北京大学

本科生毕业论文

题目：**基于组合-混合优化的多智能体合作**

Combinatorial-hybrid Optimization for Multi-agent Systems under Collaborative Tasks

姓　　名：唐子力

学　　号：1900011105

院　　系：工学院

专　　业：机器人工程

导师姓名：国萌　教授

二〇二三年　五月

版权声明

摘　要

多智能体系统可以通过并发和协作工作来高效地完成任务, 例如在物资运输、智能安防、协同搜救等领域. 多智能体的团队协调主要依赖于两个层面的工作: (i) 联盟形成: 针对不同的任务分配机器人子团队; (ii) 协作控制: 对机器人子团队, 设计协作控制策略来执行所分配的任务. 前者可能在团队规模方面是组合性的, 而后者需要在几何和动态约束下优化联合状态空间, 形成有效的控制策略. 现有的工作通常假设一个层面已经给定来解决另一个层面, 忽略它们之间的密切耦合关系.

这项工作将多智能体合作问题形式化为组合-混合优化 (Combinatorial-hybrid Optimization, CHO) 问题, 并提出该类问题的通用解决框架, 包括两个交错的优化层级: 任务联盟的动态形成和协作行为的混合优化, 其中混合优化层次将多智能体的协作策略形式化为离散的协作模式和在特定协作模式下可供优化的连续参数. 优化框架给出了两个层次的具体实现和理论保证, 并提供了可增量学习的图神经网络加速规划方法和代价估计模块. 针对上述算法框架, 我们研究了协作运输和动态抓捕这两个非平凡应用, 通过基准测试和对比实验有效地说明了耦合任务分配与协作行为规划的必要性.

关键词: 多机器人系统, 组合-混合优化, 协同控制

ABSTRACT

Multi-agent systems can be extremely efficient when working concurrently and collaboratively, e.g., for transportation, maintenance, search and rescue. Coordination of such teams often involves two aspects: (i) coalition formation: selecting appropriate sub-teams for different tasks; (ii) collaborative control: designing collaborative control strategies to execute these tasks. The former aspect can be combinatorial w.r.t. the team size, while the latter requires optimization over joint state-spaces under geometric and dynamic constraints. Existing work often tackles one aspect by assuming the other is given, while ignoring their close dependency.

This work formalizes the multi-agent collaboration problem as a combinatorial-mixed-integer optimization (CHO) problem and proposes a general solution framework for such problems, including two interleaved layers: dynamic formation of task alliances and mixed optimization of collaborative behavior, where both the discrete modes of collaboration and the continuous control parameters are optimized simultaneously and iteratively. This work proposes the specific implementations for two layers and provides the theoretical guarantees, as well as an incremental learning graph neural network acceleration method and a cost estimation module.

Two non-trivial applications of collaborative transportation and dynamic capture are studied against several baselines, effectively demonstrating the necessity of coupling two layers.

KEY WORDS: Multi-agent systems, Combinatorial-hybrid optimizations, Cooperative control

目 录

第一章　引言

1.1　研究背景

当前, 异构自主机器人团队愈发受到学术界的广泛关注, 因为其能够通过协作来完成单一机器人难以完成的任务. 比如协作运输 [1; 2]、动态拦截 [3] 和监控 [4] 等任务. 允许机器人团队同时协同工作可以显著提高团队的整体效率和能力. 机器人团队的协作协调通常包括两个方面. 一方面, 可以有多个可能的机器人子团队完成给定的任务, 但是它们的成本可能截然不同 [5; 6; 7]. 例如, 三个机器人可以比单个机器人更快地检测和捕获动态目标 (如果可能的话), 而在某些情况下, 五个机器人可能是冗余的. 因此, 适当的任务分配对于整体性能至关重要, 但不幸的是, 这通常具有与机器人数量和任务数量成指数关系的复杂度. 另一方面, 给定一个任务分配方案, 每个子组的执行通常可以归结为一个最优控制问题 [2; 8], 即如何协同机器人来最小化与任务相关的成本, 同时满足动态和几何约束 [6; 9]. 由于整个协作机器人团队的联合状态-控制空间具有长时间跨度和高维度的特征, 精确优化的复杂度很高. 最后, 通常会出现关于这两个方面的“鸡生蛋”困境 [10; 11]. 换句话说, 任务分配依赖于最优控制来评估可行性和实际成本, 而最优控制问题则需要一个可行的任务分配作为输入. 首先解决所有可能的分配的控制问题, 然后再解决任务分配问题通常是不可行的, 因为这会使两个方面的复杂度都成倍增加.

1.2　相关研究

1.2.1　任务规划问题

在多智能体系统中, 任务规划指的是首先将任务分解为子任务, 然后将其分配给团队的过程, 可参见 [5; 6; 12] 中的综述. 可以选择不同的优化准则, 通常有 MinSUM, 它最小化所有智能体成本之和 [6; 13], 和 MinMAX, 它最小化所有智能体成本的最大值 [14]. MinSUM 和 MinMAX 各有优势, MinSUM 侧重考虑所有智能体的平均成本, 有利于在总体上减小资源消耗, 但对个体的关注不足. 对于典型的多智能体合作场景, 多智能体常常通过并行工作来完成多项任务, 这导致总时间成本和个体最大时间成本密切相关, 这正是 MinMAX 指标的出发点. 然而,MinMAX 往往在代价最高的任务无法被优化时, 放弃优化其他任务的代价.

任务规划领域的经典问题有: 一对一分配问题 [15], 作者采用改进的匈牙利算法来解决将任务集合与智能体集合进行一一匹配从而最大化总收益的问题; 多车辆路径问题 [6; 12], 考虑了在时序和命令约束下的多车路径规划问题; 联盟形成问题 [7; 16],.

为了解决上述问题, 研究者提出了多种经典的方法, 包括匈牙利方法 [15]、混合整数线性规划 (MILP)[5]、基于搜索的方法 [13] 和基于市场的方法 [12].

在上述研究中,[7; 16] 所提出的方法与我们的研究最相关的, 作者提出了一种基于交换操作的分布式节点聚类算法, 用于解决没有中心节点的任务分配问题. 文中定义的交换操作本质是对分配的一种映射, 聚类结果是在所有可能的交换映射下保持代价不减小的联盟集合. 由于其对代价函数的结构和性质没有任何要求, 因此它特别适合于协作任务. 更多关于该类方法的讨论可以参考文献 [17] 中的综合研究. 在文献 [18] 中提出了一个市场基础的联盟形成算法, 但是它没有提供任何质量保证. 此外, 文献 [19] 设计了一个任意时间的但不是最优的算法. 在文献 [20] 中提出了可证明最优的算法, 但是由于组合复杂度的原因, 不能应用于大规模系统. 因此, 在文献 [21; 19] 中提出了分布式方法, 文献 [22] 中使用基于学习的方法, 但是它们无法保证解决方案的质量.

这些研究往往假设任务-智能体成本是静态和已知的, 这在协作任务中并不总是可用或者甚至是无效的, 因为一个智能体参与任务的收益取决于哪些其他智能体也参与任务, 我们无法定义智能体和任务的一对一收益, 而只能计算在特定联盟的基础上的边际收益. 同时, 对于部分多智能体合作场景, 即使是边际收益也是难以直接估计的.

1.2.2 运动规划问题

在多智能体系统中, 运动规划指的是为每个智能体设计控制策略以完成给定任务的过程. 一个常见的任务是协作导航, 其中每个智能体导航到其目标位置, 同时避免与其他智能体或障碍物发生碰撞, 详见 [23; 9]. 其他任务, 如编队、群集, 也在各种约束条件下进行研究, 详见 [24]. 另一个相关的任务是协作运输 [25; 26], 其中多个智能体通过推动或抓握将一个物体运输到目的地. 由于不同任务涉及的动态和几何约束, 这些运动规划问题仍然具有挑战性. 此外, 这些研究大多假设整个团队有一个全局目标, 无需进行任务分解和分配.

对于多智能体运动规划问题, 常见的算法通常基于搜索、采样或者优化三者中的一个或者多个. 对于多智能体的运动规划问题而言, 使用离散图搜索进行规划可以处理非凸空间以保证最优性, 但随着搜索维度的增加而遭受指数级的计算复杂度. 基于采样的规划器可以快速找到初始解, 并具有强大的最优解收敛保证. 然而, 在实践中, 算法找到的初始解距最优解往往较远, 收敛速度较慢, 并且解通常需要进行一些后处理来修复其中质量较差的部分. 基于搜索的算法可以利用运动原语 (Motion Primitives), 即预计算的轨迹在连续的空间中执行离散搜索来缓解上述问题, 然而, 将此方法扩展到更高的维度已被证明是困难的, 并且往往需要非常仔细地设计运动原语. 优化型规划器可以克服这种维数诅咒, 它们的复杂度随着状态维数的增加呈多项式级别而不是指数级别. 然而, 基于优化的规划器通常仅具有局部最优性质. 常用的做法是通过搜索或者采样确定恰当的凸域, 否则优化算法将陷入无处不在的局部最优中, 无法转移到最优解所在的凸域中.

