚未完善。由于仓库环境的多样性，有很多种方法可以解决多机器人多任务分配和路径规划的问题。并且多机器人路径规划是NP-hard问题，得到全局最优解的计算量较大。因此，本项目的目的在于基于之前研究者的基础，自己建立一种模拟仓库环境的栅格地图，提出一种可行的、接近最优的解决方案。

本项目以Kiva systems等智能仓储机器人的运行模式为原型，单个机器人的运行流程包括前往任务点取货、送到人工拣货区域出货、送回空货架到原位。同时基于一定的现实背景，构建紧凑型或者稀疏型的栅格地图，设定一定数量的任务点和机器人。任务调度方面，采用简单的分配任务到最近的机器人的策略；路径规划方面，使用整数线性规划算法规划单个子区域内部的机器人无冲突最优路径，同时使用两种不同的方案实现了不同子区域之间机器人的无冲突交互，并比较了两者的性能差异。

多机器人任务调度和路径规划问题需要考虑路径规划算法、机器人碰撞规避算法、任务分配算法等，本项目提出的解决方案，在某些特定的应用场景有一定现实意义。

6.2 不足与展望

本文存在以下不足之处：

1. 完全基于理想化的机器人与地图模型，不考虑机器人转向、加速和减速等因素，缺少机器人运动学与动力学模型支撑。
2. 地图过大、机器人数量过多时，整数线性规划的求解速度仍然不够理想，可以采用更优化的整数线性规划模型，达到更高的求解速度。
3. 缺少现实情况的考虑，如机器人电量不足、机器人中途故障的解决等。

对于未来多机器人路径规划领域的发展，有以下的展望：

1. 传统路径规划算法，如 A*算法、Dijkstra 算法等，能够快速规划单机器人的最优路径，但是无法处理多机器人碰撞的情况；而本文提出的 ILP 法可以考虑机器人与障碍物、机器人之间的碰撞，但是求解速度不如传统路径规划算法。希望未来能够有更加快速、有效的算法，在解决单机器人路径规划的同时，能够考虑多机器人之间的交互性和碰撞避免。
2. 在仓储环境下的多机器人路径规划问题，需要考虑人为因素。本文的模型基于亚马逊的 Kiva systems，存在人工拣货环节。本文将这一环节理想化，现

实中如果人工拣货不及时，会导致机器人在拣货区排队，造成拥堵。因此，这一领域除了机器人路径规划本身，应该能形成一条完整的自动化流水线。

3. 本文的任务点都是等价的，只有优先级的不同。在现实运行中，有一些种类的货物数量可能较多，因此可以对货架的使用率进行统计，对于经常用到的货架，可以将其搬至距离拣货区较近的地方。

参考文献

- [1] Raffaello D'Andrea, Peter Wurman. 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.
- [2] 赵皎云. 海康机器人智能仓储系统助力工业智造升级[J]. 物流技术与应用, 2017, 22(05): 84-88.
- [3] 哈工大机器人集团推出 Bee Robot 智能仓储系统[J]. 物流技术与应用, 2018, 23(09): 137.
- [4] 朱大奇, 颜明重. 移动机器人路径规划技术综述[J]. 控制与决策, 2010, 25(7): 961-967.
- [5] 夏清松, 唐秋华, 张利平. 多仓储机器人协同路径规划与作业避碰[J/OL]. 信息与控制: 1-8.
- [6] 图片来源
http://www.machinedesign.com/sites/machinedesign.com/files/Amazon-Robots-Factory-View_1.jpg
- [7] 图片来源
<https://static.seattletimes.com/wp-content/uploads/2015/03/c1660f2a-c80e-11e4-ad5c-e0e608790d8d-780x519.jpg>
- [8] Dongdong Chen, Zhenyun Shi, Peijiang Yuan et al. Trajectory tracking control method and experiment of AGV[C]. 2016 IEEE 14th International Workshop on Advanced Motion Control (AMC), Auckland, 2016: 24-29.
- [9] Jingjin Yu, Steven M. LaValle. Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics[J]. IEEE Transactions on Robotics, 2016, 32(5): 1163-1177.
- [10] Tom Schouwenaars, Bart De Moor, Eric Feron and Jonathan How. Mixed Integer Programming for Multi-vehicle Path Planning[C]. 2001 European Control Conference (ECC), Porto, 2001: 2603-2608.
- [11] 图片来源 <http://www.gurobi.cn/UploadFiles/benchmark.jpg>

附录

作者简历

姓名：钱尘涤 性别：男 民族：汉

出生年月：1997-01-11 籍贯：浙江省平湖市

2012.09-2015.07 浙江省平湖中学

2015.09-2019.07 浙江大学 学士学位

获奖情况：

2015 年第八届新生英语演讲比赛 三等奖

2015 年 蓝田学园田径运动会 男子 100 米 第四名

2015-2016 年 浙江大学学业三等奖学金

2015-2016 年 浙江大学优秀学生三等奖学金

2015-2016 年 浙江大学优秀学生荣誉称号

2015-2016 年 浙江大学文体奖学金

2016 年外研社杯英语写作比赛 校级三等奖

2016 年 浙江省物理创新竞赛 三等奖

2016-2017 年 浙江大学学业三等奖学金

2016-2017 年 浙江大学优秀学生三等奖学金

2016-2017 年 浙江大学优秀学生荣誉称号

2016-2017 年 浙江大学优秀学生干部荣誉称号

2017-2018 年 浙江大学学业二等奖学金

2017-2018 年 浙江大学优秀学生二等奖学金

2017-2018 年 浙江大学优秀学生荣誉称号

2017-2018 年 电气工程学院英飞特外设奖学金

参加项目：浙江大学电气工程学院第 20 期大学生科研训练项目“仓储机器人集群的智能调度与路径规划研究”结题答辩良好，于浙江大学电气工程学院第七届大学生科研训练项目成果展示交流会获三等奖。

发表的学术论文：无

本科生毕业设计（论文）任务书

一、题目：

二、指导教师对毕业设计（论文）的进度安排及任务要求：

1、进度安排

2、任务要求

起讫日期 2018 年 11 月 19 日 至 2019 年 5 月 31 日

指导教师（签名）_____ 职称_____

三、系或研究所审核意见：

负责人（签名）_____

2019 年 月 日

毕业论文(设计)考核表

一、指导教师对毕业设计(论文)的评语:

本文研究了仓储环境下的多机器人任务调度与路径规划问题，完成了以下主要工作：基于整数线性规划实现了子区域内的无碰撞最优路径规划；多机器人任务分配与任务动态增删；提出了基于重规划和单行线的两种实现多子区域间机器人无碰撞交互的方法。完成了任务书规定的任务，论文内容完整，论证可靠，文字条理清楚，格式规范。

指导教师(签名) _____

年 月 日

二、答辩小组对毕业设计(论文)的答辩评语及总评成绩:

成 绩 比 例	文献综述 占 (10%)	开题报告 占 (15%)	外文翻译 占 (5%)	毕业设计(论文)质 量及答辩占 (62%)	成果评分 占 (8%)	总评成绩
分 值						

答辩小组负责人(签名) _____

年 月 日

浙江大学本科生毕业设计（论文）专家评阅意见表

毕业设计（论文）题目		多机器人任务调度与路径规划问题研究					
学生姓名	钱尘涤		学 号	3150100820		年级	15 级
所在学院	电气工程学院			专业	自动化(电气)		
指导教师姓名	郑荣濠	职 称	副教授	所在单位	电气工程学院		
本科生毕业设计（论文）评阅意见：（对论文选题、文献综述、外文翻译、研究内容与方法、创新点、论文质量与理论水平、论文写作规范与文风和修改建议等方面加以评阅） 该毕业设计研究仓储环境中的多机器人任务调度和路径规划问题，选题贴近实际应用；作者在广泛阅读文献的基础上完成了文献综述，外文翻译选取了近期发表的相关论文，完成情况较好；作者针对栅格地图上的多机器人路径规划问题，结合仓储应用场景，通过区域划分的方式，利用规划算法实现区域内部无冲突的多机器人最优路径规划，并提出了两种解决区域间路径冲突的方法，有自己独立创新的部分；论文学术基本规范，图表符合要求，理论水平合格。参考文献和目录部分格式不统一，请修改。同意答辩。							
同意答辩		同意修改后答辩			未达到答辩要求		
评阅人签名		评阅人职称		评阅人单位			

注：请评阅专家经综合评价后，在相应栏内打“√”。

浙江大学本科生毕业设计（论文）现场答辩记录表

学院：电气工程学院

毕业届别：2019届

学生姓名	钱尘涤	学号	3150100820	专业	自动化(电气)
毕业设计（论文）题目		多机器人任务调度与路径规划问题研究			
指导教师姓名	郑荣濠	职 称	副教授	所在单位	电气工程学院
答辩时间	2019年5月29日		答辩地点	电机楼 202	
答辩组成员（签名）					

本科生毕业设计（论文）答辩记录：（要求在答辩陈述和回答问题等方面具体加以记录与评价）

1. 图例中粉红色的圆点是什么？

答：已经分配到任务，正在前往任务点的机器人。

2. 两种区域划分方法哪种更有效率？

第一种方法路径更优，但是求解时间长，对求解器压力大；第二种方法节约计算，但是路径会相对更长。

3. 方法是否自己原创？

整数线性规划不是，两种区域划分方法是原创。

4. 有没有尝试过规模更大的区域？

尝试过，但是区域越大，规划越慢，当前的区域规模比较合理。

记录人（签名）：

_____年_____月_____日

答辩小组负责人（签名）：

_____年_____月_____日

第二部分

文献综述和开题报告

一、题目：

多机器人任务调度与路径规划问题研究

二、指导教师对文献综述、开题报告、外文翻译的具体要求

1. 广泛查找阅读多机器人任务调度和路径规划研究的相关文献，明确具体研究方向，总结该研究方向国内外研究现状及存在的问题，完成文献综述。（要求查阅相关文献 20 篇以上，其中外文文献不少于 10 篇，近五年文献不少于 10 篇）。
2. 在研究现状调研的基础上，针对多机器人任务调度和路径规划中存在的问题设计本次毕业设计的研究内容，提出具体的研究方案并进行可行性分析论证。
3. 外文翻译材料应选择近五年发表在研究领域内重要会议或者重要期刊上的论文。

指导教师（签名）_____

年 月 日

目 录

指导教师对文献综述和开题报告具体要求

一、 文献综述	1
1 背景介绍.....	1
2 国内外研究现状.....	2
2.1 研究方向及进展.....	2
2.2 存在问题.....	4
3 研究展望.....	5
二、 开题报告	6
1 问题提出的背景.....	6
1.1 背景介绍.....	6
1.2 本研究的意义和目的.....	6
2 论文的主要内容和技术路线.....	7
2.1 主要研究内容.....	7
2.2 技术路线.....	7
2.3 可行性分析.....	11
3 研究计划进度安排及预期目标.....	12
3.1 进度安排.....	12
3.2 预期目标.....	12
三、 外文翻译	15
四、 外文原文	33
参考文献	49

浙江大学本科生文献综述和开题报告考核表

一、文献综述

1 背景介绍

多机器人任务调度与路径规划问题在各种领域都有广泛的研究基础和应用价值，如多机器人协作SLAM、多机器人救援任务、多机器人仓储系统等。移动机器人相对于仿人机器人来说，技术上更易实现，成本优势明显，已经渗透到3C电子、汽车、新能源、物流仓库、机械加工等多个行业^[1]。

其中，仓储机器人的研究与应用在近几年得到蓬勃发展。互联网迅速发展，电子商务方兴未艾，电子商务的发展给仓储行业带来了重大的机遇与挑战。物流仓储的传统人工操作已经难以胜任物流的需求，急需智能仓储机器人代替人工进行货物分拣与运输。

在我国大力实施“互联网+”战略的形势下，“互联网+物流”已经成为物流行业的重要发展模式。所谓的“互联网+物流”，需要具备数字化、信息化、自动化等特性。互联网为物流行业提供了数据和信息，机器人控制技术的发展为物流的自动化提供了条件。但是一些公司的智能物流技术发展比较缓慢，不仅不利于自身的发展，也限制了全国范围物流水平的提高。为了发展物流体系，需要将其与智能化仓储结合起来，智能化仓储保证了仓储环节的高效运作，是物流体系发展的基本条件。

早在2003年，意大利学者Raffaello D' Andrea便着手研究物流机器人，即Kiva systems。Kiva systems使用数百个移动机器人和强大的控制软件来提供一个完整的实现方案：存储、移动和排序库存。产品不是存储在静态货架、流架或传送带上，而是存储在仓库中心的库存舱中，而操作员则站在周边的库存站^[2]。作为电商及物流企业的代表，亚马逊在2012年收购了Kiva systems的仓储机器人。

除了亚马逊的Kiva systems之外，各国各企业的仓储机器人产业都在蓬勃发展。日前，哈工大机器人的工业机器人本体、HRG智玲机器人、HRG众导机器人、机器人智能云服务平台等等在“2018世界机器人大会”上集中亮相^[3]。哈工大机器人的自主研发的机器人Bee Robot，由于其载重量大、电池续航能力强、体积小等优点，非常适应智能仓储系统的需求。

智能仓储机器人的发展，解放了人力，避免了人力资源管理可能出现的问题，多机器人的协同运作，提高了仓储的分拣、运货效率。如今已有不少智能仓储机器人已被投入到物流领域中使用，有包装机器人、堆码机器人、拣货搬运机器人、装卸机器人、自动机械手臂等等^[4]。在仓储配送过程中，仓储机器人独立完成装卸、搬运等动作，并完成在仓库内路径和姿态的规划，在人工拣货工作站，工作人员只需将货物取下。智能仓储机器人在未来必将因为大量的需求而不断发展。

2 国内外研究现状

2.1 研究方向及进展

对于多移动机器人系统（比如仓储机器人系统）而言，最核心的问题有以下几点：

首先，路径规划是机器人在自身的位姿已知的情况下，从自身的位置作为起点，根据一定的最优化目标（如路径最短或者能源最省等），规划出一条到目标位置的可行路径。同时，考虑到机器人的数量往往不止一个，还需要在路径规划的过程中考虑机器人的碰撞避免。

另外，任务分配需要用一种有效的算法，将多个任务分配给机器人去执行，任务本身可以独立，也可以互相关联；机器人可以独立工作，也可以协同完成同一组任务。

因此，仓储机器人集群的模型可以抽象为多机器人多任务的分配和多机器人的路径规划。需要解决的有以下问题：1. 多机器人的路径规划，2. 机器人之间的碰撞避免，3. 任务的最优分配。

在路径规划方面，A*算法是移动机器人路径规划一种比较热门的算法，不少学者对A*算法及其优化做过研究。基于智能仓储机器人的背景，地图一般采用栅格地图，用矩阵存储栅格的信息，包括格子的类型（货架或者道路）、机器人的位置等。A*算法是一种启发式的算法，它的代价函数包括估计函数（即从当前节点到目标节点的估计代价）和真实代价函数（即起始节点到当前节点的真实代价）。A*算法在Dijkstra算法的基础上增加了约束，规定了搜索方向，因

此搜索效率高，且可以得到最优的路径。目前在传统A*算法上有很多改进算法，如采用邻域矩阵的方式来进行障碍搜索，以分区加权方式来表达距离信息，以叉积距离来表达角度信息^[5]，让起点和终点同时并行搜索，并根据方向矢量原则选取当前节点的下一节点^[6]等。

同时，多机器人路径规划问题需要考虑的问题除了优化目标外，还有碰撞避免机制。有效规划多个机器人的路径，可以提高协同工作的效率，同时保障机器人运行的有序性和安全性。目前有学者提出一种多层次的多机器人路径规划模型，第一层是给各机器人规划路径，第二层是防止机器人之间的碰撞。若在发生碰撞栅格的上下存在自由栅格，则随机选择其中一个仓储机器人在原栅格处等待一个步长的时间，另一仓储机器人随机运动到碰撞栅格的上下某一个自由栅格再返回至碰撞栅格避开与该仓储机器人碰撞^[7]。

早期的移动机器人路径规划一般基于栅格地图、拓扑图等。随着移动机器人应用环境的发展，出现基于人工势场、遗传算法、蚁群算法、Dijkstra等算法的路径规划技术。目前移动机器人路径规划技术多在于多传感器、多种路径规划技术的融合^[8]。

在任务分配方面，机器人任务分配的最优化目标是多样的，可以是路径长度、完成时间等。如果仅考虑所有任务分配达到全局最优，可以使用匈牙利算法^[9]。每个任务分配在不同机器人的代价不同（这有一定的实际意义，比如不同机器人到达当前任务点的距离等），目标是将任务分别指派给机器人，使得全局的代价最小。

有学者提出一种简单的分布式的任务分配方案，每个机器人探测自身附近的任务点，如果单机器人遇到多个任务点，就选取运输距离近的，如果多机器人遇到单任务点，就按照机器人的行为习惯来决定任务分配^[10]。该方法的主要优点是高效、简便和鲁棒性。

对于机器人和任务数量都较多的情况而言，任务分配和路径规划的优化目标更加多样化。在多机器人系统中，时间消耗和能量消耗是估计任务分配效益的两个重要指标^[11]。有学者在多目标最优化的多机器人多任务分配中有过研究。总的代价函数由时间代价和机器人能耗代价组成，能耗代价又分为机器人移动

代价和装卸代价，可以根据不同场景，改变不同优化目标的权重，从而获得不同目标的最优解。

以上讨论的多机器人多任务分配问题的都是基于任务互不相干的情况。有学者提出一种问题，考虑任务之间相互关联的情况（MAP-GP问题）^[12]。该模型规定单个机器人处理的总任务和每个任务群中的任务数有限，并用拍卖算法进行求解。

此外，时序逻辑也广泛应用于多机器人多任务的问题。传统的路径规划方法可以规划出最优路径，但是考虑到机器人避障措施和本身的运动学特性等复杂的约束条件，时序逻辑相比之下更有优势。有学者给出了在给定路径选择的情况下，多机器人装卸货物的最优路径，以达到总完成时间最少的目的^[13]。类似的，有学者提出在多机器人环境搜索问题下的时序逻辑方法，控制机器人的数量，并有效协调大规模的机器人集群^[14]。

2.2 存在问题

首先，A*算法本身的特点，使得在规划目标时如果总估计函数有多个最小值，则不能保证搜索的路径最优。另外，如果考虑单行线等问题，则A*算法的估计函数比较难确定，不能用简单的欧式距离或者曼哈顿距离等距离函数。而目前的改进A*算法大都基于特定应用环境，要么牺牲时间效率，要么牺牲存储空间利用率，要么忽略一定的搜索精度^[15]。

其次，多机器人路径规划是NP-Hard问题，该方面研究还尚未成熟^[16]。尤其是多最优目标的规划，目前的算法复杂度比较高，而如果用一定的启发式算法，则可以达到近似最优解。

最后，目前的研究大多在理论算法阶段，而实体机器人需要考虑环境采样、定位、通信等因素，并且必须考虑机器人本身的运动学特点。从算法和仿真到实体机器人运行，还有很长的研究。

3 研究展望

随着应用环境的复杂化，越来越多的路径规划问题需要考虑环境的变化，而不是基于静态的地图。D*算法（即动态A*算法）在动态环境中寻路非常有效，机器人检查最短路径上下一节点或临近节点的变化情况。自从D*算法提出以来已有众多应用。目前已有D*算法应用于移动机器人^[17]。由于仓储智能机器人环境情况多变，相信动态的路径规划算法及其变式在仓储智能机器人领域也会有广泛的应用。

未来的多机器人任务调度与路径规划的研究有望于着眼新兴的智能算法，如模糊逻辑、遗传算法、神经网络等，都正在不断应用于多机器人路径规划的研究^[18]。这些算法的有效性和鲁棒性都比传统的路径规划方法优秀。

同时，多机器人任务调度与路径规划可能更多地基于分布式控制和自适应控制，而不是专家系统。自适应控制系统与广泛使用的专家系统的主要区别在于具有基于实时学习和预测的适应和预防冲突的能力^[19]。

二、开题报告

1 问题提出的背景

1.1 背景介绍

多机器人的任务调度与路径规划在各领域都有广泛的研究和应用。随着物流行业的发展，物流仓库的订单数量剧增，人力形式已经不能满足装卸货物的需求，仓储机器人应运而生。

文献综述中提到了亚马逊收购的Kiva systems。Kiva机器人的拣货流程包括拣出流程和归位流程。主要包括识别目标货架、锁定机器人、举起货架、高速运行、交叉运行、排队等候、工作人员取走货物、机器人回到储物区^[20]。Kiva机器人可以举起3000磅的货架，运送到打包去，待工作人员取走货物后将货架放回，解放了大量人力，产生了机器人搬运货架到人的新型物流模式。

本项目以智能仓储机器人的运行模式为原型，基于一定的现实背景，构建栅格地图，设定一定数量的任务点和机器人。为了实现任务调度和路径规划的基本功能，设计一定的最优任务分配方案，并用一定的路径规划方法规划机器人集群的无冲突路径。

1.2 本研究的目的和意义

近年来，物流行业的发展急需仓储处理货物效率的提高，但是仓储机器人任务调度与路径规划方面的研究尚未完善。由于仓库环境的多样性，有很多种方法可以解决多机器人多任务分配和路径规划的问题。并且多机器人路径规划是NP-hard问题，得到全局最优解的计算量较大。因此，本项目的目的在于基于之前研究者的基础，自己建立一种模拟仓库环境的栅格地图，提出一种可行的、接近最优的解决方案。