目前已经有许多工作尝试耦合上述三种方法中的多个, 而不是简单地顺序执行. 在 [27] 中, 作者引入了一个带有全局推理能力的激进四旋翼轨迹生成框架, 结合了轨迹优化和离散图搜索的优点, 交替使用这两种方法, 并生成具有离散化可完备性保证的轨迹. 作者通过将优化问题建模为无约束的 QP 问题从而保证了算法的求解效率. 更多关于三种方法的对比和结合方式可以参考 [28].

1.2.3 任务与运动联合规划

将上述两个方面整合起来, 就构成了多智能体系统的任务和运动规划 (TAMP) 问题. 在 [11] 中, 作者提出了基于逐步回退的任务分解和运动规划相结合的方法来进行自主装配. 类似地,[29] 考虑了一项家具组装任务, 引入重新抓取操作来分解长期的组装操作. 这两项工作都非常强调在协作操作中的物理稳定性和顺序可行性, 但由于只考虑了少数几个智能体, 因此忽略了组合方面的问题. 在 [3; 30] 中考虑了经典的“警察和小偷”游戏, 其中多个追逐者的任务是捕捉多个逃犯. 所提出的解决方案通过采用贪心的分配策略, 如顺序选择最近目标 [3] 或最大匹配对 [30], 将任务分配和控制器设计解耦. 因此, 仍然需要一个集成的解决方案, 用于解决多智能体 TAMP 问题, 能够同时处理组合任务分配和协作控制设计.

1.3 本文方法

为了克服这些挑战, 本文提出了一个针对协作任务的多智能体系统的组合-混合优化 (Combinatorial-Hybrid Optimization, CHO) 框架. 多个智能体可以组成一个联盟协同完成一个共同的任务, 其成本不仅取决于参与人数, 还取决于协作方式和潜在的控制参数. 因此, 该框架包括两个层次: (i) 联盟结构的优化 (ii) 最优协作行为的混合搜索, 包括离散模式和连续控制参数的序列. 两个层次并发进行, 层次 (i) 在执行过程中动态选择是否执行层次 (ii), 具体来说, 只有在任务分配中需要计算特定联盟的成本时才为该联盟解决混合搜索问题以获得其实际成本. 优化框架给出了两个层次的具体实现, 联盟形成层, 我们提出了集中式 Nash 稳定联盟形成算法 (Nash-Stable Coalition Formation, NS-COAL) 与作为其拓展的 K-Serial 稳定联盟形成算法 (K-Serial Coalition Formation, KS-COAL) 算法, 并提出了一种基于图神经网络 (Graph Neural Networks, GNNs) 的神经加速器 (NAC) 来提升 KS-COAL 算法的运行效率. 在混合搜素层, 我们提出了启发式梯度引导混合搜索算法 (Heuristic-Gradient Guided Hybrid Search, HGG-HS), 旨在平衡混合搜索的效率和求解质量. 此外, 在两个层次的交互层面, 我们提供了可增量学习的代价估计模块, 用于提升估计代价函数的精度, 从而达到减少混合优化层调用次数, 提升算法效率的目的.

研究表明, 联盟结构和相关协作行为的最终解是纳什稳定的, 且具有 K-Serial 稳定的质量保证. 为了展示其适用性, 本文模拟并解决了两个非平凡的多智能体协作任务, 包括协作

运输和动态抓捕.

本文的主要贡献在于两个方面: (i) 针对协作任务的多智能体系统的组合混合优化问题的建模, 特别是针对代价不存在且只能在解决混合优化问题后得出的情况; (ii) 所提出的框架同时找到联盟形成和最优协作行为, 并具有可证明的质量保证. (iii) 提出的基于学习的加速框架在保证求解质量的同时, 有效解耦了两个层次的复杂度, 且能够通用于大部分异构团队合作任务.

第二章　问题描述与应用场景

2.1　工作空间和智能体

考虑一个 N 个智能体的团队, 它们在一个共享的工作空间 $\mathcal{X} \subset \mathbb{R}^X$ 中协作. 系统状态 $x \in \mathcal{X}$ 不仅包括智能体状态, 还包括可移动的物体和目标等其他动态组件. 由于动态和几何约束, 如智能体之间和障碍物之间的避碰, 系统需要保持在允许的子集 $\mathcal{X}_{\texttt{safe}} \subset \mathcal{X}$ 内.

2.2　参数化的协作模式

此外, 这些智能体可以通过多种参数化模式改变系统状态, 表示为 $\Xi \triangleq \xi_1, \cdots, \xi_K$. 在每个模式 $\xi_k \in \Xi$ 下, 系统状态会根据一个闭环动态进行演化, 即

$$x(t+1) \triangleq h_k\big(x(t), \mathcal{N}_k, \rho_k\big), \forall t \in [t_0, t_0 + T_k], \tag{2.1}$$

其中 $k \in \mathcal{K} \triangleq 1, \cdots, K$ 表示一个有效的模式; $\mathcal{N}_k \subseteq \mathcal{N}$ 是参与该模式的智能体子集（称为联盟）; $\rho_k \in \mathbb{R}^{P_k}$ 是选择该模式的连续参数, 具有维度 P_k; $x(t)$ 和 $x(t+1)$ 分别表示智能体在 $\mathcal{N}_k$ 中执行带参数 ρ_k 的模式 ξ_k 一次时间步骤前后的系统状态; $t_0 \geq 0$ 是任意的起始时间; T_k 是模式 ξ_k 给定的最小持续时间. 对于性能度量, 每个模式 ξ_k 在特定的 $(\mathcal{N}_k, \rho_k)$ 选择下都有一个代价函数 $c_k : \mathcal{X} \times 2^{\mathcal{N}} \times \mathbb{R}^{P_k} \to \mathbb{R}^+$ 与之关联. 本文假设与每个模式 $\xi_k \in \Xi$ 相关的函数 h_k, c_k 都可以通过显式函数或者数值模拟来获取. 此类模式通常是建立在预先为特定且更简单的目的而设计的成熟功能模块之上的, 具有连续的可变参数, 用于调整在该模式下的具体行为.

模式层的抽象的主要目的是降低多智能体合作规划的复杂度, 降低决策空间的维度. 多个机器人的联合控制变量 $U = [u_1, u_2, \cdots, u_N]$ 的维度和机器人的数目 N 成正比, 这带来了指数级的策略空间的膨胀. 然而, 在这个指数级增长的策略空间中, 大部分控制量的组合是无意义或者低效的, 比如两个机器人的控制量分别引导系统状态向着相反的方向发展, 或者不同的控制量组合对于系统的影响是相同的. 因此, 将多机器人的合作按照具体策略抽象为参数化模式是合乎情理的, 基于经验设计的成熟合作模式往往可以排除上述的无意义的合控制参数, 将高维控制变量压缩为较少的连续参数, 同时不损失大部分的控制能力. 关于模式的具体设计的更多示例可以在第 2.5节中找到.

2.3 合作任务

此外, 团队需要完成 M 项任务, 表示为 $\Omega \triangleq \omega_1, \cdots, \omega_M$. 在最普遍的意义上, 每个任务 $\omega_m \in \Omega$ 的目标都是将系统状态 x 转移到目标状态集合 $\mathcal{X}_{G_m} \in \mathcal{X}$. 为了事先这种转化, 每个任务都可以由恰当混合计划来完成, 即为任务采用适当选择的机器人联盟和参数化的模式序列, 即:

$$\varphi_m \triangleq (\xi_{k_1^m}, \mathcal{N}_{k_1^m}, \rho_{k_1^m}) \cdots (\xi_{k_L^m}, \mathcal{N}_{k_L^m}, \rho_{k_L^m}), \tag{2.2}$$

其中 $L > 0$ 是模式序列的长度, 其具体大小与我们规划的时间步长有关, 通常来说, 我们我们希望 L 取恰当值, 避免过于粗糙的规划导致的收益不精确, 或者过于精细的规划带来的高计算成本和不必要的模式切换; 对于每个模式 $\xi_{k_\ell^m} \in \Xi, \forall \ell = 1, \cdots, L$, $(\mathcal{N}_{k_1^m}, \rho_{k_1^m})$ 是如 (2.1) 节中所描述的特定模式下系统动力学函数 h_{ξ_k} 所允许的联盟和参数. 因此, 系统状态在混合规划方案 φ_m 下的演化过程收到如下的动力学和边界条件约束:

$$\begin{aligned} x_{k_\ell^m} &= h_{k_\ell^m}\big(x_{k_{\ell-1}^m}, \mathcal{N}_{k_\ell^m}, \rho_{k_\ell^m}\big); \\ x_{k_0^m} &= x_0, \; x_{k_L^m} \in \mathcal{X}_{G_m}, \end{aligned} \tag{2.3}$$