算法简单的路径规划方案在仓储中有一定的应用价值，尤其是仓库容量和机器人数急剧增大时，简单的算法计算量会更有优势。

多机器人任务调度和路径规划问题需要考虑路径规划算法、机器人碰撞规避算法、任务分配算法等，本项目提出的解决方案，在某些特定的应用场景有一定现实意义。

2 论文的主要内容和技术路线

2.1 主要研究内容

1. 任务分配到机器人的调度策略。首先要考虑的是初始状态，即所有机器人都空闲的状态，需要一种分配方案，使得在一定的指标下任务分配最优或接近最优；其次是当某一机器人完成当前任务时的新任务分配；最后应考虑动态添加任务时，任务的优先级。
2. 机器人的路径规划及其局部优化。使用一种简便、有效的路径规划方案，给单个机器人规划出从起点到目标节点的路径。
3. 机器人的碰撞规避。机器人的碰撞分为相遇碰撞和迎面碰撞^[21]。通过一定的策略避免机器人的碰撞。
4. 动态的任务添加。通过GUI或其他方案，实现程序执行过程中人工动态的货物添加。

2.2 技术路线

首先，本项目的先决条件是构造合理的栅格地图，通过合理的地图建模，为实现项目的基础功能做铺垫。其次，要在构造的地图上实现基本功能，即多任务的最优分配和多机器人的无冲突路径规划。最后，在完善了基础功能之后，可以适当添加一些附属功能。详细的技术路线如下：

1. 构造任务环境。基本的二维栅格地图如下，用二维矩阵存储地图的信息，0代表可以行走的路径，1代表障碍物或者货架，除了取货过程外不允许经过。利用MATLAB的绘图功能实现网格地图的绘制，白色代表可以行走的路径，黑色代表货架，灰色为打包区的位置（暂定）。绘图如2.1：

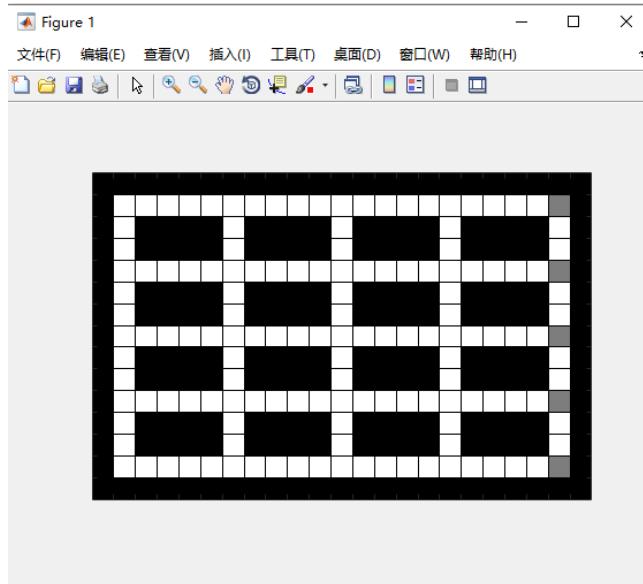


图2.1 棚格地图

每次程序的运行，随机生成一定数量的货物和机器人，以模拟各种情况。在地图上实现机器人和货物点的绘制，利用一定的绘图函数绘制当前时刻机器人和货物点的位置，如图2.2：

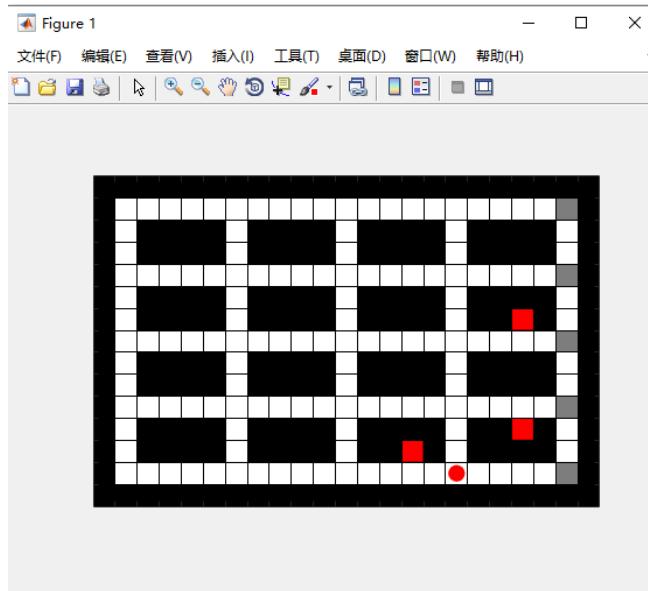


图2.2 机器人（圆）与货物点（方）

每次仿真程序运行，机器人的初始状态是空闲状态，分配到任务之后需要经过可行的路段赶到任务点，取走货架，在可行的道路上行走到打包区，停留一段时间后将货架运回起始地点。期间需要保持机器人的安全，不得与其他机

器人碰撞。每次移动的时候，所有机器人移动一个单位（除非为了避免碰撞或者没有任务而暂时不动），并定时一段时间使当前位置保留，再绘出下一时间点机器人的位置，方便人肉眼观察运行状态。

2. 多任务的分配。在初始状态，即所有机器人都空闲的状态，需要一种方案使得任务分配最优。拟采用匈牙利算法分配任务，匈牙利算法作为一种较成熟的方法，已经多次的应用在运输问题的求解上^[22]。将任务指派给对应的机器人，代价为当前任务到机器人的距离（计算距离的函数可采用欧式距离、曼哈顿距离等，但是由于规划机器人路径的算法限制，估计距离会有一些不精确），使得初始任务分配的总路径最短。由于机器人在一个时间节点内移动一个单位，总完成时间在数量上等于总距离。由于机器人到达任务点之后立即前往打包区，可视为从初始状态开始到程序结束，机器人一直在运行，因此不考虑初始任务分配的单机器人最大路径或单机器人的最大完成时间最小。如果初始机器人与货物的数量不一致，则考虑给min（机器人数，任务数）的货物和机器人进行规划。

在运行阶段，某一机器人完成当前任务时，会立即给其分配新任务，即该机器人立即前往新任务。

3. 多机器人的路径规划及其局部优化。计算上最简单的方案是在地图上给每个十字路口设置方向，方向（局部）如图2.3：

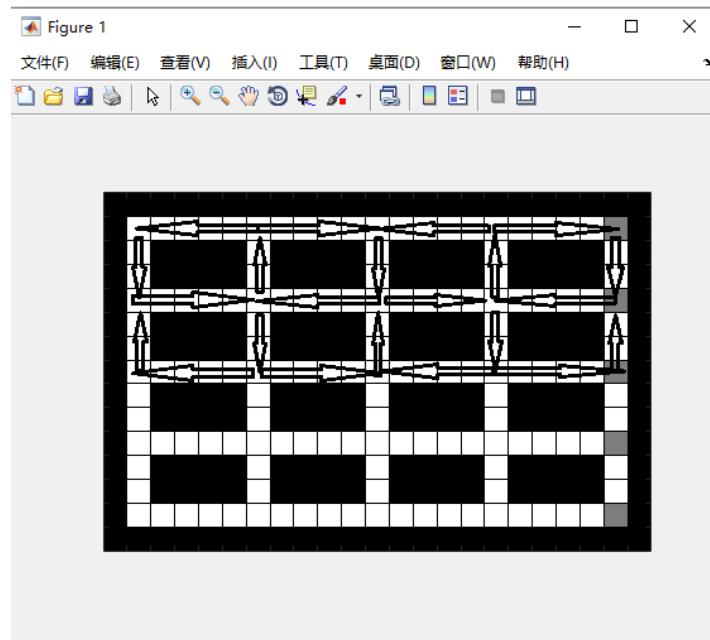


图2.3 全局单行线的方向（局部）

初始化每个十字路口的方向。如规定可以向左和向右，或者向上和向下。进而规定两个相邻路口之间的线上的方向，由一个十字路口指向另一个十字路口。以这种方式规定的单行线，在计算上非常简便，可以证明其有效性和可行性，且在机器人数很多时能够保证机器人不迎面相撞，只需考虑机器人在路口方向垂直的相撞。但是缺点是在机器人数少的情况下，效率非常低。

为了避免该方案产生的效率低下的问题，拟采用局部地图机器人路径规划的方法。实现方法是将全局地图分割成若干子地图，如图2.4：

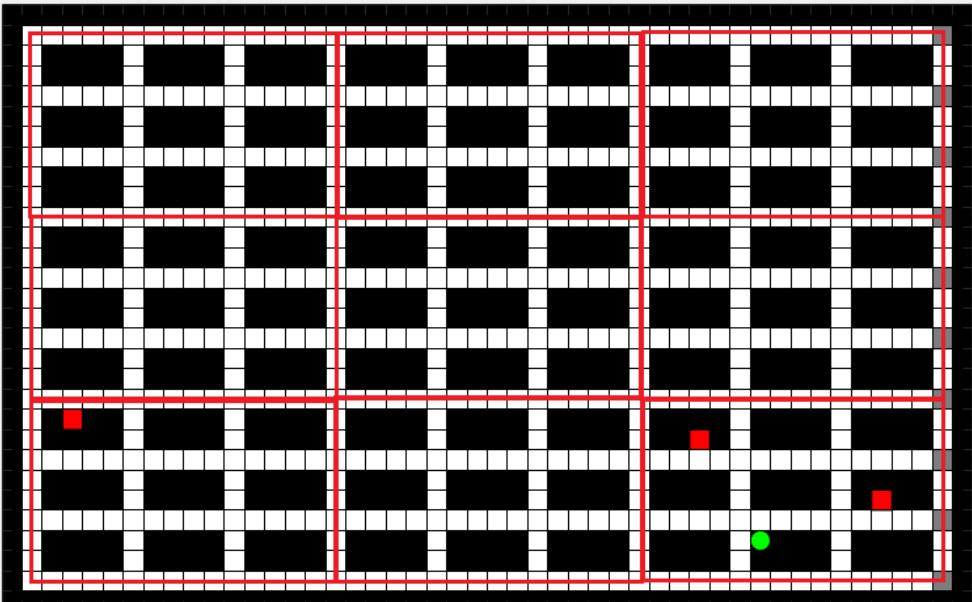


图2.4 将地图分割成若干子地图

该方法避免了全局地图规划的复杂性，只需考虑子地图内部的路径规划。当机器人被分配到任务时，只需确定自己当前所处的子地图和目标所在的子地图，然后确定大致方向（如应该向左和向上等），选择应该前往的下一个子地图。选择下一个子地图的根据是备选子地图的拥挤程度。例如机器人要到达目标点，必须向左和向上走，则考虑当前左边和上边的子地图的拥挤程度，选择拥挤度低的作为下一子地图。随后，机器人规划出一条前往下一子地图的路径，期间需要考虑的是在当前子地图中如何与其他机器人避免碰撞。到达下一子地图后，同样需要考虑选择下一子地图，并在当前子地图中规划路径。图上的每

个子地图都是等价的，每个子地图独立规划自己内部的机器人的路径，使自己内部的多机器人路径无冲突。每当有机器人离开或进入一个子地图，该子地图就需要重新规划内部路径。

该方案最明显的优势是局部规划的计算量远比全局规划小，且机器人较多时，全局的规划更有可能存在无解的情况，利用子地图进行局部规划，只需考虑局部地图的情况，解也会简洁一些。相对的缺点是要经常对子地图重新规划，且只能保证每次规划在子地图中最优，不能保证全局最优。

如果仿真结果表明机器人在子地图之间的道路上行走比较多，且出现迎面碰撞的情况，则可以结合上述两种方案，在子地图之间的道路上指定单行线，使子地图之间的路径无冲突。

4. 机器人的碰撞规避。利用局部地图的路径规划，预期不存在机器人迎面碰撞的情况。考虑到机器人在十字路口遇到的方向垂直的碰撞，规定机器人的优先级（例如可以规定正在将货物运往打包区的机器人优先级高于将货架运回原处的机器人等），优先级低的机器人原地等待一个时间单位，让优先级高的机器人先通过十字路口。由于每个机器人的速度相等，一般不存在机器人追尾相撞的问题，如果是前一个机器人由于优先级低原地等待的情况，则需要追尾的机器人同样原地等待。

5. 任务的动态添加。除了程序初始阶段随机生成的一定数量的任务以外，程序运行过程中可以利用GUI键盘或鼠标的动作，载入新的任务点，并规定其执行的优先级。

6. 机器人电量的考虑。如果要考虑机器人的电量，当机器人运行时间过长时需要对其进行重电，则要保证机器人前往充电桩前有足够的电量到达充电桩。如果机器人尚有足够的电量完成一项任务再前往充电桩，则不予充电。

2.3 可行性分析

通过阅读大量文献，已大致了解这一方向的众多的研究内容和方法。本项目旨在前人的研究基础上，实现一种计算简便、兼容性较高的算法，是前人在多机器人多任务规划方向上的延续。

在理论基础方面，目前的基础理论已经比较完备，如A*算法、Dijkstra算法等路径规划算法，以及整数线性规划、时态逻辑等机器人运动约束的研究都已经比较成熟。同时目前已经有Kiva、Bee Robot等仓储机器人实体机器人投入使用，已有相关的研究报道和技术应用，因此这一领域的可行性和应用性较强。本项目预期基于基础的路径规划算法，结合适当的路径约束，完成最终路径的规划，因此理论上具有可行性。

在科研条件方面，本项目基于MATLAB实现仿真，不需要实体机器人的硬件条件，在成本上比较节约。

本项目预期实现的软件，在地图环境、机器人和任务模型方面都有较强的可调整性，可适用于各种环境的模拟，具有较强的应用可行性。

因此，本项目具有较强的可行性。

3 研究计划进度安排及预期目标

3.1 进度安排

春学期第5周周一前：完成仿真程序架构及功能设计；

春学期第8周周一前：完成具有初步功能的多机器人任务调度与路径规划计算机仿真程序；

春学期第8周至夏学期第3周：修改完善仿真程序，撰写《毕业设计（论文）》；

夏学期第3周周一前：将完整的《毕业设计（论文）》初稿（电子版）发指导老师检查，并根据老师意见修改完善。

3.2 预期目标

本项目的基本目标是实现在任意大小的栅格地图上，利用一定规则进行多机器人的路径规划。任务和机器人数可以任意设置（大于0），任务点和机器人初始位置随机生成，以模拟任意初始情况。在初始情况时，将任务分配给相应的机器人，使机器人运行到任务点的总距离尽量小。机器人的基本任务是取

货架，运行到打包点，再把货架送回。机器人可以有效规划路径，并且机器人有碰撞避免机制。

利用MATLAB软件的编程和绘图功能实现这一过程的可视化。

将地图分为几个子地图，利用一定的运筹学原理实现局部地图的多机器人的路径规划；同时利用适当的单行线策略，使得子地图之间的机器人路径相互不冲突，达到局部最优。

考虑实际机器人的电量，机器人运行一段时间后需要去充电处充电。

可实现动态的任务添加，利用GUI进行人工添加任务点。

三、外文翻译

图上最优多机器人路径规划：完整的算法和有效的启发式

Jingjin Yu, Steven M. LaValle

摘要：我们研究四个最小化目标下的图上最优多机器人路径规划问题：完成时间（最后到达时间），（单机器人）最长运行距离，总到达时间，总距离。在预先确定了这些问题的差异，以及最优化的 NP 困难性之后，我们在这篇论文中专注于寻找解决这些 MPP 问题的算法上的解法。为了这个目标，我们首先在 MPP 和一种特殊类型的多流网络之间建立一个一对一的解决方案映射。基于这种等价和整数线性规划（ILP），我们设计了新的、完整的算法来优化这四个目标。特别是，我们计算最优完成时间的算法是第一个能够解决机器人-顶点比高达 100% 这样极具挑战性的问题的算法。然后，我们通过引入有原则的启发式算法进一步提高这些精确算法的计算性能，代价是略微损失最优性。基于 ILP 模型的算法与启发式算法的结合被证明是有效的，计算环境中密集分布的包含数百个机器人的问题的 1.x 最优解决方案，通常只需几秒钟。

1 介绍

我们研究图上多机器人最优路径规划问题 (MPP)，重点设计了完整的算法和有效的启发式算法。在 MPP 实例中，机器人被唯一地标记(即可区分的)并且被限制在任意连通图中。在没有碰撞的情况下，机器人可以在一个时间步长内从一个顶点移动到另一个相邻的顶点，这种碰撞发生在两个机器人同时移动到相同的顶点或沿着相同的边缘向相反的方向移动时。一个显著的特点是，我们的构想允许机器人在完全占用的周期同步旋转。这种公式更适合多机器人应用，尚未得到广泛研究 (除了例[1], [2] 以外)。在基本的 MPP 构想上，我们研究了四个常见的最小化目标：完成时间（最后到达时间），（单机器人）最长运行距离，总到达时间，总距离。这些全局目标与实际的多机器人应用直接相关，包括自主仓库系统[3]。例如，最小化完成时间等价于最小化任务完成的时间，而最小化总距离适用于最小化整个机器人车队的能耗。在相关著作[4] 中，我们证

明这些目标是不同的、NP 优化困难的，这表明求解最优 MPP 的努力应该是寻找有效的近最优算法。在本文中，我们试图实现这一目标，并提出了一种新的通用的优化求解 MPP 的框架。通过共同检查空间和时间维度，我们观察到 MPP 实例的解决方案与 MPP 问题派生的多货物网络流问题(多流的)的解决方案之间的一对一映射。基于这种等价，我们将 MPP 问题转化为一个整数线性规划(ILP)模型，该模型可以用 ILP 求解器求解。ILP 的通用性允许对所有四个目标进行编码，从而生成完整的优化算法。在此基础上，我们进一步引入了一些启发式算法，以提高算法的性能，同时略微降低了解的最优性。我们的方法在计算接近最优的最小完成时间解决方案方面特别有效，计算底层图中密集分布的数百个机器人的 1.x 最优解决方案，通常只需几秒钟。

相关著作：多机器人路径规划问题，在其众多的架构中，已被积极研究了几十年[2], [5] - [19]。作为一种通用子例程，多机器人无碰撞路径规划在跨装配任务中的应用[20], [21]，疏散[22]，编队控制[23]-[27]，定位[28]，微滴控制[29], [30]，物体运输[31], [32]，搜索与救援[33]等。对多机器人路径和运动规划的一般主题进行更全面的综述，参见[34]-[36]及其参考文献。

基于图形的多机器人路径规划问题的算法研究可以追溯到 1879 年[37]，这是本论文的重点，Story 在其中观察到 15 数字推盘 [38]的可行性依赖于游戏的奇偶性。15 数字推盘游戏是一种有限制的 MPP 实例，在 4×4 的网格中把 15 个有标签的游戏块从一些初始结构移动到目标结构。限制是在一个步骤中，只有靠近空顶点的一个游戏块可以移动到空顶点；另一方面，多个移动机器人可以同时移动。15 数字推盘游戏的泛化在[39]中介绍了，从 4×4 , 15 片的问题推广到 $n-1$ 片， n 个顶点的双连通图。已证明在非二部图的情况下，实例总是可行的，并给出了一个隐式算法。当图是二部图(如 15 数字推盘游戏)时，所有的铺设配置被分成两组大小相同的大小，使同一组中的任意两种配置形成一个可行的实例。在[40]中引入了进一步的推广，允许在一个有 n 个顶点的图上有 $p < n$ 个石子。对于这个问题，我们提供了一个复杂度 $O(n^3)$ 的算法来解决一个实例或者判定该实例是不可行的。

随着计算机游戏和多机器人系统的普及，并行运动被引入，鹅卵石被机器人(或代理)代替。在可行性方面，本文所研究的 MPP 问题在 $O(n^3)$ 时间复杂度

[41]下也是可解的。为了区分这些公式，我们将不允许机器人在完全占用的循环内进行循环旋转的公式表示为无循环 MPP。直到最近，关于 MPP 的算法研究主要集中在无循环的情况下。由于该问题是易于处理的[40]，大多数无循环 MPP 的算法研究都将重点放在了最优化上。[10]、[12]、[13]、[42]、[43]算法通过巧妙使用基本运算，可以在某种形式的完备性保证下，快速解决疑难问题。这些算法没有最优化保证，但生成的解决方案往往比 Kornhauser 等人给出的 $O(n^3)$ 界质量好得多。有关次优方法的更多讨论和参考资料，请参见[42]和[43]。

在最优化方面，大多数算法的结果都是探索如何限制由多个机器人引起的指数搜索空间增长。最早的这种算法之一，局部修复 A* (LRA*) [6]，同时规划机器人路径，并在冲突发生时执行局部修复。针对 LRA* 的(局部性)缺点，加窗的分层协作 A* (WHCA*) 提出使用时空窗口，在限制搜索空间大小[8]的同时，为解决局部冲突提供更多选择。一种叫做次维展开的技术在复杂的环境中表现良好；然而，机器人的密度相对较低(根据论文，每个机器人有 104 个格子)。在[1]和[2]中，没有对一个实例进行不可知的剖分，而是探索了独立检测(ID)的自然思想，只将多个机器人联合考虑(指数搜索空间增长的来源)是必要的。通过将每个合法移动视为“操作符”的操作符分解(OD)，作者生成了在计算总时间或距离最优解方面非常有效的算法(ID、OD + ID 和相关变体)。我们指出 ID 和 OD + ID 支持处理周期(即，它们适用于 MPP，而不是无循环 MPP)。最近，增加成本树搜索(CTS[45])和基于冲突的搜索(CBS[46])进一步推动了无循环 MPP 的性能。设计用于最小化完成时间的算法也被尝试过，例如[47]，但是随着机器人-顶点比的增加，解决方案的质量会迅速下降。

针对连续域多机器人路径规划问题，多种求解方法被提出。一个称为速度障碍的典型的方法[48]-[50]显式检查速度-时间空间协调机器人的运动。在[30]中，采用混合整数规划模型对机器人交互进行编码。在[51]中，我们探索了一种基于时空视角的方法。在[52]中，基于 A* 的搜索是在从连续环境中抽象出来的离散路线图上执行的。在[53]中，为了有效搜索多机器人路线图，提出了离散 RRT 算法。离散 MPP 的算法，无论有无循环，也有助于解决连续问题[16][54]。

贡献：我们研究了一个 n 顶点连通图上允许最多 n 个机器人的最优 MPP 公式，我们认为它更适合多机器人应用。这个公式并没有得到广泛的研究，可

能是由于处理机器人的循环转动固有的困难。除了问题的新颖性之外，这项工作还带来了一些算法方面的贡献。首先，基于 MPP 与多流的等价关系，我们建立了一个通用的、新颖的求解框架，允许使用 ILP 模型对最优 MPP 问题进行压缩编码。我们证明了该框架可以很容易地生成最小化完成时间(最后到达时间)、(单机器人)最长运行距离、总到达时间、总距离的完整算法，这是 MPP 最常见的四个全局目标。所得到的算法，特别是计算最小完成时间的算法，在解决机器人顶点比高达 100% 的具有挑战性的问题实例时是非常有效的。其次，我们引入了一些有原则的启发式方法，特别是在时间范围内分割 MPP 实例的 k 路分割启发式方法，以牺牲一些解决方案的最优性，从而使精确算法获得相当大的性能提升。有了这些启发式，我们就可以扩展我们的算法来处理数百个极其密集的机器人的问题，同时维护 1.x 解决方案最优。最后，我们成功利用 ILP 攻击最优 MPP 的尝试表明，在这个问题领域，ILP 方法与直接搜索方法的性能是相当的，特别是当机器人的数量变大的时候。这是令人惊讶的，因为 ILP 解决程序不是专门为 MPP 设计的。

本文的其余部分组织如下。在第二节中，我们定义了最优 MPP 问题，并简要回顾了网络流。在第三节中，我们建立了 MPP 与网络流之间的等价关系。我们在第四节中推导出完整的算法，并在第五节中继续描述性能提升启发式。我们在第六节中对算法进行了评估，并在第七节中得出结论。本文部分基于 [17] [55] [56]。与 [17] 和 [55] 相比，除了证明了 k 路分裂启发式算法的加入显著提高了计算性能外，我们还大大扩展了基于 ILP 的算法框架的通用性，该框架现在支持所有常见的基于全局时间和距离的目标。

2 预备知识

现在我们定义了 MPP 和本文研究的最优目标。在问题陈述之后，我们提供了对网络流的简要回顾。

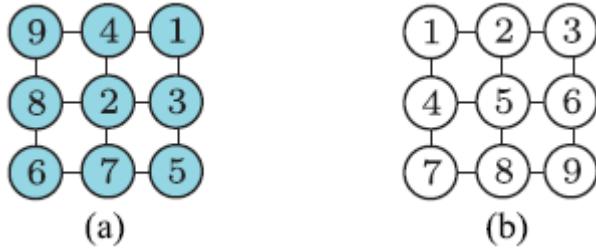


图 1 (a) 9 块问题 (b) 希望的构型

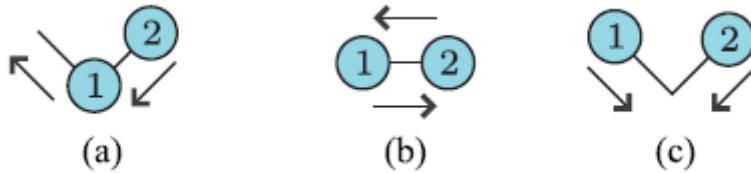


图 2 两个机器人的一些可行和不可行的移动。(a) 可行的同步移动。(b) 两个机器人“正面”碰撞时不可行的同步运动。(c) 两个机器人在一个顶点“相遇”时不可行的同步移动。

A. 图上多机器人路径规划

令 $G = (V, E)$ 是一个简单的无向连通图, $V = \{v_i\}$ 为顶点集, $E = \{\{v_i, v_j\}\}$ 为边集。令 $R = \{r_1, \dots, r_n\}$ 为 n 个机器人的集合。机器人的配置是一个从 R 到 V 的单射映射, 即, 对于给定的构型, 两个机器人 $r_i, r_j (i \neq j)$ 位于 V 上的不同顶点。在任何给定的时刻 $t = 0, 1, \dots$ 机器人假设一个配置。机器人的起始(初始)和目标配置相应地记为 x_I, x_G 。图 1(a) 显示了一种九个机器人在 3×3 的网格图的可能的配置。图 1(b) 给出了一种可能的目标配置, 其中机器人的排序基于行主排序。在离散时间步长过程中, 每个机器人可以保持静止或移动到相邻的顶点。要正式地描述一个方案, 令路径是一个映射 $p_i : Z^+ \rightarrow V$, 其中 $Z^+ := N \cup \{0\}$ 。路径 p_i 如果满足以下性质就是可行的:

- 1) $p_i(0) = x_I(r_i)$
- 2) 对每个 i , 存在最小 $t_i^f \in Z^+$ 使得 $p_i(t_i^f) = x_G(r_i)$
- 3) 对任意 $t \geq t_i^f$, $p_i(t) = x_G(r_i)$

4) 对任意 $0 \leq t < t_i^f, \{p_i(t), p_i(t+1)\} \in E$, 或 $p_i(t) = p_i(t+1)$ (如果 $p_i(t) = p_i(t+1)$,

机器人 r_i 在 t 和 $t+1$ 在同一顶点 $p_i(t)$ 上)

我们说两条路径相撞, 如果存在 $k \in \mathbb{Z}^+$ 使得 $p_i(t) = p_j(t)$ (相遇) 或者 $(p_i(t) = p_j(t+1)) \wedge (p_j(t) = p_i(t+1))$ (相向)。如图 2 所示为两个机器人在一个时间步长的可行和不可行的移动。MPP 问题的定义如下。

问题 1 (图上多机器人路径规划): 给定一个元组 (G, R, x_l, x_G) , 找到一个路径集合 $P = \{p_1, \dots, p_n\}$ 使得对于所有 $1 \leq i < j < n$: (i) p_i 是机器人 r_i 的可行路径, (ii) p_i 和 p_j 不相撞。

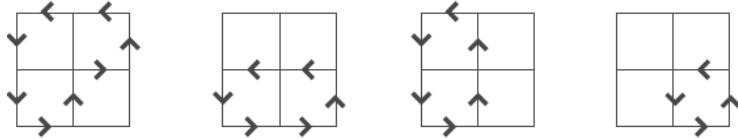


图 3 图 1 中 9 块谜题算法的四步解。有向边缘显示了机器人在这些边缘尾部的移动方向

例如, 图 1(a) 和 (b) 定义了一个 3×3 网格 MPP 问题。我们把这个特殊的问题称为 9 块问题, 它可以很容易地推广到 N^2 块问题。我们选择这个名字是因为它与经典的 15 数字推盘游戏和 $(N^2 - 1)$ 数字推盘游戏相似 [57]。尽管表面上的相似之处, 一个 N^2 问题有 N^2 个机器人, 且不需要一个 $(N^2 - 1)$ 数字推盘所需的空的交换顶点。在一个 N^2 难题中, 机器人可以沿着多个不相交的循环同步旋转。

备注: 除了少数例外(如[2]), 大多数现有的离散多机器人路径规划问题的研究都要求将空顶点作为交换空间。在这些设想中, 在一个时间步长中, 一个非相交的机器人链只有在链头移动到一个先前未占用的顶点时才可以同时移动。相比之下, 我们的 MPP 构想允许机器人在完全占用的周期内同步旋转(参见图 1 和图 3)。这意味着即使机器人的数量等于顶点的数量, 机器人仍然可以在不相交循环上移动。我们注意到 MPP 可以在多项式时间内求解, 可行性测试只需要线性时间 [41]。

B. 最优方案

令 $P = \{p_1, \dots, p_n\}$ 为一个 MPP 情况的任意可行解。对于路径 $p_i \in P$, 令 $\text{len}(p_i)$ 为路径 p_i 的长度, 是机器人 r_i 在 p_i 上改变其驻留顶点的总次数。一个机器人, 沿着 p_i 路径, 可以多次访问同一个顶点。回想一下 t_i^f 表示机器人 r_i 到达的时间。在最优 MPP 公式的研究中, 我们考察了四个常见的目标, 其中两个关注时间最优化, 另一个关注距离最优化。

目标 1 (完成时间): 计算一条最小化 $\max_{1 \leq i \leq n} t_i^f$ 的路径。

目标 2 (最大距离): 计算一条最小化 $\max_{1 \leq i \leq n} \text{len}(p_i)$ 的路径。

目标 3 (总到达时间): 计算一条最小化 $\sum_{i=1}^n t_i^f$ 的路径。

目标 4 (总距离): 计算一条最小化 $\sum_{i=1}^n \text{len}(p_i)$ 的路径。

图 1 中 9 块难题的最小完成时间四步解如图 3 所示。解决方案是最优的, 因为机器人 9 距离目标有 4 步远。

C. 网络流回顾

一个网络 $\mathfrak{N} = (G, c_1, c_2, S)$ 包括了一个有向图 $G = (V, E)$, $c_1, c_2 : E \rightarrow \mathbb{Z}^+$ 是图相应的指定容量和有向边的代价, $S \subset V$ 是源和阱的集合。令 $S = S^+ \cup S^-$, S^+ 是源的集合, S^- 是阱的集合, $S^+ \cap S^- = \emptyset$ 。对于一个顶点 $v \in V$, 令 $\delta^+(v)$ (相应地, $\delta^-(v)$) 表示 G 的通向 (或离开) v 的边。可行的(静态)流 S^+, S^- 在这个网络 \mathfrak{N} 是一个满足弧容量约束的映射 $f : E \rightarrow \mathbb{Z}^+$

$$\forall e \in E, f(e) \leq c_1(e) \quad (1)$$

非端点处的流守恒约束:

$$\forall v \in V \setminus S, \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \quad (2)$$

以及端点处的流守恒约束：

$$F(f) = \sum_{v \in S^+} (\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e)) = \sum_{v \in S^-} (\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e)) \quad (3)$$

$F(f)$ 的值是流 f 的值。经典的(单一货物)最大流量问题提出以下问题：给定一个网络 \mathbf{N} ，通过网络可以流经的最大 $F(f)$ 是多少？最小成本最大流量问题进一步要求流量在所有最大流量中具有最小总成本。也就是说，我们想要在所有的最大流量中找到一个也能使流量最小化的流量：

$$\sum_{e \in E} c_2(e) \cdot f(e) \quad (4)$$

到目前为止描述的网络流公式只考虑一种货物，对应于所有机器人都是可互换的。这些问题可以通过多货物流或简单的多流来捕获。我们有一个流函数 f_i 对于货物集合 C 中的每个商品 i ，而不是只有一个流函数 f 。约束 (1) (2) (3) 变为

$$\forall i \in C, \forall e \in E, \sum_i f_i(e) \leq c_1(e) \quad (5)$$

$$\forall i \in C, \forall v \in V \setminus S, \sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) = 0 \quad (6)$$

$$\forall i \in C, \sum_{v \in S^+} (\sum_{e \in \delta^-(v)} f_i(e) - \sum_{e \in \delta^+(v)} f_i(e)) = \sum_{v \in S^-} (\sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e)) \quad (7)$$

最大流量和最小成本流量问题也可以在多流量设置下提出；我们忽略细节。我们对网络流的回顾只涉及到与本文相关的几个方面；有关网络流主题的详细介绍，请参见[58]和[59]及其参考资料。注意，这里所述的多流模型有时也称为整数多流，因为 f_i 必须具有整数值。

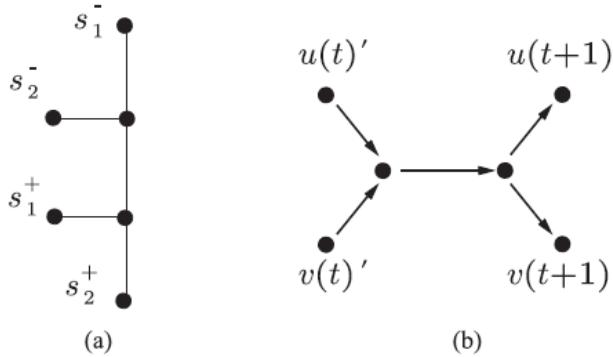
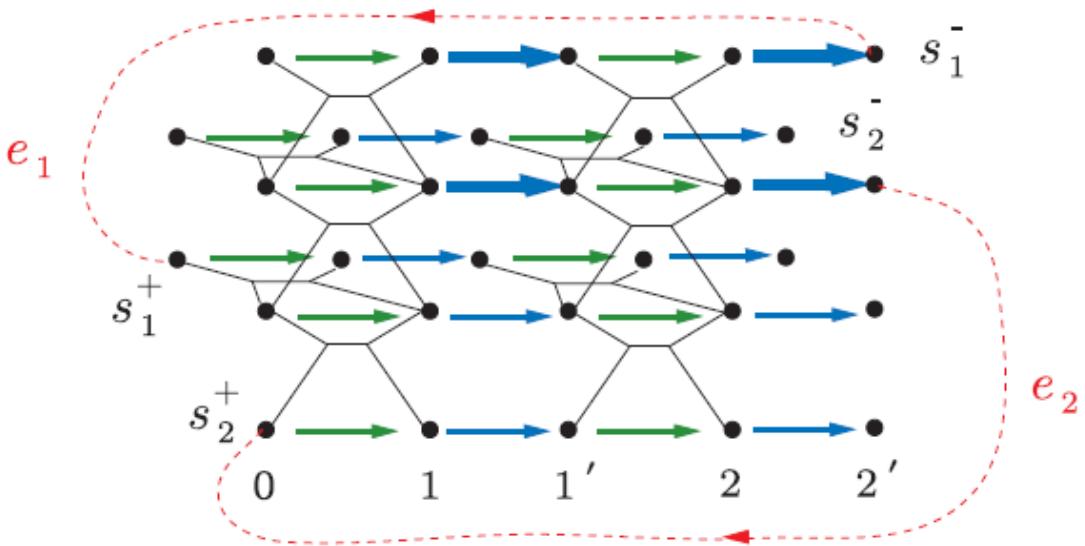


图 4 (a) 简单的 G 。(b) 通过时间分割无向边缘的合并分割小工具，用于强制执行迎头碰撞约束

3 从多机器人路径规划到多流

最优 MPP 问题与网络流问题之间存在着紧密的算法联系。最大(单货物)流量问题一般承认有效(低次多项式时间)算法解[59]，而最大多流量问题是一个众所周知的 NP 困难问题，甚至很难近似[60]。为了反映单货物流和多货物流之间的差异，在 MPP 问题中，如果只有一组可互换的机器人，（在这里，对于一个组，哪个机器人去哪个目标并不重要，只要分配给组的所有目标位置都被来自同一组的机器人占据），那么许多最优公式都允许多项式时间算法[17]。然而，一旦一组机器人分裂成两组或两组以上，为这些机器人寻找最优路径就变得非常棘手[4]。最优 MPP 和多流之间的明显相似性可能最好的解释是通过将基于图形的 MPP 问题简化为网络流问题。这个约简也将成为我们算法解的基础。

为了描述约简，我们以图 4(a) 中的无向图 G 为例，起始节点为 $\{s_i^+\}, 1, 2$ ，目标节点为 $\{s_i^-\}, 1, 2$ 。MPP 的一个实例是 $(G, \{r_1, r_2\}, x_I : r_i \mapsto s_i^+, x_G : r_i \mapsto s_i^-)$ 。我们将把问题归结为网络流问题 $\mathbf{N} = (G', c_1, c_2, S)$ 。通过构建一个图 G 的时间扩展版本的网络，可以明确考虑机器人在空间和时间上的相互作用，从而进行约简。本文中推导的时间展开可以看作是网络流随时间的执行[61]。要实现这种扩展，首先必须确定一个时间范围。对于不同的最优目标，扩展时间范围，某个自然数 T ，通常是不同的；现在我们假设 T 是固定的。为了开始构建网络，我们创建了 G 顶点的 $2T+1$ 个副本，索引为 $0, 1, 1', \dots$ ，如图 5 所示。对于每个顶点 $v \in G$ ，副本表示为 $v(0) = v(0)', v(1), v(1)', v(2), \dots, v(T)'$ 。对于每条边 $\{u, v\} \in G$ 和时间段 $t, t+1, 0 \leq t < T$ ，图 4(b) 所示的合并分割折法是在 $u(t)', v(t)'$ 和 $u(t+1)', v(t+1)'$ 之间添加的（图 5 中省略了小工具的箭头，因为它们很小）。

图 5 在基本图 4(a) 上展开时间范围为 $T = 2$ 的时扩展网络

对于小工具，我们将单位容量分配给所有的弧，单位成本分配给水平中间的弧，其他四个弧的成本为零。合并分割法确保两个机器人不能在同一时间在基础图形的边缘上以相反的方向移动，从而防止两个机器人之间的正面碰撞。为了完成图 5 的构造，对于每个顶点 $v \in G$ ，我们在每两个连续的副本之间添加一个弧（例如我们添加弧 $(v(0), v(1)), (v(1), v(1)'), \dots, (v(T), v(T)')$ ）。它们对应于图 5 中的绿色和蓝色弧线。对于所有的绿色弧线，我们分配它们的单位容量和成本；对于所有的蓝色弧线，我们分配它们单位容量和零成本。绿色的弧线允许机器人在时间步长中停留在一个顶点上，而蓝色的弧线则确保每个顶点最多能容纳一个机器人，从而加强了满足碰撞约束。图 5 中的时间扩展网络（弧 e_1 和 e_2 除外，它们很快就会变得相关）是所需的 G' 。对于集合 $S = S^+ \cup S^-$ ，我们可以简单地令 $S^+ = \{v(0) : v \in \{s_i^+\}\}$ 和 $S^- = \{v(T)' : v \in \{s_i^-\}\}$ 。 $\aleph = (G', c_1, c_2, S)$ 现在完备了，我们已经将 MPP 问题简化为 \aleph 上的整数多流问题，每个 R 的机器人作为一种货物。

定理 1：令 (G, R, x_I, x_G) 是一个 MPP 实例。在由 (G, R, x_I, x_G) 构造的具有 T 时间步长的时扩网络 \aleph 上，其解集（最大时间步长为 T ）与流值 n 的整数最大多流解之间存在二元关系。

证明 1：内射性。假设 $P = \{p_1, \dots, p_n\}$ (n 是机器人的数量) 是一个 MPP 实例的解。对于每个 p_i 和每个时间段 $t = 0, \dots, T$ ，我们标注时间段 t 上的时扩网络 G' 中的 $p_i(t)$ 和 $p_i(t)'$ 的副本(回顾一下 $p_i(t)$ 相当于 G 的一个顶点)。按顺序连接 G' 的这些顶点(有一种独特的方法)在 \aleph 上产生一个流 f_i 单元(在连接到适当的 S^+, S^- 中的源和汇顶点后，这个微不足道)。很明显，如果两个路径 p_i, p_j 没有碰撞，那么对应的流 f_i 和 f_j 在 \aleph 上是顶点不相交的路径，因此不违反任何流约束。既然任意两条 P 中的路径都不相撞，相应的流 $\{f_1, \dots, f_n\}$ 的集合是可行的，且在 \aleph 上是最大值。

满射性。假设 $\{f_1, \dots, f_n\}$ 是 \aleph 上的整数最大流，也就暗示着对于所有 i ， $|f_i| = 1$ 。首先我们确定任何流的对 f_i 和 f_j 是顶点不相交的。为了看到这一点，我们注意到 f_i 和 f_j (都是单位流)不能共享相同的源或汇顶点，这是由于蓝色弧强制的 \aleph 的单位容量结构。如果 f_i 和 f_j 在时间步长 $t > 0$ 时共享某个非汇顶点 v ，那么两个流必须经过相同的蓝色圆弧[参见图 4(b)]， v 要么是头，要么是尾，这是不可能的。因此， f_i 和 f_j 是 \aleph 上的顶点不相交的。我们可以很容易地将每个流 f_i 转换为对应的路径 p_i (删除额外的源顶点、汇顶点、小物件中间的顶点以及蓝色弧的尾部顶点之后)，并保证没有 p_i, p_j 将会因为相遇碰撞而发生碰撞。通过构造 \aleph ，我们使用的小工具确保迎头相撞也是不可能的。集合 $\{p_1, \dots, p_n\}$ 是 $MPP(G, R, x_I, x_G)$ 的解。