其中 $x_0 \in \mathcal{X}$ 是给定的初始状态. 计划 φ_m 的相关成本由累计成本给出, 即

$$c(\varphi_m) \triangleq \sum_{\ell} c_{k_\ell^m}(x_{k_{\ell-1}^m}, \mathcal{N}_{k_\ell^m}, \rho_{k_\ell^m}),$$

对于每个 $\omega_m \in \Omega$ 都成立. 此外, 由于不同的模式可以以并发方式执行不同的任务, 因此在本工作中假定不同的任务以独立的方式更改状态的不同维度, 从而我们可以将不同子联盟对系统的影响解耦, 对每个子联盟, 单独考虑其方案对系统状态的改变量系统的完整改变量即等于各子联盟状态改变量之和, 即

$$x(t+1) = x(t) + \sum_{\omega_m \in \Omega} \Big(h_{k_t^m}\big(x(t), \mathcal{N}_{k_t^m}, \rho_{k_t^m}\big) - x(t) \Big),$$

其中,$\xi_{k_t^m} \in \Xi$ 是任务 ω_m 在时间 $t \geq 0$ 的活动模式; $(\mathcal{N}_{k_t^m}, \rho_{k_t^m})$ 是关联的联盟和参数; $x(t)$ 是当前系统状态; $x(t+1)$ 是执行所有活动模式一步后的结果状态. 最后, 由于每个智能体只能参与最多一个任务, 因此有:

$$\mathcal{N}_{m_1}(t) \cap \mathcal{N}_{m_2}(t) = \emptyset, ; \forall \omega_{m_1}, \omega_{m_2} \in \Omega; \tag{2.4}$$

其中,$\mathcal{N}_{m_1}(t), \mathcal{N}_{m_2}(t)$ 表示在时间 $t \geq 0$ 执行任意两个任务 $\omega_{m_1}, \omega_{m_2} \in \Omega$ 所负责的联盟.

2.4 组合-混合优化问题 (CHO)

问题 1. *考虑上述 N 个智能体和 M 个任务的模型, 完整的组合混合优化 (CHO) 问题定义如下:*

$$\begin{aligned} \underset{\{\varphi_m\}}{\boldsymbol{min}} \quad & \left\{ \underset{m}{\boldsymbol{max}} \{c(\varphi_m)\} + \frac{1}{M} \sum_m c(\varphi_m) \right\} \\ \boldsymbol{s.t.} \quad & (2.3) - (2.4), \forall \ell, \forall m; \end{aligned} \tag{2.5}$$

其中 φ_m 是所有任务的混合计划集合; 目标是在所有任务中最大代价和平均代价之间实现平衡的最小化代价; (2.3)-(2.4)是与系统状态和联盟结构相关的动态和几何约束条件. ■

尽管我们已经对合作任务下的运动规划问题抽象为了关于参数化模式序列的混合整数规划, 但是组合-混合优化问题仍旧相当困难, 主要原因仍然是上下层求解复杂度的耦合, 联盟形成的结果将直接影响运动规划的优化问题的结构, 这导致该问题本质上是高度非凸和非线性的. 传统的优化算法将面临指数爆炸和容易陷入低质量局部最优解的问题. 因此,CHO 问题亟待提出一个广泛适用的高效率计算框架, 且充分利用多智能体合作问题的问题结构. 并且, 我们希望我们提出的计算框架能够充分考虑实际需求, 便于部署在多机器人实物系统中, 高效地完成各种场景下的规划.

2.5 应用场景

为了使得第 二 节中描述的问题表述更加具体, 本节首先介绍了可以通过 CHO 框架建模的多智能体系统的两种用例. 这些用例的详细设置和结果在第 四 节中给出.

2.5.1 合作搬运

在本节中, 我们考虑了一个经典的合作搬运问题. 工作空间为 2D 平面, 地图中包含散乱摆放的障碍物, 障碍物被固定在地面上不可移动. 在工作空间中包含数个可以移动的矩形盒子和具有圆形边缘的机器人.

N 个机器人组成的团队的任务是将 M 个相同的矩形盒子从它们的初始位置移动到杂乱无序的工作空间中的目标位置, 如图 2.1所示. 盒子的状态由其中心坐标 (x_m, y_m) 和旋转角度 ψ_m, 以及每个变量的一阶导数 $(\dot{x}_m, \dot{y}_m, \dot{\psi}_m)$ 确定. 其中目标位置为 $(x_m^\star, y_m^\star)$. 机器人的状态由其中心坐标 (x_n, y_n) 和其导数 $(\dot{x}_n, \dot{(v)}_n)$ 决定. 系统的状态变量 x 同时包含盒子和机器人的状态.

盒子被视作质量均匀的刚体, 具有平移不变的 2 阶动力学, 其所受合力与合力矩如下:

$$\begin{aligned} J\beta = &\ M_f + \Sigma M_{F_n} \\ m\vec{\alpha} = &\ \vec{f} + \Sigma\overrightarrow{F_n} \end{aligned} \tag{2.6}$$

其中 J 为盒子绕质心的垂直旋转轴的转动惯量, m 为盒子质量. f 为盒子所受地面摩擦力, Fn 为机器人 n 对盒子的推力矢量, M_n 为相应的推力力矩. 给定受力后, 盒子的状态按照 2 阶模型进行更新:

$$\begin{aligned} \dot{x}, \dot{y}, \dot{\psi} &= v_x, v_y, \omega \\ \dot{v}_x, \dot{v}_y, \dot{\omega} &= a_x, a_y, \beta \end{aligned} \tag{2.7}$$

其中,$\overrightarrow{F_n}$ 是机器人 n 对盒子的推力. 此处为了简化模型, 我们忽略了机器人与和盒子之间的摩擦力, 仅考虑其法向作用力. 刚体的与地面的摩擦力可以被按照如下积分式计算:

$$\overrightarrow{f} = \int_s \overrightarrow{v_s}\rho g ds = \int_y \int_x \rho g \left(\overrightarrow{\omega} \times \overrightarrow{r_{(x,y)}} + \overrightarrow{v_c}\right) dxdy \tag{2.8}$$

然而, 积分形式并不适合用于构建优化问题, 因此我们在实际构建优化问题的模型约束中, 我们通过将刚体划分离散网格的形式, 用离散求和来近似数值积分. 通常来说, 过于精细的网格划分是不必要的.

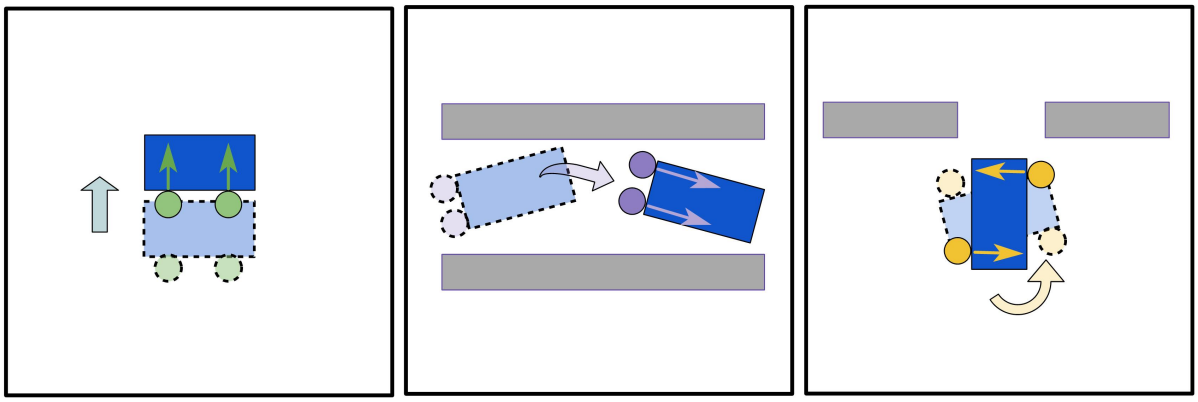


图 2.1. 在 Sec. 2.5.1中描述的合作搬运问题的三种参数化模式 (两个机器人): 长边推动 (**左**), 短边推动 (**中**), 和旋转推动 (**右**).

智能体可以在盒子的特定点接触并施加推力 $F_n \in [0, F_{\max}]$ 将其向前推. 不同的接触点组合会导致不同的系统动力学, 从而产生不同的模式 $\Xi = \xi_k$. 而其模式参数是每个机器人施加的推力 $\rho_k = F_n$, 参数的维度等于机器人数目. 而在特定接触模式下系统的动力学则由式子 (2.6),(2.7),(2.8) 给出.