备注：具有单元独立流的多流问题也可以看作是一种多路径规划问题，称为边缘不相交路径问题[62]。

4 基于整数线性规划的最优多机器人路径规划算法

由于在目标 1-4 之上优化 MPP 解决方案在计算上是难以解决的，因此将 MPP 减少到多流问题并不会使这些优化 MPP 问题变得更容易。然而，有了网络流公

式(参见第二-c 节)，就有可能为最优 MPP 公式建立 ILP 模型。这些 ILP 模型可以用强大的线性规划软件包来解决。相比于通过启发式增强的，目标通常是一个重要但有限的结构问题的 A*算法，基于 ILP 的算法这里提出具体的问题不了解结构。因此，基于 ILP 的算法似乎更有能力解决更广泛的 MPP 问题，特别是机器人顶点比高的复杂 MPP 实例。在本部分中，我们为目标 1-4 中的每一个构建 ILP 模型，假设固定时间跨度 T 。也就是说，这些模型只针对特定的 T 优化给定的目标。然后讨论完整的算法及其完整性。

A. 最小化完成时间

可以使用最大流公式计算 MPP 实例 $I = (G, R, x_I, x_G)$ 的最小完成时间解。固定一个时间跨度 T ，设 $\aleph = (G', c_1, c_2, S)$ 为 I 的时间展开网络，将 S 中每对对应的起点和目标顶点从目标到起点连接起来，将一组 n 个环回弧添加到 G' 中。我们使用 e_j 's 表示 G' 的弧线，让 n 环回弧占据前 n 指数， $e_j, 1 \leq j \leq n$ ， r_j 的弧连接目标顶点开始 r_j 的顶点。例如，对于图 5 中的 G' ， e_1 和 e_2 是 r_1 和 r_2 的环回弧。环回弧具有单位容量和零成本。接下来，对于每个弧 $e_j \in G'$ (包括环回弧)，创建 n 个二进制变量 $x_{1,j}, \dots, x_{n,j}$ 对应流过机器人 $1 \leq i \leq n$ 的弧。也就是说， $x_{i,j} = 1$ 当且仅当机器人 r_i 经过 G' 中的 e_j 。变量 $x_{i,j}$'s 必须满足两个弧容量约束和一个流量守恒约束

$$\forall e_j, \sum_{i=1}^n x_{i,j} \leq 1 \quad (8)$$

$$\begin{aligned} & \forall 1 \leq i, j \leq n, i \neq j, x_{i,j} = 0 \\ & \forall v \in G' \text{ and } 1 \leq i \leq n, \sum_{e_j \in \delta^+(v)} x_{i,j} = \sum_{e_j \in \delta^-(v)} x_{i,j} \end{aligned} \quad (9)$$

目标函数为

$$\max \sum_{1 \leq i \leq n} x_{i,i} \quad (10)$$

B. 最小化单机器人的旅行距离

为了最小化机器人行走的最大距离，网络和变量的创建与最小完成时间设置相同；约束 (8) (9) 不变。因为我们希望将所有机器人发送到它们的目标，所以可能会通过约束强

制执行最大流量

$$\forall 1 \leq i \leq n, x_{i,i} = 1 \quad (11)$$

为了对 min-max 目标函数进行编码，我们引入了一个额外的整数变量 x_{\max} 并添加了约束

$$\forall 1 \leq i \leq n, \sum_{e_j \in G', j > n} c_2(e_j) \cdot x_{i,j} \leq x_{\max} \quad (12)$$

对于一个固定的 i ，(12) 的左边表示机器人 r_i 所走的距离。注意对于 $j < n$ ，

$x_{i,j} = 0$ 。目标函数为

$$\min x_{\max} \quad (13)$$

C. 最小化总到达时间

为了最小化总到达时间，继承了最小完成时间 ILP 模型(8)、(9)和(11)中的网络和变量。为了表示目标函数，对于每一个时间段 $1 \leq t \leq T$ 和每个 $v = x_G(r_i)$ ， $1 \leq i \leq n$ ，我们创造一个二元变量 y_i^t 。然后，我们给现有变量一些新的指标。回顾对于每个弧 $e_j = (v(t), v(t)') \in G'$ （例如，图 5 中四个加粗的蓝色弧线），一个变量 $x_{i,j}$ 被创造出来。总共有 nT 个这样的变量。这里我们给予这些变量第二个指标 x_i^t 。也就是说， x_i^t 是一个二元变量，指示弧 $(v(t), v(t)') \in G'$ ， $v = x_G(r_i)$ 是否被机器人 r_i 使用了。

给定一个固定 T 的网络，如果约束 (8) (9) (11) 被满足的话，初始的 MPP 问题就有可行解。在这种情况下， $x_i^T = 1$ ，对于 $1 \leq i \leq n$ 。我们令 $y_i^T = x_i^T$ 。然后对于每个 y_i^t ， $1 \leq t < T$ 由 x_i^t 和 y_i^{t+1} 迭代地定义为

$$y_i^t \geq y_i^{t+1} + x_i^t - 1, y_i^t \leq y_i^{t+1}, y_i^t \leq x_i^t \quad (14)$$

有效地，(14) 代表了一个 x_i^t 和 y_i^{t+1} 之间的逻辑与，并把结果存储在 y_i^t 。最后， $y_i^t = 1$ 的最小 t 是机器人 r_i 到达目标（并停下）的时间。因此，对于每个

$1 \leq i \leq n$, $\sum_{t=1}^T y_i^t$ 是从 r_i 到达目标到时间 T 的时间步长。 $T - \sum_{t=1}^T y_i^t$ 是 r_i 所花费的时间。为了最小化总到达时间, 目标函数为

$$\min(nT - \sum_{1 \leq i \leq n, 1 \leq t \leq T} y_i^t) \quad (15)$$

D. 最小化总距离

从最大距离最小的 ILP 模型出发, 我们只需要改变计算最小总距离解的目标函数。我们不需要变量 x_{\max} , 只需将目标函数更新为

$$\min \sum_{e_j \in G^*, j > n, 1 \leq i \leq n} c_2(e_j) \cdot x_{i,j} \quad (16)$$

E. 算法结构和完备性

算法 1 给出了优化求解 MPP 的一般算法结构。这里, 我们假设问题 (G, R, x_I, x_G) 是可行的, 可以在 [41] 中被查到。然后, 我们进行保守估计的最小时间跨度 T_{\min} 和最大时间跨度 T_{\max} , 必须有一个 $T \in [T_{\min}, T_{\max}]$ 对应于最优解的时间跨度。然后算法通过 $[T_{\min}, T_{\max}]$ 进行搜索, 找到 T 和最优解。具体的算法 1-4 相应表示为 MINMAKESPAN, MINMAXDIST, MINTOTALTIME, 和 MINTOTALDIST, 我们现在补充如何估计它们的 T_{\min}, T_{\max} 。

当 $T_{\min} = 0$ 总是有效时, 我们可以更好地将 T_{\min} 设置为所有机器人的最大值, 即每个机器人的最短路径长度, 忽略所有其他机器人。 T_{\min} 明显适用于所有四个目标。对于 MINMAKESPAN, 我们可以把 T_{\max} 设置为一个可行解的需要时间, 可以用 [41] 计算。最坏的情况, $T = O(|V|^3)$ 。这建立了 MINMAKESPAN 的完备性, 我们可以假设我们对 MINMAKESPAN 有一个最优的 T_{opt} 。

对于 MINMAXDIST, 我们令 $T_{\max} = nT_{opt}$ 。这是正确的, 因为在每一步中, 至少有一个机器人必须移动一段距离。在 $nT_{opt} + 1$ 时间之后, 通过鸽巢原理, 一些机器人必然移动了 $T_{opt} + 1$ 步。然而, 由于 T_{opt} 对 MINMAKESPAN 是最优的, 有一个

解使得没有机器人运动了超过 T_{opt} 。这产生了矛盾。

相同的 $T_{max} = nT_{opt}$ 在 MINTOTALTIME 上有效的原因是不同的。在 $nT_{opt} + 1$ 步之后，至少一个机器人花了这么多时间。另一方面，MINMAKESPAN 最优解不超过 nT_{opt} 。因此 MINTOTALTIME 的最佳解决方案不需要超过 nT_{opt} 的时间跨度。对于 MINTOTALDIST， $T_{max} = nT_{opt}$ 经常有效。在 $nT_{opt} + 1$ 步之后，机器人的总旅行时间一定超过 nT_{opt} 。然而，MINTOTALTIME 的最优解不需要超过 nT_{opt} 的步数。我们已经证明了四个算法的完备性。

命题 2：MINMAKESPAN, MINMAXDIST, MINTOTALTIME, 和 MINTOTALDIST 都是完备的。

5 有效计算近似最优解的启发式方法

我们已经证明了基于 ILP 的算法是完备的，并且总是能得到最优解。它们在处理相对较小但高度受限的问题方面的性能实际上相当好(见第六节)。但是随着问题规模的增长(例如，随着图 G 和机器人数 n 的增加)，计算时间迅速增加。从实际的角度来看，快速计算质量好但不是最优的解决方案可能比等待最优解决方案的时间长得多。在本节中，我们将介绍一些实现这一目标的启发式方法，并特别关注具有最小完成时间的计算解决方案。

A. 构建更紧凑的模型

为了从求解器中提取最佳性能，使用精益模型是有益的(例如，列和行最少)。到目前为止，我们的重点是提供一个通用的基于多流的框架，以便可以轻松构建 ILP 模型。在模型转换阶段，模型可以进一步简化。本节讨论的启发式旨在使约束(8)和(9)的表示更紧凑。因此，它们适用于所有目标。

更好的碰撞约束编码：在建立网络流模型(如图 5)时，我们使用了一个合并分割小工具(见图 4(b))来加强迎头碰撞约束和额外的时间步长(如图 5 中的蓝色弧线)来避免碰撞。当我们将其转换成线性约束时，这些结构可以被简化，从而得到如图 6 所示的更紧凑的结构。

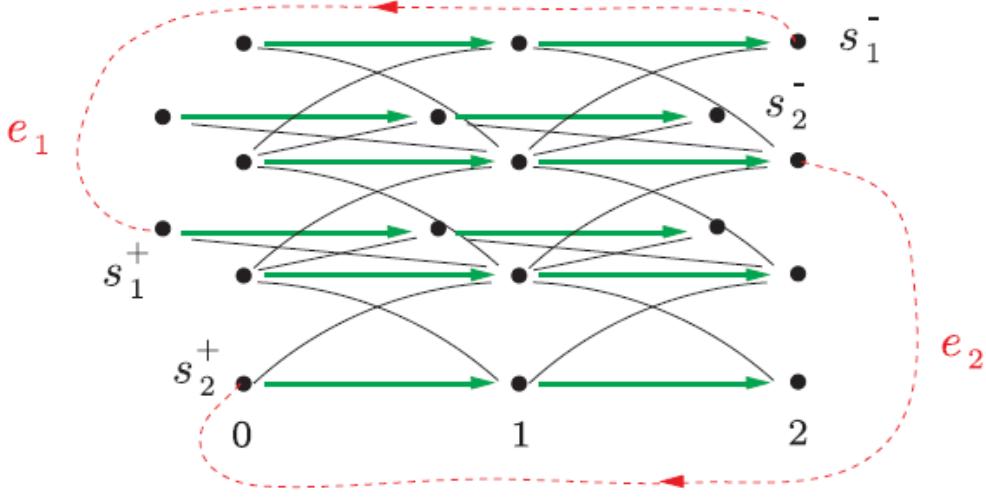


图 6 图 5 所示网络流图的更简洁的表示

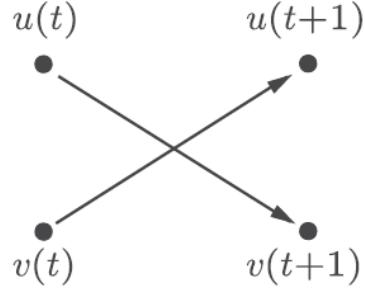


图 7 简化的合并分割小工具强制迎头碰撞约束

在新的结构中，每个合并拆分小工具现在有两个弧，而不是五个。并且，蓝色的边没了。更新的小工具对于时间段 t 和 $t+1$ 之内的边 $\{u, v\} \in E$ 如图 7 所示（注意由于蓝边被移去，一些例如 $v(t)'$ 的顶点不需要了）。也就是说，每个机器人只需要两个变量，而不是五个。将这些二元变量表示为机器人 r_i 的 $x_{i,(u(t),v(t+1))}$ 和 $x_{i,(v(t),u(t+1))}$ ，单个小物件的迎头碰撞约束可以方便地编码为

$$\sum_{i=1}^n x_{i,(u(t),v(t+1))} + \sum_{i=1}^n x_{i,(v(t),u(t+1))} \leq 1 \quad (17)$$

然后，为了执行满足碰撞约束，例如，在顶点 $v(t)$ 处，我们只需要使用 $v(t)$ 处最多一条出弧，例如

$$\sum_{e_j \in \delta^-(v(t)), 1 \leq i \leq n} x_{i,j} \leq 1 \quad (18)$$

新的 ILP 模型的尺寸大约是原始模型的一半。

可达性分析：在时间展开图中，存在冗余二进制(弧)变量，这些变量永远不可能为真，因为某些弧永远无法到达。例如，在图 6 中，在 $t=0$ 时，唯一可能使用的输出弧是来自 s_1^+ 和 s_2^+ 的弧。其余的可以安全地移除。一般来说，对于每个机器人 r_i ，根据其从起始点到目标点的可达性，可以删除相当数量的二进制变量 $x_{i,j}$'s。

B. 在时域上分治

在评估基于 ILP 模型的最优完成时间算法时，我们发现 ILP 求解器的运行时间随着模型大小的增加呈指数增长。

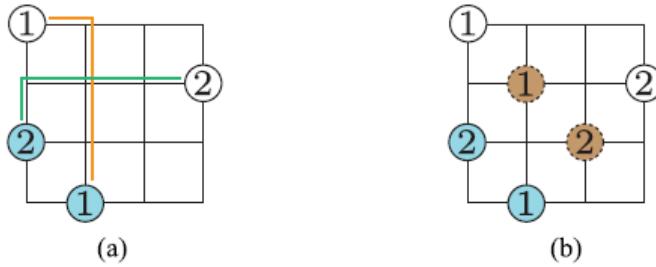


图 8 (a) 简单的双机器人问题。(b) 时间分治实例

这阻止了算法在超过几十个机器人的实例上很好地执行。这一观察结果虽然妨碍了精确算法的有效性，但却为高效启发式提供了有用的见解。我们发现当机器人的顶点比不是很高时(例如不接近一)，相对较小的 ILP 模型对 ILP 求解器来说没有太大的挑战。即使有大量的机器人这仍然正确(例如，数以百计)。为了将基于 ILP 模型的方法应用于更具有挑战性的问题(例如，使用数百个机器人快速解决问题)，我们只需限制提供给求解器的单个 ILP 模型的大小。一种方法是在时域内分治。我们用一个简单的例子(见图 8)来说明这个想法。

在图 8 (a) 中，我们在 3×3 的网格上有两个机器人的规划问题。为了执行启发式，我们首先为每一对起始点和目标点计算一条最短路径。在这种情况下，我们分别得到机器人 1 和 2 的橙色和绿色路径。然后，如果我们决定将问题分割成两个更小的问题，对于每条路径，它都被分割成两个长度相等或接近相等的部分，中间节点被设为中间目标。在我们的例子中，我们可以将机器人 1 的

中间目标位置设置为左上角的(1, 1) [图 8(b) 中标记为 1 的棕色圆盘]。对于机器人 2, 由于中间位置与机器人 1 的中间位置重合, 我们选择一个备选的空闲位置作为机器人 2 的中间目标, 在这种情况下(2, 2), 从左上角开始。第一个实例的中间目标也将作为第二个实例的起始位置。这将生成两个子实例, 它们都需要每个步骤进行时间扩展, 从而有效地使单个 ILP 模型的大小大约是需要四个步骤进行时间扩展的原始 ILP 模型的一半。一般来说, 我们可以将一个问题在时域内任意分割成许多较小的实例。

如果一个问题被分成 k 个子问题, 我们称之为启发式的 k 路分割。因为划分是随时间推移的, 所以被划分的实例之间没有交互。一旦我们为每个子实例获得了解方案, 就可以通过连接结果将解决方案粘在一起。在实际应用中, 该启发式算法在不影响路径生成空间的前提下, 显著提高了算法性能; 我们在计算实验中观察到一致的加速。

备注: k 路分割启发式在设计上特别适合于完成时间目标, 因为完成时间目标在分割子问题上的可加性。除了完成时间目标外, 启发式算法也很好地适用于距离目标 (例如目标 2 和 4), 只要找到距离最优解所需的时间范围与最小完成时间解所需的时间范围相差不大。启发式不直接应用于目标 3, 因为总时间不是分裂子问题的加法。例如, 假设进行双向分割, 每个子问题的时间范围为 $T/2$ 。如果机器人 r_i 没有在第一个子问题 (即 $0 \leq t \leq T/2$), 它对总距离的贡献为 0。然而, 如果 r_i 在第二个子问题 (即 $T/2 \leq t \leq T$), r_i 将贡献至少 $T/2$ 的总到达时间。尽管如此, k 路分割在这种情况下仍然有用, 因为我们可以使用它来快速计算执行时间展开的初始 T 。

四、外文原文

Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics

Jingjin Yu and Steven M. LaValle

Abstract—We study optimal multirobot path planning on graphs (MPP) over four minimization objectives: the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. Having established previously that these objectives are distinct and NP-hard to optimize, in this paper, we focus on efficient algorithmic solutions for solving these optimal MPP problems. Toward this goal, we first establish a one-to-one solution mapping between MPP and a special type of multiflow network. Based on this equivalence and integer linear programming (ILP), we design novel and complete algorithms for optimizing over each of the four objectives. In particular, our exact algorithm for computing optimal makespan solutions is a first that is capable of solving extremely challenging problems with robot-vertex ratios as high as 100%. Then, we further improve the computational performance of these exact algorithms through the introduction of principled heuristics, at the expense of slight optimality loss. The combination of ILP model based algorithms and the heuristics proves to be highly effective, allowing the computation of $1.x$ -optimal solutions for problems containing hundreds of robots, densely populated in the environment, often in just seconds.

I. INTRODUCTION

We study the problem of optimal *multirobot path planning on graphs* (MPP), focusing on the design of *complete algorithms and effective heuristics*. In an MPP instance, the robots are uniquely labeled (i.e., distinguishable) and are confined to an arbitrary connected graph. A robot may move from a vertex to an adjacent one in one time step in the absence of collision, which occurs when two robots simultaneously move to the same vertex or along the same edge in opposing directions. A distinguishing feature is that our formulation allows robots on fully occupied cycles to rotate synchronously. Such a formulation, more appropriate for multirobot applications, has not been widely studied (except, e.g., [1], [2]). Over the basic MPP formulation, we look at four commonly studied minimization objectives: the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. These global objectives have direct relevance toward real-world multirobot applications, including autonomous

Manuscript received July 13, 2015; revised April 3, 2016; accepted June 19, 2016. Date of publication August 10, 2016; date of current version September 30, 2016. This paper was recommended for publication by Associate Editor H. Kress-Gazit and Editor C. Torras upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant 1328018 (National Robotics Initiative) and Grant 1617744 (Division of Information and Intelligent Systems-Robot Intelligence).

J. Yu is with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08901 USA (e-mail: jingjin.yu@cs.rutgers.edu).

S. M. LaValle is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL 61820 USA (e-mail: lavalle@illinois.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TRO.2016.2593448

warehouse systems [3]. For example, minimizing makespan is equivalent to minimizing the task completion time, whereas minimizing total distance is applicable to minimizing the fuel consumption of the entire fleet of robots.

In a related work [4], we show that these objectives are pairwise distinct and NP-hard to optimize, suggesting that efforts on solving optimal MPP should be directed at finding effective near-optimal algorithms. In this paper, we make an attempt toward this goal and propose a novel yet general framework for solving MPP optimally. Examining space and time dimensions jointly, we observe a one-to-one mapping between a solution for an MPP instance and that for a multicommodity network flow problem (multiflow¹) derived from the MPP problem. Based on the equivalence, we translate the MPP problem into an integer linear programming (ILP) model solvable using an ILP solver. The generality of ILP allows the encoding of all four objectives to yield complete optimization algorithms. From here, we further introduce several heuristics to boost the algorithmic performance at a slight loss of solution optimality. Our method is especially effective in computing near-optimal minimum makespan solutions, capable of computing $1.x$ -optimal solutions for hundreds of robots densely populated on the underlying graph, often in just seconds.

Related work: Multirobot path planning problems, in its many formulations, have been actively studied for decades [2], [5]–[19]. As a universal subroutine, collision-free path planning for multiple robots finds applications in tasks spanning assembly [20], [21], evacuation [22], formation control [23]–[27], localization [28], microdroplet manipulation [29], [30], object transportation [31], [32], search and rescue [33], etc. See [34]–[36] and the references therein for a more comprehensive review on the general subject of multirobot path and motion planning.