图 2.1里列出了对于两个机器人的子团队可以采取的三种参数化模式, 包括 (i) 长边推动,(ii) 窄边推动,(iii) 旋转推动, 即在盒子的两侧施加方向相反的力使盒子旋转. 显然, 三种

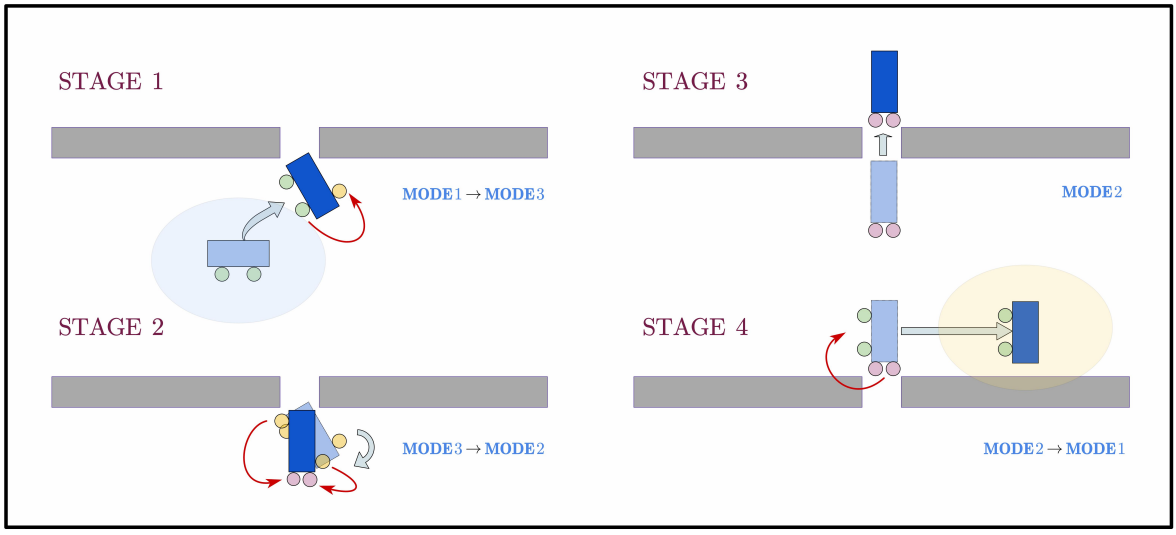


图 2.2. 机器人团队通过变换接触模式使得盒子可以通过狭窄的通道

模式有各自适应的场景. 在空旷的环境中, 采用长边推动是最为合适的, 因为在这种接触模式下, 系统的可控性更高, 相应地具有更小的控制代价. 而在狭窄的通道中, 由于几何约束的限制, 机器人团队不得不采用短边推动模式, 尽管这可能意味着更大的不确定性和控制代价, 以及更小的曲率约束范围. 而如果我们希望获得更大的曲率范围, 则可以采用旋转推动模式, 机器人施加相反方向的力使得盒子可以在原地旋转, 从而调整到更好的位姿.

显然, 更多的智能体可以带来更高的可控性和更快的移动速度, 但同时也带来更大的控制成本, 随着机器人数目增大, 收益将边际递减.

此外, 如图2.2所示, 为了完成给定任务目标, 可能需要一系列具有不同参数的不同模式, 图中机器人在狭窄的通道口首先转换到旋转推动模式, 调整盒子的角度以对准通道口, 然后切换到短边推动模式, 使盒子通过通道.

2.5.2 动态抓捕问题

本问题考虑的是动态捕捉问题, 涉及到 N 个追捕者和 M 个逃避者在一个杂乱的二维工作区域内的运动. 它们的状态和速度分别用 $(x_n(t), v_n(t))$ 和 $(y_m(t), v_m(t))$ 表示. 所有代理都具有相同的最大速度 $v_{\max}$, 这意味着追捕者需要进行协作才能捕捉到逃避者. 每个逃避者都有一个与之对应的捕捉任务, 它以相对距离为权重向着所有追捕者的相反方向运动.

此外, 如图 2.3 所示, 每个任务有三种参数化模式 $\Xi = \xi_1, \xi_2, \xi_3$: (i) 纯追击. 追捕者遵循“视线”策略, 向着指定的逃避者移动, 即 $v_n = v_{\max}\texttt{point}(x_n, \rho_1)$, 其中 $\texttt{point}(x_n, \rho_1)$ 是由被指定的逃避者的位置 $\rho_1 \in \mathbb{R}^2$ 确定的速度方向单位向量. (ii) 躲藏和攻击. 追捕者藏在障碍物后面, 直到指定的逃避者出现, 然后包围它, 即 $v_n = \texttt{nav}(x_n, \rho_2)$, 其中 $\texttt{nav}(\cdot)$ 是一个导航函数, 它将追捕者引导到隐藏或攻击位置 $\rho_2 \in \mathbb{R}^2$. (iii) 围捕. 追捕者通过减小逃避者的

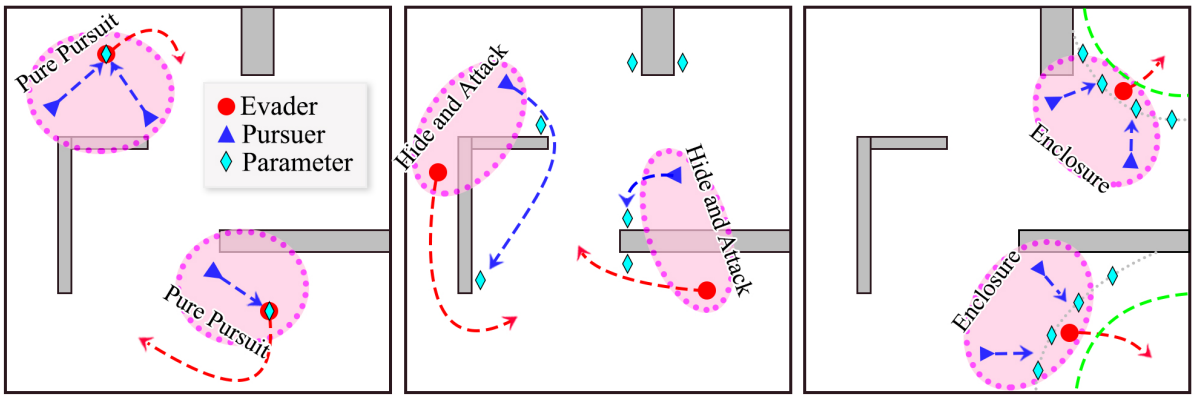


图 2.3. 第 Sec. 2.5.2节中描述的动态抓捕问题的三种参数化模式: 纯追踪 (左), 隐藏和突击 (中), 和围捕 (右).

优势区域 [31] 来将被指定的逃避者驱赶到一个角落, 直到被捕捉, 即 $v_n = \texttt{corner}(x_n, \rho_3)$, 其中 $\texttt{corner}(\cdot)$ 是一个优化算法, 在给定其他的追踪者 $\{x_n\}$ 和关键位置 $\rho_3 \in \mathbb{R}^2$ 的条件下计算最优的速度方向. 因此, 每个追捕者都应该选择一系列模式、联盟和相关参数, 以尽快捕捉所有逃避者.

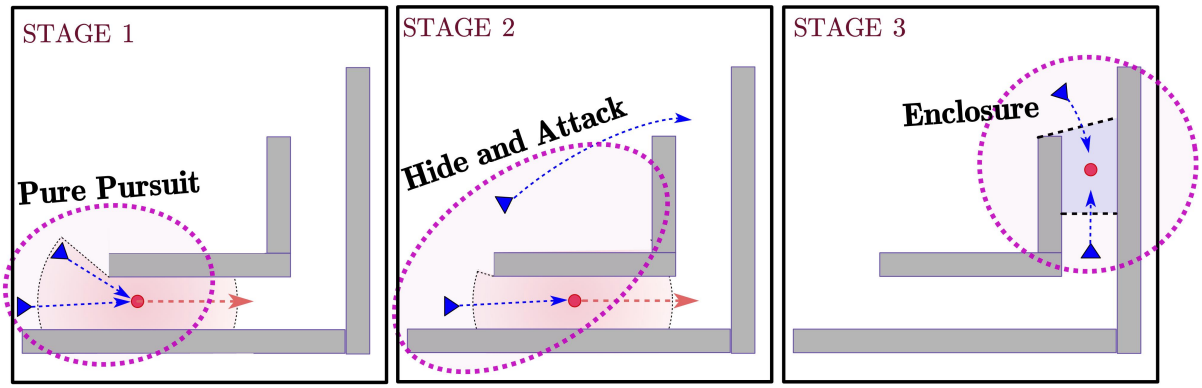


图 2.4. 机器人团队通过转换合作模式来完成抓捕任务

如图2.4所示, 对于困难的抓捕任务, 多个智能体切换合作模式可能是必须的. 图中的抓捕过程分为三个阶段, 阶段一中两个抓捕者采用纯追踪的模式, 但是由于逃逸者速度较大, 纯追踪无法在有限的时间内完成任务, 因此在阶段二中, 抓捕者切换到隐藏和突击模式, 其中一个抓捕者逼迫逃避者沿着障碍物形成的狭窄通道前进, 而另一个抓捕者则依托障碍物形成的视线遮蔽, 转移到通道的另一个出口等待逃避者出现. 阶段三中, 两个抓捕者在通道口形成了包围, 逃避者的优势范围被局限在了一个封闭的多边形中, 这导致逃避者必然在短时间内被捕捉.

第三章　解决方案

本章提出的解决方案通过两个交错且并行的层次来解决问题 1: (i) 联盟形成 (ii) 合作行为的混合优化. 第一层为每个任务提出候选联盟, 以最小化整体成本. 然后, 将这些候选联盟发送到第二层, 为每个联盟解决受约束的混合优化, 以确定完成分配任务的最佳混合计划和实际成本. 这些成本随后被反馈到第一层, 以调整现有联盟. 这个过程重复直到找到一个 Nash 稳定解或 K-Serial 稳定解. 每层在本章节中都有详细描述.

3.1　联盟形成问题

第一层将任务集分配给可用智能体集合. 首先, 我们将根据相关文献 [17; 32] 对任务分配问题进行形式化描述. 特别地, 我们考虑按如下定义的联盟结构:

$$\mathcal{F} \triangleq (\mathcal{R}, \Omega, f), \tag{3.1}$$

其中 $\mathcal{R} = 1, \cdots, N$ 表示 N 个智能体的团队; $\Omega = \omega_1, \cdots, \omega_M$ 表示 M 个任务的集合; $f : 2^{\mathcal{R}} \times \Omega \to \mathbb{R}^+$ 是给定任务 Ω 中任意潜在联盟的代价函数. 由于上述代价函数通常是未知的, 因此最初通过简单的启发式方法（例如欧几里得距离）, 表示为 $\widehat{f}$. 特别地, 为了刻画不同机器人的能力范围, 我们用 Ω_i 代表智能体 i 可以执行的任务集合, 如果存在一个智能体可以执行任务 ω_{m_1} 和 ω_{m_2}, 则称 ω_{m_1} 和 ω_{m_2} 是相邻的. 相应的, 如果两个智能体 i 和 j 可以执行同一个任务, 则称智能体 i 和 j 是相邻的, 符号表示为 $i \in \mathcal{N}(j)$.

定义 1 (任务分配). *一个任务分配 (Assignment) 被定义为在联盟结构 $\mathcal{F}$ (3.1) 中的一个有效解, 记为 $\mu = (\mathcal{R}_m, \omega_m), \forall \omega_m \in \Omega$, 其中 $\mathcal{R}_m \subset \mathcal{R}$ 是执行公共任务 $\omega_m \in \Omega$ 的联盟, 且 $\mathcal{R}_{m_1} \cap \mathcal{R}_{m_2} = \emptyset, \forall m_1 \neq m_2$. 此外, 对一个任务分配 μ, 估计成本如下:*

$$C(\mu) \triangleq \max_{\omega_m \in \Omega} \widehat{f}_m + \frac{1}{M} \sum_{\omega_m \in \Omega}, \widehat{f}_m, \tag{3.2}$$

其中 $\widehat{f}_m \triangleq \widehat{f}(\mathcal{R}_m, \omega_m)$ 是每个联盟的估计成本; $C(\cdot)$ 是基于 (2.5) 的平衡成本.

■

设 $\mu(\omega_m) = \mathcal{R}_m$ 表示任务 ω_m 所对应的联盟,$\mu(n) = \omega_m$ 表示分配给代理 n 的任务. 更重要的是, 可以通过以下切换操作修改分配 μ.

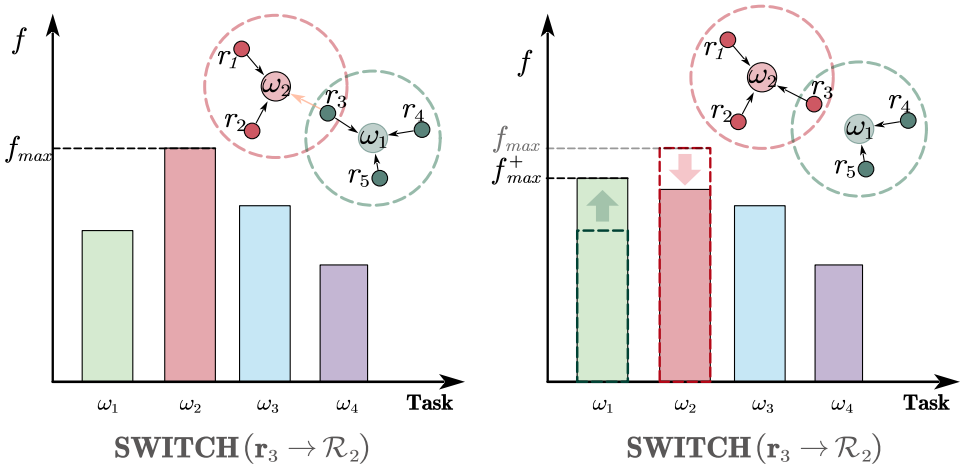


图 3.1. 在3.2小节中介绍的联盟形成算法示意. 智能体通过切换所选联盟来降低式 (3.2)所描述的总代价.

定义 2 (切换操作). 分配给代理 $n \in \mathcal{R}$ 任务 $\omega_m \in \Omega$ 的操作称为切换操作, 记作 ϕ_n^m. 当且仅当代理 n 能够执行任务 ω_m 时, 切换 ϕ_n^m 才是有效的. 因此, 通过 ϕ_n^m, 分配 μ 可以被修改为新的分配 $\hat{\mu}$, 使得 $\hat{\mu}(n) = \omega_m$. 为简便起见, 记 $\hat{\mu} = \phi_n^m(\mu)$. ■

定义 3 (Nash 稳定). 一个任务分配 $\mu^\star$ 被称作 *Nash* 稳定的, 如果不存在任意切换操作 σ_n^m 使得 $C(\sigma_n^m(\mu)) < C(\mu^\star)$, 其中代价函数 $C(\cdot)$ 按照 (3.2)中定义. $\forall n \in \mathcal{N}$. ■

问题 2. 给定式 (3.1) 中的联盟结构 $\mathcal{F}$, 通过定义 *3*, 找到一个 *Nash* 稳定的分配 $\mu^\star$. ■

3.2 集中式 Nash 稳定联盟形成

在该小节中, 我们将提出一种名为集中式 Nash 稳定联盟形成 (Nash-Stable Coalition Formation, NS-COAL) 的集中式任务分配的解决方案, 采用迭代进行的切换操作来降低所有任务的最大成本, 并最终导出 Nash 稳定的任务分配解.

3.2.1 算法描述

首先, 我们需要随机或基于估计代价函数贪婪地导出初始分配 μ_0. 对于初始分配中的 M 个联盟, 我们按照初始的估计代价对联盟进行排序, 代价相等导致的并列情况可以按任意方式打破:

$$(\mathcal{R}_{m^{[k]}}, \omega_{m^{[k]}}) = \mathbf{argmax}^{[k]}_{(\mathcal{R}_m, \omega_m) \in \mu_0} \widehat{f}_m, \tag{3.3}$$

算法将尝试将切换操作作用在具有较大估计代价的联盟, 从而高效地降低总代价. 为了实现这一点, 算法初始化循环指标 $k = 1$, 并开始一个循环, 循环内部依次执行下面的步骤:

1. 选择具有第 k 大估计代价的目标联盟, 计算其实际代价, 并更新估计代价字典. 如果再次访问该联盟的代价时可以直接调用.

2. 假如此时该联盟不再是第 k 大, 则重新选择第 k 大的联盟, 回到 (1). 如果仍然是第 k 大, 则继续执行步骤 (3)

3. 遍历所有可执行任务 $m^{[k]}$ 的智能体 n, 判断下式是否成立:

$$\begin{aligned}&\mathbf{max}\left\{f\left(\mathcal{R}_{m^\star}\cap\{n\},\,\omega_{m^\star}\right),\,f\left(\mathcal{R}_{m^{[k^-]}}\backslash\{n\},\,\omega_{m^{[k^-]}}\right)\right\}\\&<\mathbf{max}\left\{\widehat{f}(\mathcal{R}_{m^\star},\,\omega_{m^\star}),\,\widehat{f}(\mathcal{R}_{m^{[k^-]}},\,\omega_{m^{[k^-]}})\right\},\end{aligned}\tag{3.4}$$

   其中 $(\mathcal{R}_{m^{[k^-]}},\omega_{m^{[k^-]}})$ 是智能体 n 原来所属的联盟, 该联盟具有第 k^- 大的总代价. 如果成立, 则应用切换操作, 获得新的分配. 并且检查 $k_b=1,2,\cdots,k$ 如果 $\omega_{m^{[k_b]}}$ 和 $\omega_{m^\star}$ 或 $\omega_{m^{[k^-]}}$ 相邻, 则使得迭代指标 k 回到 $\mathbf{min}(k,k_b)$, 并返回到 (1). 需要注意的是, 不等式3.4左侧为实际代价, 因此如果该联盟的实际代价尚未被计算过, 则需要调用混合优化层求解实际代价. 但在后文中我们将讨论一种情况, 如果我们能够保证估计收益为实际收益的下界的条件, 则此处不等式左侧的收益可以替换为估计收益而不影响解的质量.