The algorithmic study of graph-based multirobot path planning problems, which is the focus of this paper, can be traced to 1879 [37], in which Story makes the observation that the feasibility of the 15-puzzle [38] depends on the *parity* of the game. The 15-puzzle is a restricted MPP instance moving 15 labeled game pieces on a 4×4 grid, from some initial configuration to some goal configuration. The restriction is that only a single game piece near the single empty vertex may move to the empty vertex in a step; multiple mobile robots, on the other hand, could move simultaneously. A generalization of the 15-puzzle is introduced in [39], extending the problem from 15 game pieces on a 4×4 grid to $n - 1$ labeled *pebbles* on an n -vertex, 2-connected graph. It is shown, together with an implied algorithm, that an instance is always feasible if the graph is nonbipartite.

¹multiflow is used here to refer the multicommodity flow problem.

When the graph is bipartite (such as the 15-puzzle), all pebble configurations are split into two groups of equal size such that any two configurations in the same group form a feasible instance. A further generalization is introduced in [40], allowing $p < n$ pebbles on a graph with n vertices. For this problem, an $O(n^3)$ algorithm is provided to solve an instance or decide that the instance is infeasible.

As computer games and multirobot systems gain popularity, concurrent movements are introduced and pebbles are replaced with robots (or agents). On the feasibility side, the MPP problem studied in this paper is shown to be solvable also in $O(n^3)$ time [41]. To distinguish the formulations, we denote the formulation that does not allow cyclic rotations of robots along fully occupied cycles as *cycle-free* MPP. Until recently, the majority of algorithmic study on MPP is on the cycle-free case. Since the problem is shown to be tractable [40], most algorithmic study of cycle-free MPP put some emphasis on optimality. Through the clever use of primitive operations, algorithms from [10], [12], [13], [42], and [43] could quickly solve difficult problems with some form of completeness guarantees. These algorithms do not have optimality guarantees, but the produced solutions are often of much better quality than the $O(n^3)$ bound given by Kornhauser *et al.* [40]. For more discussion and references on suboptimal methods, see [42] and [43].

On the optimality side, most algorithmic results explore ways to limit the exponential search space growth induced by multiple robots. One of the first such algorithms, local repair A* (LRA*) [6], plans robot paths simultaneously and performs local repairs when conflicts arise. Focusing on fixing the (locality) shortcomings of LRA*, windowed hierarchical cooperative A* (WHCA*) proposes using a space-time window to allow more choices for resolving local conflicts while simultaneously limiting the search space size [8]. A technique called subdimensional expansion is shown to perform well in complex environments [44]; the robot density is, however, relatively low (104 cells per robot according to the paper). In [1] and [2], instead of applying an agnostic dissection of an instance, the natural idea of independence detection (ID) is explored to only consider multiple robots jointly (the source of exponential search space growth) as necessary. With operator decomposition (OD) that treats each legal move as an “operator,” the authors produced algorithms (ID, OD + ID, and related variants) that prove to be quite effective in computing total time- or distance-optimal solutions. We point out that ID and OD + ID have support for handling cycles (i.e., they apply to MPP instead of cycle-free MPP). More recently, increasing cost tree search (ICTS [45]) and conflict-based search (CBS [46]) have further pushed the performance on cycle-free MPP. Algorithms designed for minimizing makespan have also been attempted, e.g., [47], but the solution quality degrades rapidly as the robot-vertex ratio increases.

Many approaches have also been proposed for solving multirobot path planning problems in the continuous domain. A representative method called velocity obstacles [48]–[50] explicitly examines velocity-time space for coordinating robot motions. In [30], mixed integer programming models are employed to encode the robot interactions. A method based on the space-time perspective, similar to ours, is explored in [51]. In

[52], an A*-based search is performed over a discrete roadmap abstracted from the continuous environment. In [53], discrete-RRT is proposed for the efficient search of multirobot roadmaps. Algorithms for discrete MPP, cycle-free or not, have also helped solving continuous problems [16], [54].

Contributions: We study the optimal MPP formulation allowing up to n robots on a n -vertex connected graph, which we believe is better suited for multirobot applications. The formulation is not widely studied, perhaps due to the inherent difficulty in handling cyclic rotations of robots. Beside the novelty of the problem, this work brings several algorithmic contributions. First, based on the equivalence relationship between MPP and multiflow, we establish a general and novel solution framework, allowing the compact encoding of optimal MPP problems using ILP models. We show that the framework readily produces complete algorithms for minimizing the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance, which are perhaps the four most common global objectives for MPP. The resulting algorithms, in particular the one for computing the minimum makespan, are highly effective in solving challenging problem instances with a robot-vertex ratio up to 100%. Second, we introduce several principled heuristics, in particular a k -way split heuristic that divides an MPP instance over the time horizon, to give the exact algorithms a sizable performance boost at the expense of some loss of solution optimality. With these heuristics, we are able to extend our algorithms to tackle problems with several hundred robots that are extremely densely populated, while at the same time maintain 1.x solution optimality. Last, but not least, our successful exploitation of ILP to attack optimal MPP shows that the ILP method is competitive with direct search methods in this problem domain, especially when the number of robots becomes large. This is surprising because ILP solvers are not designed specifically for MPP.

The rest of the paper is organized as follows. In Section II, we define optimal MPP problems and provide a brief review of network flow. We establish the equivalence relationship between MPP and network flow in Section III. We derive complete algorithms in Section IV and continue to describe the performance-boosting heuristics in Section V. We evaluate the algorithms in Section VI and conclude in Section VII. This paper is partly based on [17], [55], [56].² In comparison to [17] and [55], beside demonstrating significantly improved computational performance due to the addition of the k -way split heuristic, we have substantially extended the generality of our ILP-based algorithmic framework, which now supports all common global time- and distance-based objectives.

II. PRELIMINARIES

We now define MPP and the optimality objectives studied in this paper. Following the problem statements, we provide a brief review of *network flow*.

²[56] is a preliminary poster presentation.

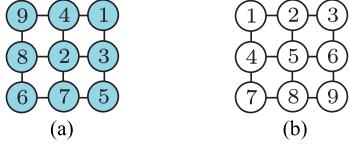


Fig. 1. (a) 9-puzzle problem. (b) Desired goal configuration.

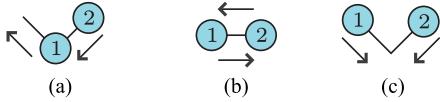


Fig. 2. Some feasible and infeasible moves for two robots. (a) Feasible synchronous move. (b) Infeasible synchronous move in which two robot collide “head-on.” (c) Infeasible synchronous move in which two robots “meet” at a vertex.

A. Multirobot Path Planning on Graphs

Let $G = (V, E)$ be a connected, undirected, simple graph, with $V = \{v_i\}$ being the vertex set and $E = \{\{v_i, v_j\}\}$ the edge set. Let $R = \{r_1, \dots, r_n\}$ be a set of n robots. A *configuration* of the robots is an injective map from R to V , i.e., for a given configuration, two robots r_i and r_j ($i \neq j$) occupy different vertices of V . At any given time step $t = 0, 1, \dots$, the robots assume a configuration. The *start (initial)* and *goal* configurations of the robots are denoted as x_I and x_G , respectively. Fig. 1(a) shows a possible configuration of nine robots on a 3×3 grid graph. Fig. 1(b) shows a possible goal configuration, in which the robots are ordered based on *row-major ordering*.³

During a discrete time step, each robot may either remain stationary or move to an adjacent vertex. To formally describe a plan, let a *path* be a map $p_i : \mathbb{Z}^+ \rightarrow V$, in which $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$. A path p_i is *feasible* if it satisfies the following properties:

- 1) $p_i(0) = x_I(r_i)$.
- 2) For each i , there exists a smallest $t_i^f \in \mathbb{Z}^+$ such that $p_i(t_i^f) = x_G(r_i)$.
- 3) For any $t \geq t_i^f$, $p_i(t) = x_G(r_i)$.
- 4) For any $0 \leq t < t_i^f$, $\{p_i(t), p_i(t+1)\} \in E$ or $p_i(t) = p_i(t+1)$ (if $p_i(t) = p_i(t+1)$, robot r_i stays at vertex $p_i(t)$ between the time steps t and $t+1$).

We say that two paths p_i, p_j are in *collision* if there exists $k \in \mathbb{Z}^+$ such that $p_i(t) = p_j(t)$ (meet collision) or $(p_i(t) = p_j(t+1)) \wedge (p_i(t+1) = p_j(t))$ (head-on collision). As an illustration, Fig. 2 shows the feasible and infeasible moves for two robots during a single time step.⁴ The MPPproblem is defined as follows.

Problem 1 (Multirobot path planning on graphs): Given a 4-tuple (G, R, x_I, x_G) , find a set of paths $P = \{p_1, \dots, p_n\}$

³In this paper, we use shaded discs to mark start locations of robots and discs without shades for goal locations.

⁴We assume that the graph G allows only “meet” or “head-on” collisions. The assumption is mild. For example, a (arbitrary dimensional) grid with unit edge distance is such a graph for robots with radii of no more than $\sqrt{2}/4$ (two robots traveling on adjacent edges are the closest to each other when they are in the middle of these edges).

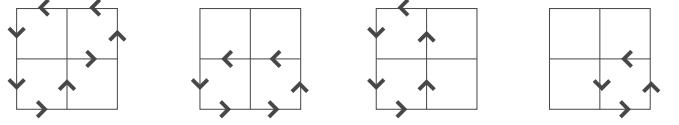


Fig. 3. Four-step solution from our algorithm for the 9-puzzle from Fig. 1. The directed edges show the moving directions of the robots at the tail of these edges.

such that for all $1 \leq i < j \leq n$: (i) p_i is a feasible path for robot r_i , and (ii) p_i and p_j are not in collision.

For example, Fig. 1(a) and (b) defines an MPP problem on the 3×3 grid. We call this particular problem the *9-puzzle* problem, which readily generalizes to N^2 -puzzles. We pick the name due to its similarity with the classic 15-puzzle and $(N^2 - 1)$ -puzzles [57]. Despite the superficial similarities, an N^2 -puzzle has N^2 robots and does not require an empty *swap* vertex that is required for the $(N^2 - 1)$ -puzzle. In an N^2 -puzzle, robots may rotate synchronously along multiple disjoint nonintersecting cycles.

Remark: With a few exceptions (e.g., [2]), most existing studies on discrete multirobot path planning problems require empty vertices as swap spaces. In these formulations, in a time step, a nonintersecting chain of robots may move simultaneously only if the head of the chain is moving into a previously unoccupied vertex. In contrast, our MPP formulation allows synchronized rotations of robots along fully occupied cycles (see, e.g., Figs. 1 and 3). This implies that even when the number of robots equals the number of vertices, robots can still move on disjoint cycles. We note that MPP can be solved in polynomial time with a feasibility test taking only linear time [41]. \triangle

B. Optimal Formulations

Let $P = \{p_1, \dots, p_n\}$ be an arbitrary feasible solution to some fixed MPP instance. For a path $p_i \in P$, let $\text{len}(p_i)$ denote the length of the path p_i , which is the total number of times the robot r_i changes its residing vertex while following p_i . A robot, following a path p_i , may visit the same vertex multiple times. Recall that t_i^f denotes the arrival time of robot r_i . In the study of optimal MPP formulations, we examine four common objectives with two focusing on time optimality and two focusing on distance optimality.

Objective 1 (Makespan): Compute a path set P that minimizes $\max_{1 \leq i \leq n} t_i^f$.

Objective 2 (Maximum Distance): Compute a path set P that minimizes $\max_{1 \leq i \leq n} \text{len}(p_i)$.

Objective 3 (Total Arrival Time): Compute a path set P that minimizes $\sum_{i=1}^n t_i^f$.

Objective 4 (Total Distance): Compute a path set P that minimizes $\sum_{i=1}^n \text{len}(p_i)$.

A four-step minimum makespan solution to the 9-puzzle problem from Fig. 1 is illustrated in Fig. 3. The solution is optimal because robot 9 is four steps away from its goal.

C. Network Flow Review

A *network* $\mathcal{N} = (G, c_1, c_2, S)$ consists of a *directed graph* $G = (V, E)$ with $c_1, c_2 : E \rightarrow \mathbb{Z}^+$ being the maps specifying

capacities and *costs* over directed edges (arcs), respectively, and $S \subset V$ as the set of *sources* and *sinks*. Let $S = S^+ \cup S^-$, with S^+ denoting the set of source vertices, S^- denoting the set of sink vertices, and $S^+ \cap S^- = \emptyset$. For a vertex $v \in V$, let $\delta^+(v)$ [resp., $\delta^-(v)$] denote the set of arcs of G going to (resp., leaving) v . A feasible (static) S^+, S^- -flow on this network \mathcal{N} is a map $f : E \rightarrow \mathbb{Z}^+$ that satisfies arc capacity constraints

$$\forall e \in E, \quad f(e) \leq c_1(e) \quad (1)$$

the flow conservation constraints at nonterminal vertices

$$\forall v \in V \setminus S, \quad \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \quad (2)$$

and the flow conservation constraints at terminal vertices

$$\begin{aligned} F(f) &= \sum_{v \in S^+} \left(\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \right) \\ &= \sum_{v \in S^-} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right). \end{aligned} \quad (3)$$

The quantity $F(f)$ is called the *value* of the flow f . The classic (single commodity) *maximum flow* problem asks the following question: Given a network \mathcal{N} , what is the maximum $F(f)$ that can be pushed through the network? The *minimum cost maximum flow* problem further requires the flow to have a minimum total cost among all maximum flows. That is, we want to find a flow among all maximum flows that also minimizes the quantity

$$\sum_{e \in E} c_2(e) \cdot f(e). \quad (4)$$

The network flow formulation described so far only considers a *single commodity*, corresponding to all robots being interchangeable. For general MPP formulations, the robots are distinct and must be treated as different commodities. Such problems can be captured with *multicommodity flow* or simply *multipath flow*. Instead of having a single flow function f , we have a flow function f_i for each commodity i from a set C of commodities. The constraints (1), (2), and (3) become

$$\forall i \in C, \forall e \in E, \sum_i f_i(e) \leq c_1(e) \quad (5)$$

$$\forall i \in C \quad \forall v \in V \setminus S, \quad \sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) = 0 \quad (6)$$

$$\begin{aligned} \forall i \in C, \quad &\sum_{v \in S^+} \left(\sum_{e \in \delta^-(v)} f_i(e) - \sum_{e \in \delta^+(v)} f_i(e) \right) \\ &= \sum_{v \in S^-} \left(\sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) \right). \end{aligned} \quad (7)$$

Maximum flow and minimum cost flow problems may also be posed under a multipath setup; we omit the details. Our review of network flows only touches aspects pertinent to this paper; for a thorough coverage on the subject of network flows, see [58] and [59] and the references therein. Note that the multipath model

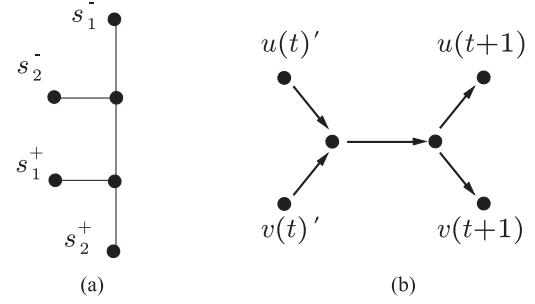


Fig. 4. (a) Simple G . (b) *Merge-split* gadget for splitting an undirected edge through time steps, for enforcing the head-on collision constraint.

stated here is sometimes also referred to as *integer multiflow* because f_i must have integer values.

III. FROM MULTIROBOT PATH PLANNING TO MULTIFLOW

A close algorithmic connection exists between optimal MPP and network flow problems. Maximum (single commodity) flow problems generally admit efficient (low-degree polynomial time) algorithmic solutions [59], whereas maximum multiflow is a well-known NP-hard problem, difficult to even approximate [60]. Mirroring the disparity between single- and multicommodity flows, in the domain of MPP problems, if there is a single group of interchangeable robots (here, for a group, it does not matter which robot goes to which goal as long as all goal locations assigned to the group are occupied by robots from the same group), then many optimal formulations admit polynomial time algorithms [17]. However, as soon as a single group of robots splits into two or more groups, finding optimal paths for these robots become intractable [4]. The apparent similarity between optimal MPP and multiflow is perhaps best explained through a graph-based reduction from MPP problems to network flow problems. The reduction will also form the basis of our algorithmic solution.

To describe the reduction, we use as an example the undirected graph G in Fig. 4(a), with start vertices $\{s_i^+\}, i = 1, 2$ and goal vertices $\{s_i^-\}, i = 1, 2$. An instance of MPP is given by $(G, \{r_1, r_2\}, x_I : r_i \mapsto s_i^+, x_G : r_i \mapsto s_i^-)$. We will reduce the problem to a network flow problem $\mathcal{N} = (G', c_1, c_2, S)$. The reduction proceeds by constructing a network that is a *time-expanded* version of the graph G , which allows the explicit consideration of the interactions among the robots over space and time. Time expansion derived in this paper can be viewed as performing *network flow over time* [61]. To carry out this expansion, a time horizon must first be determined. For different optimality objectives, the expansion time horizon, some natural number T , is generally different; for now we assume that T is fixed.

To begin building the network, we create $2T + 1$ copies of G 's vertices, with indices $0, 1, 1', \dots$, as shown in Fig. 5. For each vertex $v \in G$, denote these copies $v(0) = v(0)', v(1), v(1)', v(2), \dots, v(T)'$. For each edge $\{u, v\} \in G$ and time steps $t, t+1, 0 \leq t < T$, the *merge-split* gadget shown in Fig. 4(b) is added between $u(t)', v(t)',$ and $u(t+1), v(t+1)'$

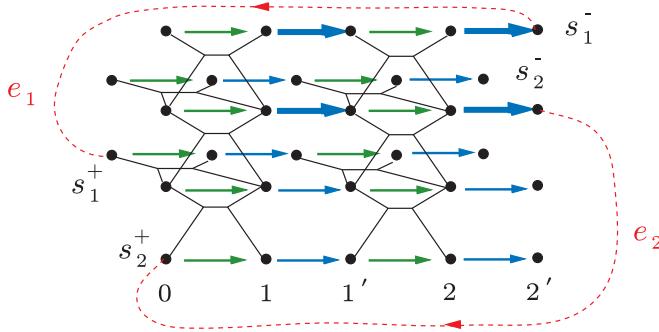


Fig. 5. Time-expanded network with an expansion time horizon of $T = 2$ over the base graph Fig. 4(a).

(arrows from the gadget are omitted from Fig. 5 since they are small). For the gadget, we assign unit capacity to all arcs, unit cost to the horizontal middle arcs, and zero cost to the other four arcs. The merge-split gadget ensures that two robots cannot travel in opposite directions on an edge of the underlying graph in the same time step, which prevents head-on collision between two robots. To finish the construction of Fig. 5, for each vertex $v \in G$, we add one arc between every two successive copies (i.e., we add the arcs $(v(0), v(1)), (v(1), v(1)'), \dots, (v(T), v(T)')$). These correspond to the green and blue arcs in Fig. 5. For all green arcs, we assign them unit capacity and cost; for all blue arcs, we assign them unit capacity and zero cost. The green arcs allow robots to stay at a vertex during a time step, whereas blue arcs ensure that each vertex holds at most one robot, enforcing the meet collision constraint.

The time-expanded network in Fig. 5 (with the exception of arcs e_1 and e_2 , which will become relevant shortly) is the desired G' . For the set $S = S^+ \cup S^-$, we may simply let $S^+ = \{v(0) : v \in \{s_i^+\}\}$ and $S^- = \{v(T)' : v \in \{s_i^-\}\}$. $\mathcal{N} = (G', c_1, c_2, S)$ is now complete; we have reduced MPP to an integer multiflow problem on \mathcal{N} , with each robot from R as a single type of commodity.

Theorem 1: Let (G, R, x_I, x_G) be an MPP instance. There is a bijection between its solution set (with a maximum number of time steps up to T) and the integer maximum multiflow solutions of flow value n on the time-expanded network \mathcal{N} constructed from (G, R, x_I, x_G) with T time steps.

Proof: Injectivity. Assume that $P = \{p_1, \dots, p_n\}$ (n is the number of robots) is a solution to an MPP instance. For each p_i and every time step $t = 0, \dots, T$, we mark the copy of $p_i(t)$ and $p_i(t)'$ (recall that $p_i(t)$ corresponds to a vertex of G) at time step t in the time-expanded graph G' . Connecting these vertices of G' sequentially (there is a unique way to do this) yields one unit of flow f_i on \mathcal{N} (after connecting to appropriate source and sink vertices in S^+, S^- , which is trivial). It is straightforward to see that if two paths p_i, p_j are not in collision, then the corresponding flows f_i and f_j on \mathcal{N} are vertex disjoint paths and therefore do not violate any flow constraint. Since any two paths in P are not in collision, the corresponding set of flows $\{f_1, \dots, f_n\}$ is feasible and maximal on \mathcal{N} .

Surjectivity: Assume that $\{f_1, \dots, f_n\}$ is an integer maximum multiflow on the network \mathcal{N} , which implies that $|f_i| = 1$ for all i 's. First, we establish that any pair of flows f_i and f_j are vertex disjoint. To see this, we note that f_i and f_j (both are unit flows) cannot share the same source or sink vertices due to the unit capacity structure of \mathcal{N} enforced by the blue arcs. If f_i and f_j share some nonsink vertex v at time step $t > 0$, both flows then must pass through the same blue arc [see Fig. 4(b)] with v being either the head or the tail vertex, which is not possible. Thus, f_i and f_j are vertex disjoint on \mathcal{N} . We can readily convert each flow f_i to a corresponding path p_i (after deleting extra source vertex, sink vertices, vertices in the middle of the gadgets, and tail vertices of blue arcs) with the guarantee that no p_i, p_j will collide due to a meet collision. By the construction of \mathcal{N} , the gadget we used ensures that a head-on collision is also impossible. The set $\{p_1, \dots, p_n\}$ is then a solution to the MPP defined by (G, R, x_I, x_G) . ■

Remark: A multiflow problem with unit individual flows can also be viewed as a type of multipath planning problem, known as the *edge disjoint path* problem [62]. △

IV. COMPLETE INTEGER LINEAR PROGRAMMING-BASED ALGORITHMS FOR OPTIMAL MULTIROBOT PATH PLANNING ON GRAPHS PROBLEMS

Because optimizing MPP solutions over Objectives 1–4 are computationally intractable, reducing MPP to multiflow problems does not make these optimal MPP problems any easier. However, with a network flow formulation (see Section II-C), it becomes possible to establish ILP models for optimal MPP formulations. These ILP models can then be solved with powerful linear programming packages. In comparison to A*-based algorithms augmented with heuristics,⁵ which often target an important but limited set of problem structures, ILP-based algorithms proposed here are *agnostic* to specific problem structures. As such, ILP-based algorithms appear more capable of addressing a wider range of MPP problems and in particular difficult MPP instances in which the robot-vertex ratio is high. In this section, we build ILP models for each of Objectives 1–4, assuming a fixed time span T . That is, these models only optimize the given objective for a specific T . The discussion of the full algorithms and their completeness then follows.

A. Minimizing the Makespan

A minimum makespan solution to an MPP instance $I = (G, R, x_I, x_G)$ can be computed using a *maximum multiflow* formulation. Fixing a time span T , let $\mathcal{N} = (G', c_1, c_2, S)$ be the time-expanded network for I , a set of n loopback arcs are added to G' by connecting each pair of corresponding start and goal vertices in S , from the goal to the start. We use e_j 's to denote arcs of G' , and let the n loopback arcs take the first n indices, with $e_j, 1 \leq j \leq n$, being the arc connecting the goal vertex of r_j to the start vertex of r_j . For example, for the G' in

⁵Here, the term heuristics does not refer to the admissible heuristic used by A* itself. Instead, it refers to ways of breaking the search space of a multirobot path planning problem into disjoint, smaller subspaces.

Fig. 5, e_1 and e_2 are the loopback arcs for r_1 and r_2 , respectively. The loopback arcs have unit capacities and zero costs. Next, for each arc $e_j \in G'$ (including the loopback arcs), create n binary variables $x_{1,j}, \dots, x_{n,j}$ corresponding to the flow through that arc, one for each robot $1 \leq i \leq n$. That is, $x_{i,j} = 1$ if and only if robot r_i passes through e_j in G' . The variables $x_{i,j}$'s must satisfy two arc capacity constraints and one flow conservation constraint

$$\forall e_j, \sum_{i=1}^n x_{i,j} \leq 1 \quad (8)$$

$$\forall 1 \leq i, j \leq n, i \neq j, x_{i,j} = 0$$

$$\forall v \in G' \text{ and } 1 \leq i \leq n, \sum_{e_j \in \delta^+(v)} x_{i,j} = \sum_{e_j \in \delta^-(v)} x_{i,j}. \quad (9)$$

The objective function is

$$\max \sum_{1 \leq i \leq n} x_{i,i}. \quad (10)$$

B. Minimizing the Maximum Single-Robot Traveled Distance

For minimizing the maximum distance traveled by any robot, the network and variable creation remains the same as the minimum makespan setup; constraints (8) and (9) remain unchanged. Because we want to send all robots to their goals, a maximum flow may be forced through the constraint

$$\forall 1 \leq i \leq n, x_{i,i} = 1. \quad (11)$$

To encode the min–max objective function, we introduce an additional integer variable x_{\max} and add the constraint

$$\forall 1 \leq i \leq n, \sum_{e_j \in G', j > n} c_2(e_j) \cdot x_{i,j} \leq x_{\max}. \quad (12)$$

For a fixed i , the left side of (12) represents the distance traveled by robot r_i . Note that $x_{i,j} = 0$ for $j < n$. The objective function is then simply

$$\min x_{\max}. \quad (13)$$

C. Minimizing the Total Arrival Time

For minimizing the total arrival time, the network and variables from the minimum makespan ILP-model and constraints (8), (9), and (11) are inherited. To represent the objective function, for each time step $1 \leq t \leq T$ and each $v = x_G(r_i)$, $1 \leq i \leq n$, we create a binary variable y_i^t . Then, we give new indices to certain existing variables. Recall that for each arc $e_j = (v(t), v(t')) \in G'$ (e.g., the four extra bold blue arcs in Fig. 5), a variable $x_{i,j}$ is created. There are nT such variables. Here, we give these variables a second index x_i^t . That is, x_i^t is the binary variable indicating whether arc $(v(t), v(t')) \in G'$, $v = x_G(r_i)$ is used by robot r_i .

Given a network with a fixed T , if constraints (8), (9), and (11) can be satisfied, then there is a feasible solution to the original MPP problem. In this case, $x_i^T = 1$ for $1 \leq i \leq n$. We let $y_i^T = x_i^T$. Then, each y_i^t , $1 \leq t < T$ is defined recursively over x_i^t and

Algorithm 1: MPP-ILP-OPTIMIZATION

Input: (G, R, x_I, x_G) and obj , the objective
Output: $P = \{p_1, \dots, p_n\}$, the optimal solution

- 1 $T_{\min} \leftarrow$ Initial time span (an underestimate, e.g., 0)
- 2 $T_{\max} \leftarrow$ Estimated time span containing optimal solution
- 3 $P \leftarrow \emptyset$
- 4 **for** T from T_{\min} to T_{\max} **do**
- 5 Build ILP model for obj with time span T
- 6 Attempt solving model using an optimizer
- 7 $P \leftarrow$ best solution found so far
- 8 **end**
- 9 **return** P

y_i^{t+1} as

$$y_i^t \geq y_i^{t+1} + x_i^t - 1, \quad y_i^t \leq y_i^{t+1}, \quad y_i^t \leq x_i^t. \quad (14)$$

Effectively, (14) performs a logical AND over x_i^t and y_i^{t+1} and stores the result in y_i^t . In the end, the smallest t for which $y_i^t = 1$ is the time robot r_i reaches its goal (and stops). Therefore, for each $1 \leq i \leq n$, $\sum_{t=1}^T y_i^t$ is the number of time steps from the time r_i arrives at its goal until time T . Thus, $T - \sum_{t=1}^T y_i^t$ is the time spent by r_i . To minimize the total arrival time, the objective function can be expressed as

$$\min \left(nT - \sum_{1 \leq i \leq n, 1 \leq t \leq T} y_i^t \right). \quad (15)$$

D. Minimizing the Total Distance

From the ILP model for minimizing the maximum distance, we need to change only the objective function for computing a minimum total distance solution. We do not need the variable x_{\max} and simply update the objective function to

$$\min \sum_{e_j \in G', j > n, 1 \leq i \leq n} c_2(e_j) \cdot x_{i,j}. \quad (16)$$

E. Algorithm Structure and Completeness

The general algorithm structure for optimally solving MPP is outlined in Algorithm 1. Here, we assume that the problem (G, R, x_I, x_G) is feasible, which can be readily checked [41]. Then, we make conservative estimates on the minimum time span T_{\min} and the maximum time span T_{\max} such that there must be a $T \in [T_{\min}, T_{\max}]$ corresponding to the time span for an optimal solution. The algorithm then simply searches through $[T_{\min}, T_{\max}]$ to locate T and the optimal solution. Denoting the specific algorithms for Objectives 1–4 as MINMAKESPAN, MINMAXDIST, MINTOTALTIME, and MINTOTALDIST, respectively, we now fill in how to estimate T_{\min} and T_{\max} for them.

While $T_{\min} = 0$ always works, we can do better by setting T_{\min} as the maximum over all robots the shortest path length for each robot, ignoring all other robots. This T_{\min} clearly applies to all four objectives. For MINMAKESPAN, we may set T_{\max} to be the time needed for a feasible solution, which can be computed with [41]. In the worst case, $T = O(|V|^3)$. Note that

this establishes the completeness of MINMAKESPAN and we may assume we have an optimal T_{opt} for MINMAKESPAN.

For MINMAXDIST, we set $T_{\text{max}} = nT_{\text{opt}}$. This is true because at each time step, at least one robot must move a distance of one. After $nT_{\text{opt}} + 1$ time, by the pigeonhole principle, some robot must have moved $T_{\text{opt}} + 1$ steps. However, since T_{opt} is optimal for MINMAKESPAN, there exists a solution in which no robots travel more than T_{opt} . This yields a contradiction.

The same $T_{\text{max}} = nT_{\text{opt}}$ works for MINTOTALTIME for a different reason. After $nT_{\text{opt}} + 1$ time steps, at least one robot must have spent this much time. On the other hand, the total time from the optimal MINMAKESPAN solution is no more than nT_{opt} . So the optimal solution for MINTOTALTIME will not require a time span of more than nT_{opt} . For MINTOTALDIST, $T_{\text{max}} = nT_{\text{opt}}$ also works. After $nT_{\text{opt}} + 1$ time steps, the total distance traveled by all robots must exceed nT_{opt} . However, the optimal solution for MINTOTALTIME will not require more than nT_{opt} total number of moves. We have shown that the four algorithms are all complete.

Proposition 2: MINMAKESPAN, MINMAXDIST, MINTOTALTIME, and MINTOTALDIST are all complete.

V. HEURISTICS FOR EFFECTIVE COMPUTATION OF NEAR-OPTIMAL SOLUTIONS

We have established that ILP-based algorithms are complete and always produce optimal solutions. Their performance in handling relatively small but highly constrained problems are in fact quite good (see Section VI). As problem sizes grow (i.e., as the graph G and the number of robots n increase), however, the computation time grows rapidly. From a practical point of view, it may be far more desirable to quickly compute a good quality but suboptimal solution than to wait longer for the optimal solution. In this section, we introduce several heuristics to accomplish this goal, with a particular focus on computing solutions with the minimum makespan.

A. Building More Compact Models

To extract the best performance out of a solver, it is beneficial to have a lean model (i.e., fewest columns and rows). So far, our focus has been to provide a general multiflow-based framework so that the ILP models can be easily built. During the model translation stage, the models can be further simplified. The heuristics discussed in this section aim at making the representation of constraints (8) and (9) more compact. As such, they apply to all objectives.

Better encoding of the collision constraints: In building the network flow model (e.g., Fig. 5), we used a merge-split gadget [see Fig. 4(b)] for enforcing the head-on collision constraint and extra time steps (e.g., the blue arcs in Fig. 5) for avoiding meet collisions. When we translate them into linear constraints, these structures can be simplified to yield the more compact structure illustrated in Fig. 6.

In the newer structure, each merge-split gadget now has two arcs instead of five. Also, the blue edges are gone. The updated gadget for an edge $\{u, v\} \in E$ between time steps t and $t + 1$ is shown in Fig. 7 (note that due to the removal of the blue arcs,

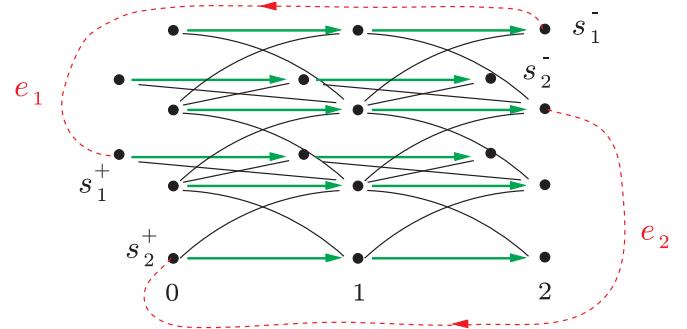


Fig. 6. More compact representation of the network flow graph from Fig. 5.

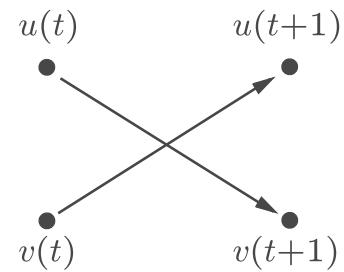


Fig. 7. Simplified merge-split gadget for enforcing the head-on collision constraint.

vertices such as $v(t)'$ are no longer needed). That is, instead of five, only two variables are needed for each robot. Denoting these binary variables as $x_{i,(u(t),v(t+1))}$ and $x_{i,(v(t),u(t+1))}$ for a robot r_i , the head-on collision constraint for a single gadget can be readily encoded as

$$\sum_{i=1}^n x_{i,(u(t),v(t+1))} + \sum_{i=1}^n x_{i,(v(t),u(t+1))} \leq 1. \quad (17)$$

Then, to enforce the meet collision constraint, for example, at a vertex $v(t)$, we simply require that at most one outgoing arc from $v(t)$ may be used, i.e.,

$$\sum_{e_j \in \delta^-(v(t)), 1 \leq j \leq n} x_{i,j} \leq 1. \quad (18)$$

The new ILP model is roughly half of the size of the original model.

Reachability analysis: In the time-expanded graph, there are redundant binary (arc) variables that can never be true because some arcs are never reachable. For example, in Fig. 6, at $t = 0$, the only outgoing arcs that can possibly be used are those originating from s_1^+ and s_2^+ . The rest can be safely removed. In general, for each robot r_i , based on its reachability from its start vertex and to its goal vertex, a sizable number of binary variables $x_{i,j}$'s can be deleted.

B. Divide-and-Conquer Over the Time Domain

In evaluating the ILP model-based algorithm for optimal makespan computation, we observe that the ILP solver running time appears to grow exponentially as the size of the model

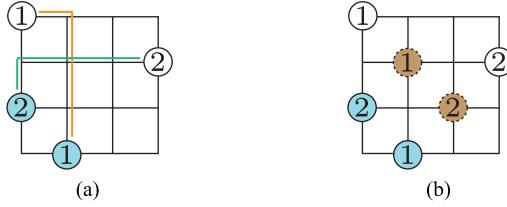


Fig. 8. (a) Simple two-robot problem. (b) Time-divided instances.

grows. This prevents the algorithm from performing well over instances with more than a few tens of robots. The observation, while hampering the effectiveness of the exact algorithm, turns out to offer a useful insight toward a highly effective heuristic. We find that when robot-vertex ratio is not critically high (i.e., not approaching one), relatively small ILP models do not present much challenge for ILP solvers. This is true even when there are a large number of robots (i.e., in the hundreds). To apply the ILP model-based method to more challenging problems (e.g., solving problems with hundreds of robots quickly), we simply limit the size of the individual ILP model fed to the solver. One way to achieve this is through *divide-and-conquer over the time domain*. We use a simple example (see Fig. 8) to illustrate the idea.

In Fig. 8(a), we have a planning problem for two robots on a 3×3 grid. To execute the heuristic, we first compute a shortest path for every pair of start and goal locations. In this case, we get the orange and green paths for robots 1 and 2, respectively. Then, if we decide to split the problem into two smaller problems, for each of the paths, it is split into two equal or nearly equal length pieces, and the middle node is set as the intermediate goal. In our example, we may do this for robot 1 by setting the intermediate goal location at (1,1) from the top-left corner [the brown disc labeled 1 in Fig. 8(b)]. For robot 2, because the middle location coincides with that of robot 1, we pick an alternative unoccupied location as the intermediate goal for robot 2, in this case (2,2) from the top-left corner. The intermediate goals for the first instance will also serve as the start locations of the second instance. This yields two child instances both requiring a time expansion with two steps each, effectively making the individual ILP model roughly half the size of the original one that required a time expansion with four steps. In general, we may divide a problem into arbitrarily many smaller instances in the time domain.

If a problem is divided into k subproblems, we call the resulting heuristic a *k -way split*. Because the division is over time, there is no interaction between the divided instances. Once we obtain a solution for each child instance, the solutions can be glued together by concatenating the results. In practice, this heuristic dramatically improves algorithm performance without significant negative impact on path makespans; we observe a consistent speedup in computational experiments.

Remark: The k -way split heuristic, by design, is particularly apposite for the makespan objective, owing to the additive nature of the makespan objective over the split subproblems. Beside the makespan, the heuristic also applies to distance objectives

(i.e., Objectives 2 and 4) quite well, as long as the time horizon required for finding distance optimal solution does not differ greatly from the time horizon required for minimum makespan solution. The heuristic does not directly apply to Objective 3 because total time is not additive over the split subproblems. As an example, suppose that a two-way split is carried out with each subproblem having a time horizon of $T/2$. If a robot r_i does not move in the solution to the first subproblem (i.e., $0 \leq t \leq T/2$), it contributes 0 to the total distance. However, if r_i moves even a single step in the solution to the second subproblem (i.e., $T/2 \leq t \leq T$), then r_i will contribute at least $T/2$ to the total arrival time. Nevertheless, the k -way split is still helpful in this case as we may use it to quickly compute an initial T for performing the time expansion. \triangle

VI. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our optimal and near-optimal MPP algorithms with an emphasis on MIN-MAKESPAN. Our performance evaluation covers a broad spectrum of typical problem settings. For each setting, we push the limit on the robot-vertex ratio to as high as 100%. To the best of our knowledge, the majority of the settings with high robot-vertex ratio have never been attempted with much success prior to our study.⁶

After examining a large number of existing approaches including OD+ID, ID, WHCA*, ICTS, CBS, and COOPT, we focus our comparison on ID-based anytime algorithm and COOPT due to problem similarity and our emphasis on 100% success rate.⁷ We point out that there are various differences between our robotics-based MPP formulation and these methods to which we compare. OD+ID and ID support cycles, whereas CBS and WHCA* appear to be designed for cycle-free MPP, as they could not solve any 9-puzzle. The problem definition for COOPT suggests it solves MPP, but it employs a cycle-free subroutine for finding feasible solutions. Except for COOPT, most of these methods are designed for optimizing total time and total distance optimal objectives, and do not naturally extend to the makespan. However, the associated makespans produced by these algorithms are usually of good quality. Among these, our experiments show that ID-based anytime algorithm is the most versatile due to its IDA*-like incremental structure. On the other hand, OD+ID, ICTS, CBS, and WHCA* do not scale well when the robot-vertex ratio goes beyond 10%. COOPT is designed for makespan.

We implemented all algorithms (MINMAKESPAN, MIN-MAXDIST, MINTOTALTIME, and MINTOTALDIST) in the Java programming language. We take advantage of multicore CPUs when the k -way split heuristic is being used. Also, Gurobi [64], the ILP solver used in our implementation, can engage multiple

⁶For completeness, 15-puzzle and 24-puzzle have been successfully attempted [63]. Beside, differences in not allowing cyclic rotation, the solutions also do not support multiple vacant cells.

⁷Some of these algorithms were evaluated without requiring that all robots reach their goals. For example, in the WHCA* work [8], if an instance with n robots is solved for $p < n$ robots, the problem is counted as partially solved. We require each instance to be fully solved to be counted as a success, which is a stringent requirement for robot path planning.

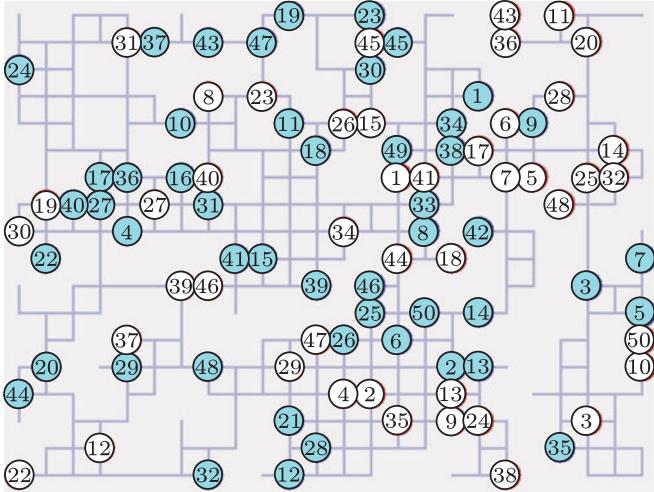


Fig. 9. Typical 24×18 grid instance with 25% vertices removed to simulate obstacles and 50 start and goal locations. Note that the connectivity of the graph is low in some areas. For example, the lower right corner blob is only singly-connected to the rest of the graph. Some (blue shaded) start locations may overlap with some (not shaded) goal locations.

cores automatically for hard problems. We ran all our tests on a MacBook Pro laptop computer (Intel Core i7-4850HQ, 16 GB memory). We obtained the C code implementing OD + ID, ID, and WHCA* among others from Trevor Standley. We modified (the original code supports only 32×32 grids) and compiled the code as a 64-bit windows executable under MSVC 2010 with all speed optimization flags turned on. C# code for CBS was retrieved from the authors' online repository.⁸ The comparison to COBOPT uses the result provided in [47], which covers only 8×8 and 16×16 grids.