4. k 自增 1.

3.2.2 性能分析

命题 1. *最终的分配 μ_{K_M} 是实际成本函数下,$\mathcal{F}$ 的 Nash 稳定分配.*

证明. 首先,μ_{K_M} 中所有联盟的成本是通过混合优化导出的实际成本. 其次, 每个联盟都找不到能够降低总体成本的切换操作, 满足定义 3 中的 Nash 稳定条件. □

定理 3.2.1. *如果对任意联盟, 估计代价始终小于等于实际代价, 即 $\widehat{f}(\mathcal{R}_{\updownarrow},\omega_m)\le f(\mathcal{R}_{\updownarrow},\omega_m)$ 则在保证最终分配仍然为 Nash 稳定分配的基础上, 算法中的条件3.4左侧的实际代价可以被替换为估计代价,*

证明. 我们将算法中的条件3.4左侧的实际代价替换为始终低估的估计代价, 获得以下不等式:

$$\begin{aligned}&\mathbf{max}\left\{\widehat{f}\left(\mathcal{R}_{m^\star}\cap\{n\},\,\omega_{m^\star}\right),\,\widehat{f}\left(\mathcal{R}_{m^{[k^-]}}\backslash\{n\},\,\omega_{m^{[k^-]}}\right)\right\}\\&<\mathbf{max}\left\{\widehat{f}(\mathcal{R}_{m^\star},\,\omega_{m^\star}),\,\widehat{f}(\mathcal{R}_{m^{[k^-]}},\,\omega_{m^{[k^-]}})\right\},\end{aligned}\tag{3.5}$$

假设新算法的收敛解 μ^* 不是 Nash 稳定解, 则一定存在一个切换操作 ϕ 使得 μ^* 的实际代价下降, 即 ϕ 使得不等式3.4成立. 然而, 收敛解 μ^* 一定对所有可能的切换操作都不满足条

Algorithm 1: NS-COAL 算法

Input: Coalition structure $\mathcal{F}$.

Output: Nash-Stable 分配 $\mu^{\star}$.

/* Initial Assignment */

Initialize μ as μ_0 by randomization or greedy ways only by estimated cost $\hat{C}$

Initialize loop index k=0

while *not terminated* **do**

  /* sort the coalitions and select the target coalition */

4 $(\mathcal{R}_{m^{[k]}}, \omega_{m^{[k]}}) = \mathbf{argmax}^{[k]}_{(\mathcal{R}_m, \omega_m) \in \mu_0} \widehat{f}_m$,

  /* calculate real cost of target coalition by solving hybrid optimization */

5 $\widehat{f}(\mathcal{R}_{m^{[k]}}, \omega_{m^{[k]}}) \leftarrow \mathbf{HO}(\mathcal{R}_{m^{[k]}}, \omega_{m^{[k]}})$

6 **if** $(\mathcal{R}_{m^{[k]}}, \omega_{m^{[k]}})$ *has no longer the k-th largest cost* **then**

7 jump to line 4.

8 **for** $n \in \{n | m^{[k]} \in \Omega_n\}$ **do**

9 **if** *Inequality in 3.4 holds* **then**

  /* apply switch operation on current assignment. */

10 $\mu \leftarrow \phi_n^{m^{[k]}} \mu$

11 **for** $k_b = 1, 2, \cdots, k$ **do**

12 **if** $\omega_{m^{[k_b]}} \in \mathcal{N}\omega_{m^{\star}}$ *or* $\omega_{m^{[k_b]}} \in \mathcal{N}\omega_{m^{[k^-]}}$ **then**

13 $k \leftarrow \mathbf{min}(k, k_b)$ jump to line 4.

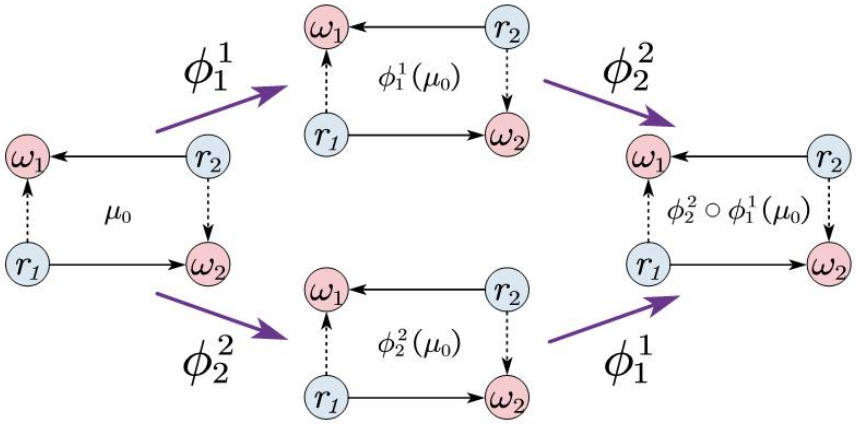
14 $k \leftarrow k + 1$

件3.5, 且由于估计代价小于实际代价, 因而对于所有切换操作均有:

$$\begin{aligned} &\mathbf{max}\left\{f\left(\mathcal{R}_{m^{\star}} \cap \{n\}, \omega_{m^{\star}}\right), f\left(\mathcal{R}_{m^{[k^-]}} \backslash \{n\}, \omega_{m^{[k^-]}}\right)\right\} \\ &\geq \mathbf{max}\left\{\widehat{f}\left(\mathcal{R}_{m^{\star}} \cap \{n\}, \omega_{m^{\star}}\right), \widehat{f}\left(\mathcal{R}_{m^{[k^-]}} \backslash \{n\}, \omega_{m^{[k^-]}}\right)\right\} \\ &\geq \mathbf{max}\left\{\widehat{f}(\mathcal{R}_{m^{\star}}, \omega_{m^{\star}}), \widehat{f}(\mathcal{R}_{m^{[k^-]}}, \omega_{m^{[k^-]}})\right\}, \end{aligned} \tag{3.6}$$

即对于所有切换操作, 不等式3.4不成立, 矛盾, 因此原命题成立. □

注解 1. 需要说明的是,*Nash* 稳定解只是一种局部最优解, 为了追求获得更低成本的任务分配解, 我们进一步考察了任务分配问题和 *Nash* 稳定分配的数学性质. 图 *3.2*展示了一个特殊的场景. 两个机器人需要被分配给两个任务. 当前的任务分配如图中紫色实线箭头所示, 图中以离散形式给出了所有任务的成本函数. 按照成本函数和*3.2*中的综合成本定义, 我们可以计算出当前分配的成本: $\mathrm{C}(\mu_0) = \mathbf{max}\ f_i + \mathbf{mean}\ f_i = 4 + (4+4)/2 = 8$. 对于该分配,



$$f(\omega_1,\{r_1\})=2, f(\omega_1,\{r_2\})=4, f(\omega_2,\{r_1\})=4, f(\omega_2,\{r_2\})=2$$
$$f(\omega_1,\{r_1,r_2\})=2, f(\omega_2,\{r_1,r_2\})=1, f(\omega_2,\emptyset)=10, f(\omega_2,\emptyset)=10$$

图 3.2. Nash 稳定的次优解 (μ_0) 和最优解 $(\phi_2^2 \circ \phi_1^1(\mu_0))$

当前可以选择的切换操作有 ϕ_1^1, ϕ_2^2, 其作用在分配上得到的结果如下所示:

$$\begin{aligned}\phi_1^1(\mu_0) &= \left\{\left(\omega_1, \{\mathrm{r}_1, \mathrm{r}_2\}\right), (\omega_2,)\right\}, \mathrm{C}\left(\phi_1^1(\mu_0)\right) = 15.5 > \mathrm{C}(\mu_0) \\ \phi_1^2(\mu_0) &= \left\{(\omega_1,), \left(\omega_2, \{\mathrm{r}_1, \mathrm{r}_2\}\right)\right\}, \mathrm{C}\left(\phi_2^2(\mu_0)\right) = 15.5 > \mathrm{C}(\mu_0)\end{aligned} \tag{3.7}$$

以上计算显示, 对于当前的分配 μ_0, 任何切换操作将增大总成本. 这满足了 *Nash* 稳定分配的定义, 即每个联盟都不能通过切换操作在不增大总成本的条件下降低自身成本. 然而如果我们同时作用两个切换操作, 则可以获得最优解 $\phi_1^1 \cdot \phi_2^2(\mu_0) = \{(\omega_1, \{r_1\}), (\omega_2, \{r_2\})\}$, 其成本为: $\mathrm{C}\left(\phi_1^1 \cdot \phi_1^2(\mu_0)\right) = 4 < \mathrm{C}(\mu_0)$. 以上分析显示出我们在 *3.2*中提出的算法并不能保证给出该简单场景下的最优解.

3.3 分布式 K-Serial 稳定联盟形成

针对我们在注解1 中讨论的 Nash 稳定解的不足, 我们将进一步拓展上一节所描述的算法, 在保证 Nash 稳定性质的基础下, 有效提升任务分配解的质量.

为了符号简便, 定义函数 $\mu(\omega_m) = \mathcal{R}_m$, 它返回任务 ω_m 的联盟, 反之,$\mu(i) = \omega_m$ 返回分配给机器人 i 的任务, $\forall i \in \mathcal{R}_m$.

定义 4 (链式变换). 一个链式变换被定义为一系列允许的切换, 即

$$\Phi \triangleq \phi_{i_1}^{m_1} \circ \phi_{i_2}^{m_2} \circ \phi_{i_\ell}^{m_\ell} \cdots \circ \phi_{i_L}^{m_L}, \tag{3.8}$$

其中 L 是总长度; $i_{\ell+1} \in \mathcal{N}_\ell$ 是形成链的约束,$\forall \ell = 0, \cdots, L-1$. 如果所有内部切换都是有效的, 则链式变换 Φ 是有效的. 因此, 通过 Φ, 可以将分配 μ 递归地转换为新分配 $\hat{\mu}$, 通过

应用 Φ 中的转换 $\phi_{i_\ell}^{m_\ell} \in \Phi$, 使得 $\hat{\mu}(i_\ell) = \omega_{m_\ell}, \forall \ell = 0, \cdots, L-1$. 为简洁起见, 用 $\hat{\mu} = \Phi(\mu)$ 表示. ■

定义 5 (根变换). 如果一个链式变换始于机器人 i 的切换操作, *i.e.* 在 (3.8) 中有 $i_1 = i$, 则称其为机器人 i 的一个根变换, 简记为 Φ_i. ■

注解 2. 上述链式变换与 *[32]* 中的“任意修改”不同. 在“任意修改”中, 变换序列可以在任何机器人上操作. 然而, 式 (3.8) 中的链式结构要求只有相邻的机器人可以连续执行切换操作. 主要动机是将可能的变换搜索空间限制在距离根节点为 K 的局部通信子网络内的机器人, 从而加快收敛速度, 且保持了较好的质量. ■

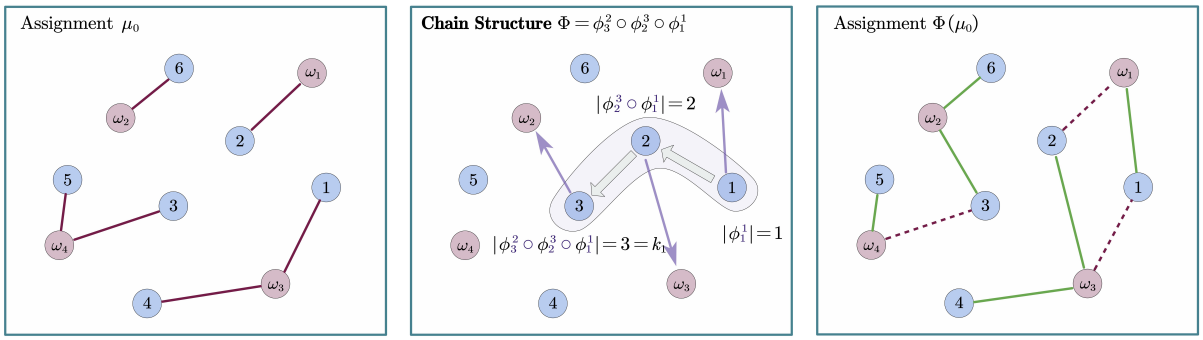


图 3.3. 原分配 (**左**), 链式变换 (**中**), 变换后的分配 (**右**)

定义 6 ($\mathcal{K}$-Serial 稳定). 如果对于每个机器人 $i \in \mathcal{R}$, 不存在任何根转换 Φ_i 满足以下条件, 则分配 $\mu^\star$ 是 $\mathcal{K}$-*Serial* 稳定 *(KSS)* 的: *(i)* Φ_i 是有效的; *(ii)* $\varrho(\Phi_i(\mu^\star)) > \varrho(\mu^\star)$; *(iii)* $|\Phi_i| \leq k_i$, 其中 k_i 是在公式(3.1)中定义的最优性指数. ■

注解 3. 上述的 *KSS* 解概念包含了文献 *[32]* 中的纳什均衡和全局最优解作为特殊情况. 也就是说, 如果选择指数使得对于所有的 $i \in \mathcal{R}$, 都有 $k_i = 1$, 那么 *KSS* 解就等价于纳什均衡, 因为最多只允许一个机器人转换其自己的任务. 如果对于所有的 $i \in \mathcal{R}$, 都有 $k_i = N$ 且通信图连通, 那么 *KSS* 解就等价于全局最优解, 因为所有机器人都可以同时转换任务. 因此, 参数 $\mathcal{K} = k_i$ 提供了适应性的灵活性, 可以根据时间预算来自适应地改进解决方案. ■

问题 3. 给定一个 (3.1)中的联盟结构 $\mathcal{F}$, 设计一个分布式算法求解定义 *6*中描述的 *KSS* 任务分配解. ■

3.3.1 算法描述

我们提出了一种完全分布式、异步的、随时可行的算法——K-Serial 稳定联盟形成算法 ($\mathcal{K}$-serial stable coalition,KS-COAL) 来解决问题3. 如图3.4所示, 其主要思想是将分布式机器人间的信息传递与本地优化相结合. 每个机器人 i 试图找到以自身为根的局部最优转换, 以最大程度地改善当前已知的最佳解. 此外, 每个机器人还参与以其他机器人为根的转

换的优化, 即作为(3.8)中定义的链的一部分. 最后, 一个共识协议在并行运行, 以促进团队内对共同 KSS 解的收敛. 特别的, 为了有效利用混合优化层来调整任务分配, 在机器人的每一次本地循环中, 我们都将把部分具有较小总代价的分配所包含的联盟发送到混合优化层, 并行运行混合优化层以计算实际代价, 并用于更新这些分配的估计代价. 在算法运行过程中, 每个机器人都将维护一个字典映射用于储存不同联盟的估计代价, 并随着信息传播而进行同步和更新. 在后文的描述中, 除非特别说明, 代价均指代估计代价.

更具体地, 如 Alg. 2所述, 每个机器人都异步地遵循相同的算法流程. 最初在 $t=0$ 时, 每个机器人通过将一个可执行任务分配给自己来计算初始部分解集. 然后, 机器人 i 向其所有邻居 $j \in \mathcal{N}_i$ 发送以下消息:

$$\pi_{i\to j} \in \{(\mu_0, \varepsilon, k_i),\ \mu_0(i) = \omega_m \in \Omega_i\}, \tag{3.9}$$

其中,μ_0 是初始解,ε 表示空变换, k_i 是机器人 i 的最优性指数. 此外, 这个消息被添加到一个本地消息缓冲区 Π_i 中, 参见算法第 1 行.