A. Performance of MINMAKESPAN and k -Way Split Heuristic

We begin our experimental evaluation focusing on MINMAKESPAN and the k -way split heuristic. For this purpose, we use as the base graph a 24×18 grid with varying number of vertices (0–25%) removed to simulate obstacles. The connectivity of the graph is always maintained. We note that with 25% vertices removed, the graph is already sparsely connected at some places (see e.g., Fig. 9), making solving these problems optimally a challenging task. In presenting the computational results, each data point in the figures is an average over ten sequentially, randomly generated instances. For each obstacle percentage, we start from the lowest number of robots (usually 10 or 20) and allocate a maximum of 600 s for each problem instance. If an instance takes more than 600 s to produce a result, the instance is stopped and we move to the next obstacle percentage. This also means that a data point is given only if each of the ten instances is completed within 600 s. Over the same set of problem instances, the MINMAKESPAN algorithm is executed in the exact manner (which produces optimal makespan solutions) and with the k -way split heuristic.

⁸<https://bitbucket.org/eli.boyarski/>.

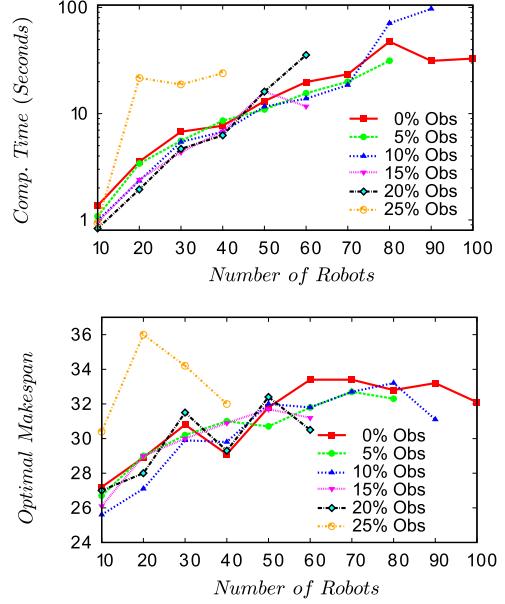


Fig. 10. [top] Average computation time of the exact MINMAKESPAN algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The (average) optimal makespan.

The exact makespan computation result is summarized in Fig. 10. For all obstacle settings, the MINMAKESPAN algorithm computes optimal makespan solutions consistently for up to 100 robots with an average computation time of no more than 100 s. Notably, for 50 robots and obstacles up to 20%, the MINMAKESPAN algorithm is able to complete in about 15 s in all cases. From the top plot of Fig. 10, we observe that for each fixed obstacle percentage, the computation time appears to grow exponentially with respect to the number of robots. The computational difficulty of a particular problem instance also depends heavily on the actual optimal makespan. For example, a problem instance in the case of 25% and 20 robots has a particularly long makespan (see the bottom plot of Fig. 10), resulting in a large jump of the computation time. Examining the instance reveals that a narrow corridor-like setting is present, similar to the scenario from Fig. 9, which requires two robots to pass through the corridor from opposing directions. This induces a much larger T in the time expansion step, causing a significant increase in computation time.

The k -way split heuristic brings a significant performance boost, allowing a much higher robot-vertex ratio in general. In our tests, we are often able to double or even triple the supported robot density. For the 24×18 grid, we evaluated the k -way split heuristic for k up to 16. The four-way split performance is illustrated in Fig. 11. In the figure, we measure optimality using a conservatively estimated *optimality ratio*. To obtain this, we divide the objective value returned by the optimizer over a conservative estimate (a lower bound). For makespan, this lower bound estimate is obtained by first computing the shortest path for each robot ignoring all other robots. The minimum makespan estimate is obtained by taking the maximum length over all these shortest paths. Clearly, the optimality ratio obtained this way is an overestimate.

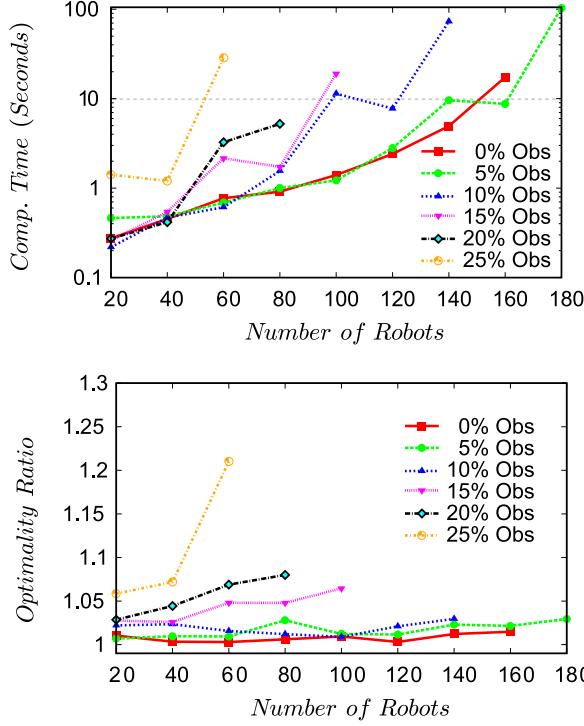


Fig. 11. [top] Average computation time of the MINMAKESPAN algorithm (with four-way split heuristic) over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

We make three comments over Fig. 11. First, from the top plot, we observe that our method is highly effective in terms of computation time, capable of computing minimum makespan solutions for up to 180 robots, which translates to a maximum robot-vertex ratio of 44%. The majority of the cases are solved within 10 s. Even when there are 25% obstacles, we could solve the problem consistently for up to 60 robots in about 40 s. Second, we again observe an exponential relationship between computation time and the number of robots. Third, all computed solutions are very close to being optimal, with all but one case having an optimality ratio of below 1.1. The average minimum makespan for these instances is about 35.

The rest of the k -way split evaluation is presented in Fig. 12, in which the computation time and optimality ratio are shown side by side. To improve clarity, axis labels are omitted. The omitted keys are the same as those from Fig. 10, representing different obstacle percentages. In conjunction with Figs. 10 and 11, as k increases, we observe a general trend of reduced computation time at the expense of loss of optimality. With 16-split, MINMAKESPAN can solve problems with 300 robots, corresponding to a robot-vertex ratio of 69%.

For comparison, we ran ID-based anytime algorithm over the same set of instances with a 600-s time limit (we also attempted OD + ID, CBS, and WHCA*, which are unable to consistently go past 40 robots under the same setup; we used 21×21 grid for CBS). The result is plotted in Fig. 13. ID actually performs quite well for up to 60 robots, which can be attributed to its A* root with minimum overhead as compared to our method. However,

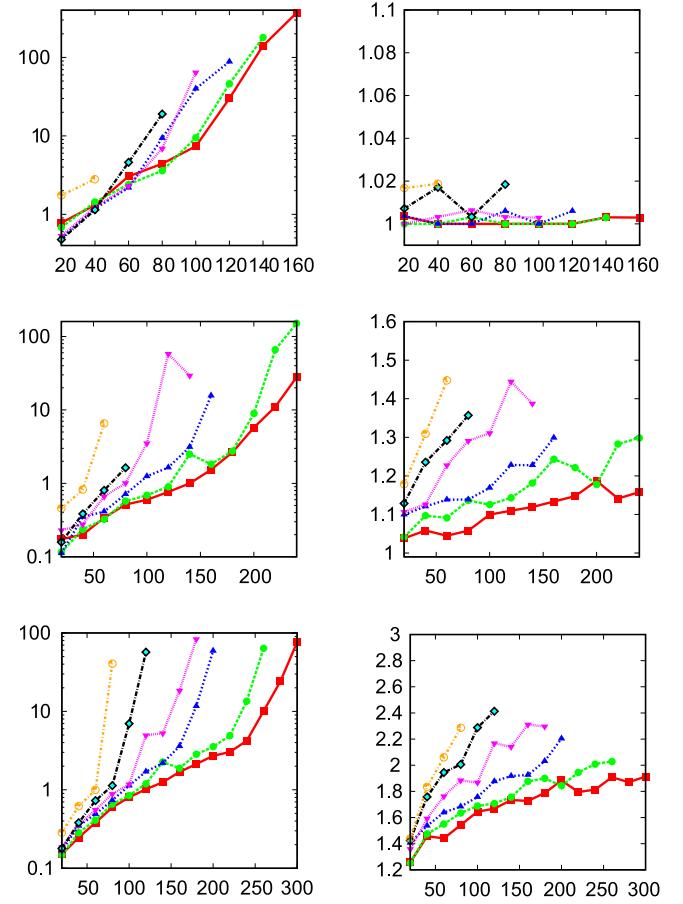


Fig. 12. Performance of the MINMAKESPAN algorithm with k -way split for $k = 2$ (top row), 8 (middle row), and 16 (bottom row). The computation time and optimality ratio plots for each k are shown side by side. The x -axis for all plots represents the number of robots. The y -axis for the plots on the left represents the computation time in seconds. The y -axis for the plots on the right represents the optimality ratio. The keys for all plots are the obstacle percentage, identical to that of Fig. 10.

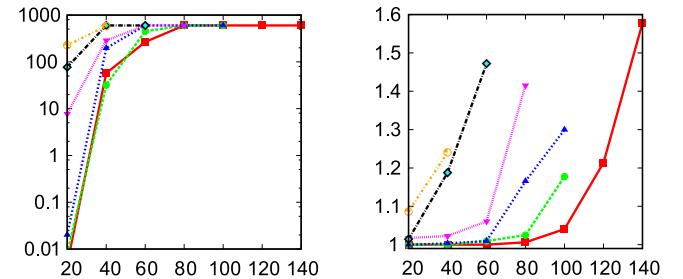


Fig. 13. Performance of the ID-based anytime algorithm over the same set of problem instances. The plot setup is the same as that from Fig. 12.

the performance of ID degrades faster—it does not scale well beyond 100 robots in our tests. MINMAKESPAN, with two-way split, readily outperforms ID when there are 40 or more robots. Conceivably, it may be possible to combine ID and k -way split to make it run faster. However, adding k -way split to ID will inevitably make the overall makespan more suboptimal.

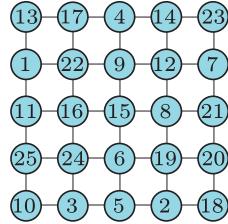


Fig. 14. Instance of a 25-puzzle problem solved by MINMAKESPAN.

B. Minimum Makespan Solution to N^2 -Puzzles

Next, we evaluate the efficiency of the MINMAKESPAN algorithm for finding minimum makespan solutions to the N^2 -puzzle for $n = 3, 4, 5$, and 6 . These problems have a robot-vertex ratio of 100%, making them highly constrained. Note that an N^2 -puzzle instance is always solvable for $n \geq 3$ [41]; this means that all states are connected in the state (search) space. We ran the MINMAKESPAN algorithm on 100 randomly generated N^2 -puzzle instances for $n = 3, 4, 5$. For the 9-puzzle, computation on all instances completed successfully with an average computation time of 0.46 seconds per instance. To compare the computational result, we implemented a (optimal) BFS algorithm. The BFS algorithm is heavily optimized. For example, cycles are precomputed and hard coded to save computation time. Since the state space of the 9-puzzle is small, the BFS algorithm is capable of optimally solving the same set of 9-puzzle instances with an average computation time of about 1.08 s per instance.

Once we move to the 16-puzzle, the power of general ILP solvers becomes evident. MINMAKESPAN solved all 100 randomly generated 16-puzzle instances with an average computation time of 4.2 s. On the other hand, the BFS algorithm with a priority queue that worked for the 9-puzzle ran out of memory after a few minutes. As our result shows that an optimal solution for the 16-puzzle generally requires 6 time steps, it seems natural to also try bidirectional search, which cuts down the total number of states stored in memory. To complete such a search, one side of the bidirectional search generally must reach a depth of 3, which requires storing over 5×10^8 states (the branching factor is over 1000), each taking 64 bits of memory. This translates into over 4 GB of raw memory and over 8 GB of working memory, which is more than the JavaVM can handle: A bidirectional search ran out of memory after about 10 min in general. We also experimented with C++ implementation using STL libraries, which yields a similar result (i.e., ran out of memory before reaching a search depth of 3).

For the 25-puzzle, without a good heuristic, bidirectional search cannot explore even a tiny fraction of the fully connected state space with about 10^{25} states. On the other hand, MINMAKESPAN again consistently solves the 25-puzzle, with an average computational time of 391.6 s over 100 randomly created problems. Fig. 14 shows one of the solved instances with a seven-step solution given in Fig. 15. Note that seven steps are the least possible as it takes at least seven steps to move robot 10 to its desired goal. We also tested MINMAKESPAN on the

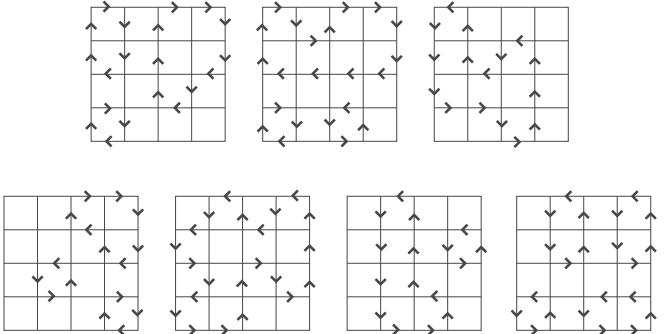


Fig. 15. Optimal seven-step solution (from left to right, then top to bottom) to the 25-puzzle problem from Fig. 14, by MINMAKESPAN in about 15 min.

36-puzzle. While we had some success here, MINMAKESPAN generally does not seem to solve a randomly generated instance of the 36-puzzle within 24 h, which has 3.7×10^{41} states and a branching factor of well over 10^6 .

As a comparison, CBS and WHCA* cannot solve the 9-puzzle. OD + ID and ID can solve only the 9-puzzle but could not solve any 16-puzzles in 600 s.

C. Minimum Makespan on 8×8 , 16×16 , and 32×32 Grids

In this section, we evaluate MINMAKESPAN with underlying graphs that are 8×8 grids, 16×16 grids, and 32×32 grids with 20% obstacles. In addition to further demonstrating the effectiveness of MINMAKESPAN, this allows us to better compare our results. 8×8 and 16×16 grids are used as the environment for evaluation in [47]. 32×32 grids with 20% obstacles are used for evaluation in [2] and [8].

For 8×8 and 16×16 grids, the instances are constructed using the same procedure stated in Section VI-A. Again, each data point is an average over ten sequentially randomly created instances. Given the size of 8×8 and 16×16 grids, we limit k to 8; using 16-way split can solve more instances but incurs an average solution optimality between 2 and 4. Each instance is given a time limit of 600 s. The outcome of these experiments is plotted in Fig. 16, along with the result from running ID, CBS, OD + ID, and WHCA*. MINMAKESPAN can solve problems with 50 robots to almost true optimal solutions in just 10 s. With the four-way split, MINMAKESPAN can further push to 60 robots (robot-vertex ratio of 94%) with solutions that are within 1.7-optimal. ID can only handle up to 30 robots. As reported in [47], COOPT generally takes more than half an hour to produce its final solution when there are 24 or more robots.⁹ The solution quality also degrades quickly as the number of robots increases. For example (Fig. 2 and Fig. 3 in [47]), at 50 robots, COOPT takes over an

We observe that, over the 8×8 grid, with the two-way split heuristic, MINMAKESPAN can solve problems with 50 robots to almost true optimal solutions in just 10 s. With the four-way split, MINMAKESPAN can further push to 60 robots (robot-vertex ratio of 94%) with solutions that are within 1.7-optimal. ID can only handle up to 30 robots. As reported in [47], COOPT generally takes more than half an hour to produce its final solution when there are 24 or more robots.⁹ The solution quality also degrades quickly as the number of robots increases. For example (Fig. 2 and Fig. 3 in [47]), at 50 robots, COOPT takes over an

⁹We did not directly run COOPT over our randomly created instances. However, the instances in [47] are created in an identical manner. Therefore, given that similarly powered computers are used, the computation time and solution makespan are directly comparable between ours and those from [47].

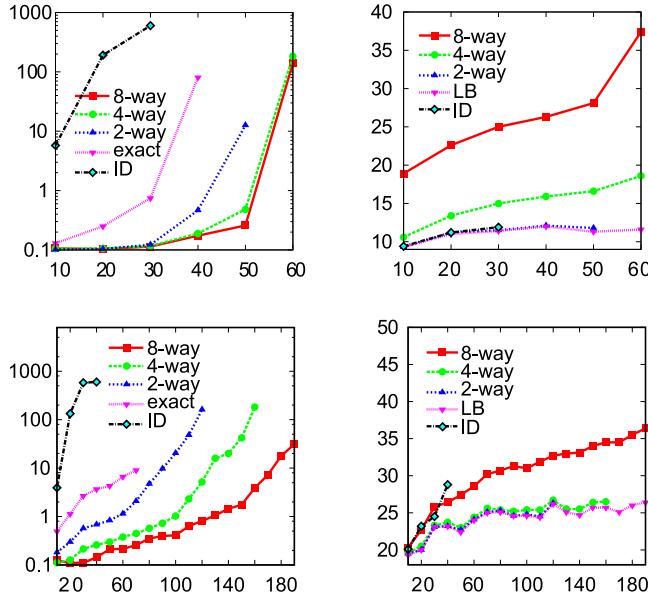


Fig. 16. Performance of the MINMAKESPAN algorithm with k -way split for $k = 2\text{--}8$ and the ID-based anytime algorithm. The computation time and solution makespan plots are shown side by side. The x -axis for all plots represents the number of robots. The y -axis for the plots on the left represents the computation time in seconds. The y -axis for the plots on the right represents the solution makespan. Note that we did not plot the makespan computed by the exact MINMAKESPAN algorithm. Instead, the lower bound (LB) estimate of makespan is plotted (in magenta color). [top] Result on the 8×8 grid. [bottom] Result on the 16×16 grid.

hour to produce a solution with a makespan of over 160, whereas our two-way split heuristic yields a near-optimal makespan of 11.8 in just 10 s.

Over the 16×16 grid, MINMAKESPAN is able to handle instances with 190 (robot-vertex ratio of 74%) robots with the eight-way split, while at the same time yielding solutions that are always less than 1.4-optimal. When switched to the four-way split, MINMAKESPAN can consistently solve problems with up to 160 robots to no worse than 1.03-optimal. In comparison, ID can solve instances with up to 80 robots to relatively good quality. Taking on average an hour of computation, COBOPT can handle up to 128 robots; the solution makespans are also highly suboptimal. For example, (see [47, Figs. 4 and 5]), for about 100 robots, the computed makespan by COBOPT is at 200, whereas the optimal makespan is about 25. Across 8×8 and 16×16 grids, we observe a speedup of over 100 times when MINMAKESPAN (with the four-way split heuristic) is compared with COBOPT. At the same time, our method yields solutions with much smaller makespan.

A classical test scenario is the 4-connected 32×32 grid with 20% vertices randomly removed.¹⁰ For completeness, we also

¹⁰Some work (e.g., [2]) also adopts an 8-connected model. That is, each vertex is on the grid is connected to its 8-neighborhood. This causes the unit cost to be assigned to all edges, although a diagonal edge should have length $\sqrt{2}$ times that of a nondiagonal edge. Since we are modeling robots in this work, we do not discuss the 8-connected model here. However, we mention that our algorithms easily extend to the 8-connected model. Our tests show that we can in fact compute near-optimal makespan for 400 robots on 32×32 grids assuming 8-connectivity.

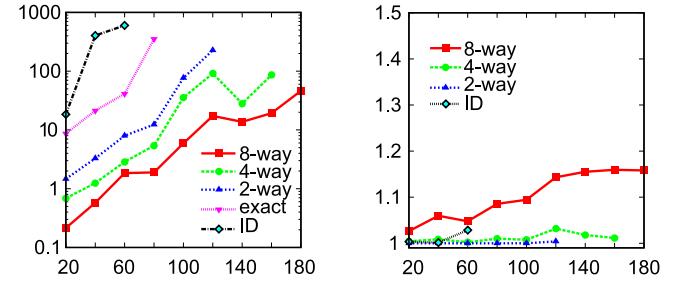


Fig. 17. Performance of MINMAKESPAN and the ID-based anytime algorithm over 32×32 grid. The axis setup is the same as that from Fig. 12.

perform an evaluation of this setup. We randomly generated the instance as before, ran the test, and plotted the result in Fig. 17. From the figure, we observe a pattern consistent with experiments on the 8×8 , 16×16 , and 24×18 grids.

D. Algorithm Performance Over All Objectives

Finally, we evaluate the performance of our algorithms at optimizing all objectives. The result on MINMAKESPAN is already presented in Section VI-A, which we also use as the solution to optimizing Objective 2 (i.e., we simply use MINMAKESPAN in place of MINMAXDIST due to its superior performance). Note that no change to the plots are needed here because, on one hand, we can use a (near)-optimal makespan solution as a solution to minimize the maximum single-robot traveled distance. This is true because the makespan of a solution is always no less than the minimum maximum single-robot distance. On the other hand, the lower bound estimate for the minimum makespan is the same as that for the minimum maximum single-robot distance.

Before moving to MINTOTALTIME and MINTOTALDIST, we note that these algorithms possess some properties of an *anytime algorithm*, which is of practical importance. In solving these ILP models, the solver generally uses variations of the *branch-and-bound* algorithm [65]. For computing total time (and distance) optimal solutions, a branch-and-bound algorithm always computes a feasible solution first and then iteratively improves over the feasible solution. This naturally leads to the steadily improving solution quality commonly observed in an anytime algorithm. The anytime property of MINTOTALTIME and MINTOTALDIST allows us to set a desired suboptimal threshold to reduce the computation time. Note that the same cannot be said for MINMAKESPAN because a feasible solution is also an optimal solution for the minimum makespan case.

Our next set of results focuses on the MINTOTALTIME algorithm (see Fig. 18). The general setup is the same as that used in Section VI-A. In particular, the same set of problem instances is used. In our experiment, we limit both time (600 s) and required suboptimality threshold (automatically adjusted) to achieve a balanced performance. The lower bound estimate for computing optimality ratio is obtained by summing over the individual shortest path lengths. The actual optimal total time is about 15 times the number of robots (i.e., 150 for 10 robots and 1500 for 100 robots), regardless of the percentage of obstacles. We observe that the MINTOTALTIME algorithm is fairly effective,

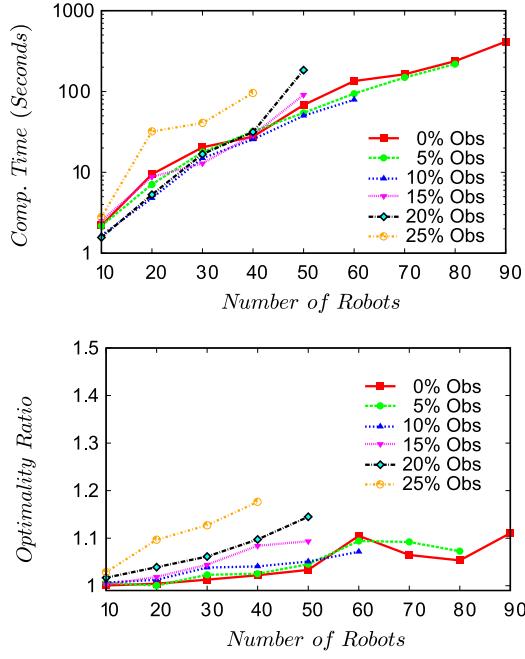


Fig. 18. [top] Average computation time of the MINTOTALTIME algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

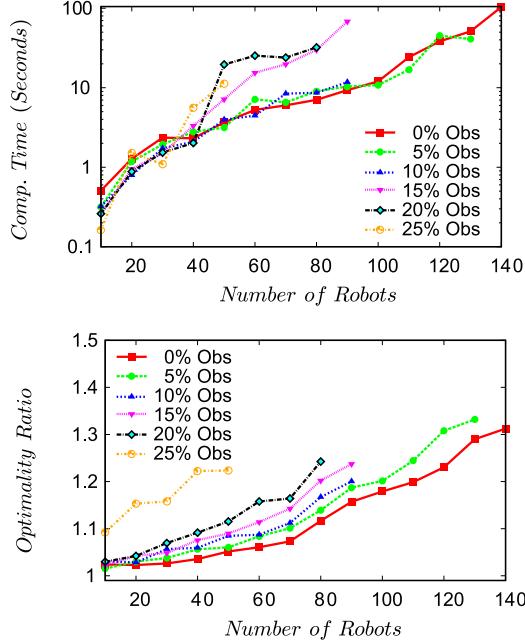


Fig. 19. [top] Average computation time of the MINTOTALDIST algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

capable of computing 1.1-optimal solutions for up to 100 robots in the allocated time.

Similar outcomes are also observed in the performance evaluation of the MINTOTALDIST algorithm with the four-way split heuristic (see Fig. 19). The optimal total distance is again about 15 times the number of robots. In comparison with the total time

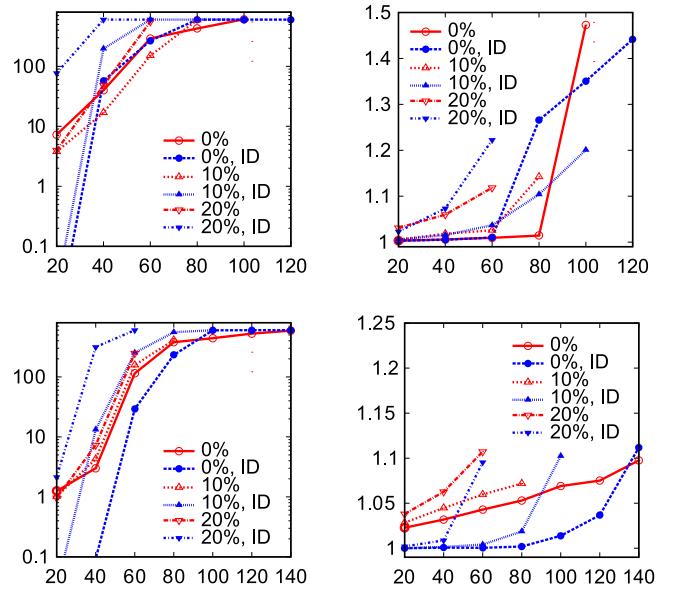


Fig. 20. Performance of MINTOTALTIME, MINTOTALDIST, and the ID-based anytime algorithm over 24×18 with 0–20% obstacles. Each instance is allowed 600 s of time. The axis setup is the same as that from Fig. 12. [top] MINTOTALTIME versus total time ID; the red lines correspond to data for MINTOTALTIME. [bottom] MINTOTALDIST versus total distance ID; the red lines correspond to data for MINTOTALDIST.

optimal case, due to the four-way split heuristic, the MINTOTALDIST algorithm is faster but produced solutions that are more suboptimal but still quite good.

For comparison, we run MINTOTALTIME, MINTOTALDIST, and ID (total time and total distance versions) on 24×18 grids with 0–20% obstacles with a maximum time limit of 600 s. The setting is slightly different from that used in obtaining Figs. 18 and 19; we do not set a suboptimal threshold here. The result is plotted in Fig. 20. In the case of total time (Objective 3), ID could solve more instances. For the instances that are solved, the achieved optimality is similar. For total distance (Objective 4), we observe similar outcomes. Here, ID could produce one more data point; the achieved optimality by both methods are again comparable (and good). Overall, MINTOTALTIME and MINTOTALDIST are competitive with ID.

VII. CONCLUSION

In this paper, we propose a general algorithmic framework for solving MPP problems optimally or near-optimally. Through an equivalence relationship between MPP and multiflow, we provide ILP model-based algorithms for minimizing the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. With additional heuristics, our algorithmic solutions are highly effective, capable of computing near-optimal solutions for hundreds of robots in seconds in scenarios with very high robot-vertex ratio. In pushing for high-performance algorithms aiming at solving MPP optimally or near-optimally, our eventual goal is to apply these algorithms to multirobot path planning problems in continuous domains. To this end, preliminary work has begun

to show promising results, producing algorithms that can compute solutions for around a hundred disc robots in bounded two-dimensional environments with holes [66].

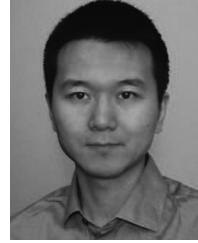
ACKNOWLEDGMENT

The authors would like to thank E. Boyarski and T. Standley for sharing their respective source codes. They also thank the reviewers for their help in improving the quality of the paper.

REFERENCES

- [1] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proc. AAAI Nat. Conf. Artif. Intell.*, 2010, pp. 173–178.
- [2] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 668–673.
- [3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–19, 2008.
- [4] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," ArXiv, 2015. [Online]. Available: <http://arxiv.org/pdf/1507.03289v1.pdf>
- [5] M. A. Erdmann and T. Lozano-Pérez, "On multiple moving objects," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1986, pp. 1419–1424.
- [6] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Autom.*, vol. 8, no. 6, pp. 707–717, Dec. 1992.
- [7] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. Robot. Autom.*, vol. 14, no. 6, pp. 912–925, Dec. 1998.
- [8] D. Silver, "Cooperative pathfinding," in *Proc. 1st Conf. Artif. Intell. Interact. Digit. Entertainment*, 2005, pp. 23–28.
- [9] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Trans. Robot.*, vol. 22, no. 4, pp. 650–665, Aug. 2006.
- [10] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *J. Artif. Intell. Res.*, vol. 31, pp. 497–542, 2008.
- [11] R. Jansen and N. Sturtevant, "A new approach to cooperative pathfinding," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2008, pp. 1401–1404.
- [12] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 3613–3619.
- [13] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 294–300.
- [14] J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 430–435.
- [15] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," presented at the *Robotics, Science Systems*, Seattle, WA, USA, 2009.
- [16] K. Solovey and D. Halperin, " k -color multi-robot motion planning," in *Proc. Workshop Algorithmic Found. Robot.*, 2012, pp. 191–207.
- [17] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X* (ser. *Springer Tracts in Advanced Robotics*), vol. 86. Berlin, Germany: Springer-Verlag, 2013, pp. 157–173.
- [18] M. Turpin, K. Mohita, N. Michael, and V. Kumar, "CAPT: Concurrent assignment and planning of trajectories for multiple robots," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 98–112, 2014.
- [19] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," presented at the *Robotics, Science Systems*, Rome, Italy, 2015.
- [20] D. Halperin, J.-C. Latombe, and R. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, nos. 3/4, pp. 577–601, 2000.
- [21] B. Nnaji, *Theory of Automatic Robot Assembly and Programming*. London, U.K.: Chapman & Hall, 1992.
- [22] S. Rodriguez and N. M. Amato, "Behavior-based evacuation planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 350–355.
- [23] T. Balch and R. C. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Trans. Robot. Autom.*, vol. 14, no. 6, pp. 926–939, Dec. 1998.
- [24] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 165–171.
- [25] B. Shucker, T. Murphey, and J. K. Bennett, "Switching rules for decentralized control with simple control laws," in *Proc. Am. Control Conf.*, Jul. 2007, pp. 1485–1492.
- [26] B. Smith, M. Egerstedt, and A. Howard, "Automatic generation of persistent formations for multi-agent networks under range constraints," *Mobile Netw. Appl. J.*, vol. 14, no. 3, pp. 322–335, Jun. 2009.
- [27] H. Tanner, G. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Trans. Robot. Autom.*, vol. 20, no. 3, pp. 443–455, Jun. 2004.
- [28] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Auton. Robots*, vol. 8, no. 3, pp. 325–344, Jun. 2000.
- [29] J. Ding, K. Chakrabarty, and R. B. Fair, "Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 12, pp. 1463–1468, Dec. 2001.
- [30] E. J. Griffith and S. Akella, "Coordinating multiple droplets in planar array digital microfluidic systems," *Int. J. Robot. Res.*, vol. 24, no. 11, pp. 933–949, 2005.
- [31] M. J. Matarić, M. Nilsson, and K. T. Simsarian, "Cooperative multi-robot box pushing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1995, pp. 556–561.
- [32] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1995, pp. 235–242.
- [33] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1997, pp. 193–200.
- [34] H. Choset *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA, USA: MIT Press, 2005.
- [35] J.-C. Latombe, *Robot Motion Planning*. Boston, MA, USA: Kluwer, 1991.
- [36] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006. [Online]. Available: <http://planning.cs.uiuc.edu/>
- [37] E. W. Story, "Note on the '15' puzzle," *Am. J. Math.*, vol. 2, pp. 399–404, 1879.
- [38] S. Loyd, *Mathematical Puzzles of Sam Loyd*. New York, NY, USA: Dover, 1959.
- [39] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *J. Combinatorial Theory, B*, vol. 16, pp. 86–96, 1974.
- [40] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proc. IEEE Symp. Found. Comput. Sci.*, 1984, pp. 241–250.
- [41] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning," presented at the *Workshop Algorithmic Foundations Robotics*, Istanbul, Turkey, 2014.
- [42] Q. Sajid, R. Luna, and K. E. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," presented at the *5th Symp. Combinatorial Search*, Niagara Falls, ON, Canada, 2012.
- [43] B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and rotate: A complete multi-agent pathfinding algorithm," *J. Artif. Intell. Res.*, vol. 51, pp. 443–492, 2014.
- [44] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3260–3267.
- [45] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, 2013.
- [46] E. Boyarski *et al.*, "ICBS: The improved conflict-based search algorithm for multi-agent pathfinding," presented at the *8th Annu. Symp. Combinatorial Search*, Ein Gedi, Israel, 2015.
- [47] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Proc. 12th Pacific Rim Int. Conf. Artif. Intell.*, 2012, pp. 564–576.
- [48] K. Kant and S. Zucker, "Towards efficient trajectory planning: The path velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.
- [49] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 1928–1935.
- [50] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 3475–3482.
- [51] A. F. van der Stappen, I. Karamouzas, and R. Geraerts, "Space-time group motion planning," in *Proc. 10th Workshop Algorithmic Found. Robot.*, 2012, pp. 227–243.

- [52] M. Peasgood, C. Clark, and J. McPhee, “A complete and scalable strategy for coordinating multiple robots within roadmaps,” *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 283–292, Apr. 2008.
- [53] K. Solovey, O. Salzman, and D. Halperin, “Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning,” in *Proc. Int. Workshop Algorithmic Found. Robot.*, 2014, pp. 591–607.
- [54] A. Krontiris, Q. Sajid, and K. Bekris, “Towards using discrete multiagent pathfinding to address continuous problems,” presented at the *Workshop Multi-Agent Pathfinding*, Toronto, ON, Canada, 2012.
- [55] J. Yu and S. M. LaValle, “Planning optimal paths for multiple robots on graphs,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 3612–3617.
- [56] J. Yu and S. M. LaValle, “Fast, near-optimal computation for multi-robot path planning on graphs,” in *Proc. 17th AAAI Conf. Late-Breaking Develop. Field Artif. Intell.*, 2013, pp. 155–157.
- [57] D. Ratner and M. Warmuth, “The $(n^2 - 1)$ -puzzle and related relocation problems,” *J. Symbolic Comput.*, vol. 10, pp. 111–137, 1990.
- [58] J. E. Aronson, “A survey on dynamic network flows,” *Ann. Oper. Res.*, vol. 20, no. 1, pp. 1–66, 1989.
- [59] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
- [60] M. Andrews and L. Zhang, “Hardness of the undirected edge-disjoint paths problem,” in *Proc. 37th Annu. ACM Symp. Theory Comput.*, 2005, pp. 276–283.
- [61] L. R. Ford and D. R. Fulkerson, “Constructing maximal dynamic flows from static flows,” *Oper. Res.*, vol. 6, pp. 419–433, 1958.
- [62] N. Robertson and P. D. Seymour, “Graph minors. XIII. The disjoint paths problem,” *J. Combinatorial Theory, B*, vol. 63, no. 1, pp. 65–110, 1995.
- [63] A. Felner, R. E. Korf, R. Meshulam, and R. C. Holte, “Compressed pattern databases,” *J. Artif. Intell. Res.*, vol. 30, pp. 213–247, 2007.
- [64] Gurobi Optimization, Inc., “Gurobi optimizer reference manual,” 2014. [Online]. Available: <http://www.gurobi.com>
- [65] A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
- [66] J. Yu and D. Rus, “An effective algorithmic framework for near optimal multi-robot path planning,” in *Proc. Int. Symp. Robot. Res.*, 2015.



Jingjin Yu received the B.S. degree from the University of Science and Technology, Hefei, China, in 1998, the M.S. degrees in chemistry from the University of Chicago, Chicago, IL, USA, in 2000, in mathematics from the University of Illinois at Chicago, Chicago, in 2001, and in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, in 2010, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Champaign, in 2013.

He is an Assistant Professor in the Department of Computer Science at Rutgers University, New Brunswick, NJ, USA. He was with the University of Illinois at Urbana-Champaign, Champaign, as a Postdoctoral Researcher. He was a Postdoctoral Researcher with Massachusetts Institute of Technology from 2013 to 2015, with a joint appointment with Boston University from 2013 to 2014. He is broadly interested in the areas of robotics and control, focusing on issues related to computational complexity, and the design of efficient algorithms with provable guarantees.



Steven M. LaValle received the Ph.D. degree in electrical engineering from the University of Illinois, Champaign, IL, USA, in 1995.

He is a Professor in the Department of Computer Science at the University of Illinois. From 1995 to 1997, he was a Postdoctoral Researcher and a Lecturer with the Department of Computer Science, Stanford University, Stanford, CA, USA. From 1997 to 2001, he was an Assistant Professor with the Department of Computer Science, Iowa State University, Ames, Iowa, USA. His research interests include robotics, virtual reality, sensing, planning algorithms, computational geometry, and control theory. He is mostly known for his introduction of the Rapidly-Exploring Random Tree algorithm, which is widely used in robotics and other engineering fields. He was also an early Founder and Chief Scientist of Oculus VR, acquired by Facebook in 2014, where he developed patented tracking technology for consumer virtual reality. He led a team of perceptual psychologists to provide principled approaches to virtual reality system calibration, health and safety, and the design of comfortable user experiences. He also authored the books *Planning Algorithms, Sensing and Filtering*, and *Virtual Reality*.

参考文献

- [1] 叶杨笙. 通用移动机器人调度系统研究与设计[D]. 浙江大学, 2018.
- [2] Raffaello D'Andrea, Peter Wurman. 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.
- [3] 哈工大机器人集团推出Bee Robot智能仓储系统[J]. 物流技术与应用, 2018, 23(09): 137.
- [4] 傅雅男, 李朝敏. 浅谈物流视角下的智能仓储机器人的运用与改进[J]. 商场现代化, 2018(23): 77-78.
- [5] 陈若男, 文聪聪, 彭玲, 尤承增. 改进 A*算法及其在室内机器人路径规划中的应用 [J/OL]. 计算机应用: 1-8.
- [6] 陈豪, 李勇, 罗靖迪. 基于改进 A*算法优化的移动机器人路径规划研究[J]. 自动化与仪器仪表, 2018(12): 1-4.
- [7] 夏清松, 唐秋华, 张利平. 多仓储机器人协同路径规划与作业避碰[J/OL]. 信息与控制: 1-8.
- [8] 徐兵兵, 郝荣飞. 机器人路径规划技术的现状与发展[J]. 电子技术与软件工程, 2018(24): 81.
- [9] Harold W. Kuhn. The Hungarian method for the assignment problem[J]. Naval Research Logistics, 1955, 2(1-2): 83-97.
- [10] Ivana Budinská, Štefan Havlí. Task allocation within a heterogeneous multi-robot system[C]. Slovakia: Proceedings of the 28th International Conference on Cybernetics & Infomatics, 2016.
- [11] Jianping Chen, Jianbin Wang, Qijun Xiao, Changxing Chen. A multi-robot task allocation method based on multi-objective optimization[C]. Singapore: 2018 15th International Conference on Control, Automation, Robotics and Vision, 2018.
- [12] Lingzhi Luo, Nilanjan Chakraborty, Katia Sycara. Provably-good distributed algorithm for constrained multi-robot task assignment for grouped gasks[J]. IEEE Transactions on Robotics, 2015, 31(1): 19-30.
- [13] Jingjin Yu, Steven M. LaValle. Optimality and robustness in multi-robot path planning with temporal logic constraints[J]. IEEE Transactions on Robotics, 2016, 32(5): 1163-1177.
- [14] Yunus Emre Sahin, Petter Nilsson, Necmiye Ozay. Provably-correct coordination of large collections of agents with counting temporal logic constraints[C]// ACM/IEEE International Conference on Cyber-Physical Systems, 2017.
- [15] 陈豪, 李勇, 罗靖迪. 基于改进A*算法优化的移动机器人路径规划研究[J]. 自动化与仪器仪表, 2018(12): 1-4.
- [16] 夏清松, 唐秋华, 张利平. 多仓储机器人协同路径规划与作业避碰[J/OL]. 信息与控制: 1-8.
- [17] 张希闻, 肖本贤. 改进 D~*算法的移动机器人路径规划[J]. 传感器与微系统, 2018, 37(12): 52-58.
- [18] Nannan Lu, Yunlu Gong, Jie Pan. Path planning of mobile robot with path rule mining based on GA[C]// IEEE Control & Decision Conference, 2016.

-
- [19] Sekou Diane, Sergey Manko, Valery Lokhin. Forecasting conflicts in multi-robot systems based on intelligent feedback[C]// International Siberian Conference on Control & Communications, 2016.
 - [20] 王晓春. 机器人在仓储管理中的“货到人”技术浅谈——基于亚马逊物流中心 KIVA 橙色机器人[J]. 现代经济信息, 2017(13): 78-80.
 - [21] Jingjin Yu, Steven M. LaValle. Optimal multirobot path planning on graphs: complete algorithms and effective heuristics[J]. IEEE Transactions on Robotics, 2016, 32(5): 1163-1177.
 - [22] 张雨晨, 王竹芳. 基于匈牙利算法的运输问题改进算法[J]. 价值工程, 2019, 38(04): 78-82.

毕业论文（设计）文献综述和开题报告考核

一、对文献综述、外文翻译和开题报告评语及成绩评定

成绩比例	文献综述 占（10%）	开题报告 占（15%）	外文翻译 占（5%）
分值			

开题报告答辩小组负责人（签名）_____

年 月 日