在 $t>0$ 时刻, 每个机器人 $i \in \mathcal{N}$ 都会按照算法中第 3 行至第 15 行相同的发送消息、接收消息、局部优化和共识、计算实际代价的循环. 首先, 对于任何存储的消息, 当 $\pi = (\mu, , \Phi, k) \in \Pi_i$ 且 $|\Phi| \le k_i$, 即 $|\Phi| \le k$ 时, 机器人 i 将该消息中继给它的邻居, 即 $\pi_{i\to j} = \pi, \forall j \in \mathcal{N}_i$. 同时, 在接收到每个来自机器人 j 的消息 $\pi_{j\to i}$（其中 $i \in \mathcal{N}_j$）后, 它将 $\pi_{j\to i}$ 存储在其接收消息缓冲区中. 在局部优化期间, 机器人 i 将从接收缓冲区中取出消息 $\pi = (\mu, \Phi, k) \in \Pi_i$, 通过将额外的转换操作附加到现有变换上生成一个新消息 π^+, 即:

$$\pi^+ = (\mu,\ \Phi \circ \phi_i^m,\ k), \tag{3.10}$$

其中,μ 是与 π 中相同的初始解; ϕ_i^m 将任务 ω_m 分配给机器人 i. 注意, 仅当 $|\Phi| \le k$ 时, 即长度限制未达到时才允许进行此修改. 此新消息 π^+ 存储在 Π_i 中, 稍后将与邻居共享, 参见第 9-10 行. 更重要的是, 机器人 i 维护其当前已知的最佳分配, 表示为 $\mu_i^\star$. 每次收到消息或在 (3.10) 中生成新消息时, 都会更新它, 即

$$\mu_i^\star = \Phi(\mu), \quad \text{if } \hat{C}(\Phi(\mu)) < \hat{C}(\mu_i^\star), \tag{3.11}$$

如第 11 行所示. 其中, 更新的代价 $\hat{C}(\Phi(\mu))$ 可以由机器人 i 本地计算为 $\hat{C}(\Phi(\mu)) = \hat{C}(\mu_i^\star) + \Delta_f(\omega_m) + \Delta_f(\omega_m')$, 其中 ω_m, ω_m' 分别是更新前后分配给机器人 i 的任务; $\Delta_f(\cdot)$ 是更新前后 ω_m, ω_m' 的任务估计代价之差.

正如前面提到的, 一个共识协议将被并行地执行, 使团队收敛到相同的 KSS 解决方案. 共识协议包含根节点替换和最优分配共享的机制, 避免了算法过早终止或搜索空间被过度

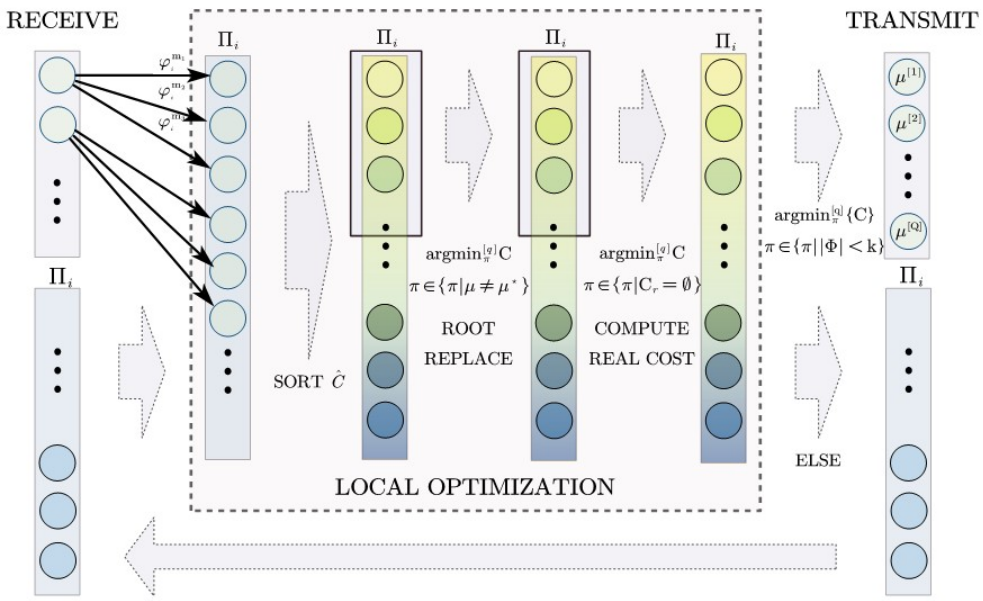


图 3.4. KS-COAL 算法示意图

约束. (I) 最优分配共享 (Optimal Assignment Sharing, OAS): 每个机器人 i 与其邻居分享其最佳解 $\mu_i^\star$, 即通过直接发送关联的消息 $\pi_i^\star$. 然后机器人 i 可以在第12行中本地计算从其他机器人收到的最佳分配 $C(\mu_j^\star)$, 并更新自身储存的最佳分配. 随后, 如果任何接收到的分配提供更低的整体成本, 即 $C(\mu_j^\star) < C(\mu_i^\star)$, 则机器人 i 将更新其最佳分配. (II) 根节点替换 (Root Assignment Replacement, RAR): 我们将把所有存储在 Π_i 中的消息 $\pi = (\nu, \Phi_i, k_i)$ 的根节点 ν 替换为当前已知的最优分配 $\nu^\star$, 即(3.12)式中的转换.

$$(\nu,\, \Phi_i,\, k_i) \to (\nu_i^\star,\, \Phi_i,\, k_i), \tag{3.12}$$

相应地, 将根变换 Φ_i 应用于 $\mu_i^\star$ 以生成新的解 $\Phi_i(\mu_i^\star)$, 其代价可以通过回溯 $\Phi_i = \phi_{i_1}^{m_1} \circ \phi_{i_2}^{m_2} \cdots \circ \phi_{i_L}^{m_L}$ 中的链操作, 并直接查询沿着链的每个机器人 i_ℓ 的变化来计算. 如果 $\Phi_i(\mu_i^\star)$ 提供了比 $\mu_i^\star$ 更低的代价, 则用它替换 $\mu_i^\star$ 并与邻居分享, 参见第15行.

完成局部优化后, 我们将对当前本地消息缓冲区 Π_i 的消息更新实际代价, 具体的, 我们检查消息所包含的联盟是否被计算过真实代价, 这通常可以由一个附加的布尔类型标志位来记录. 对于未计算的联盟, 机器人将调用混合优化层, 并行地计算多个联盟的最优决策和对应的实际成本并返回到上层. 机器人将更新自身储存的估计代价字典映射, 并以此重新计算本地消息缓冲区中所有消息的估计代价.

特别地, 如图3.4中所描述的, 我们在实际使用该算法时, 为了加快获得高质量分配的速度, 我们采用一种朴素的启发式策略, 即在每轮循环中, 在完成消息接收、附加切换操作完成后, 我们将对本地消息缓冲区中的消息按照估计代价从小到大进行排序, 然后在所有根节点不等于当前最优节点的消息集合中, 选择代价最小的 Q 个进行 Root Replace 操作, 其他消息保持不变, 从而搜索具有较大潜力的分配. 然后, 我们也将在所有包含未计算过实际代

价的联盟的消息集合中, 选择代价最小的 Q 个进行混合优化, 计算其内部联盟的实际收益并更新估计代价字典映射. 同理, 发送消息时, 我们也将只发送所有满足 $|\Phi| < k$ 的消息集合中具有前 Q 小代价的消息, 避免因为消息数目随迭代轮次指数上升而导致无法在有限的时间周期内完成局部优化. 值得一提的是, 尽管 Min-Q 策略将带来本地消息缓冲区中根节点没有实时同步的问题, 但有效改善了算法的运算效率, 以更小的时间成本获得了同样质量的任务分配解. 容易证明的是, Min-Q 版本的算法仍然可以保证获得 KSS 解.

与我们在定理中证明关于 NS-COAL 算法的性质类似, 如果估计代价函数始终小于等于实际代价, 那么我们可以进一步减少在每一次迭代中必需的实际代价计算次数. 具体地, 对于 KS-COAL 算法而言, 我们只需要在每一次消息收发的迭代过程中计算当前所储存的最优分配 $\mu^\star$ 的实际代价, 对于本地消息缓冲区中的其他分配我们均按照估计代价进行排序和更新, 而这样的操作不会对算法的收敛性和解的质量保证造成任何影响, 这将在推论中得到进一步讨论.

完成以上步骤后, 算法将检查本地最佳分配 $\mu_i^\star$ 是否已经维持了一定的轮数而未发生改变. 如果是, 算法将终止.

3.3.2 性能分析

对于我们提出的算法, 以下定理保证了算法收敛到一个 KSS 解. 此外, 如果将 $\mathcal{K}$ 设置为相应值, 则表明该解自然地扩展为全局最优解.

定理 3.3.1. *对任意 i, 智能体 i 所储存的最优方案的代价 C_i^* 随着 t 增加单调递减并收敛到一个常数 C_i, 同时 $\mu^\star$ 收敛到 μ_i, 且存在 $t_N \in R$, 使得 $t > t_N$ 时, $C_n^\star$=C_n.*

证明. 根据我们算法的最优分配更新机制, 智能体 i 一旦发现一个新的 assignment μ 满足 $C(\mu) < C_n^\star$, 将用 μ 代替 μ_n^*, 并将 $U(\mu)$ 赋值给 $C_n^\star$. 因此在算法运行的过程中, 智能体 i 储存的 $C^\star$ 单调递增. 设算法的最优解为 μ^*, 则有 $C^\star \leq C(\mu^*)$, 因此 $C_n^\star(t)$ 有上界. 由于单调有界函数必收敛, 因此 $C^\star(t)$ 收敛, 收敛值为 c_n. 同时, 由于分配仅有有限种离散取值, 因此 $C^\star$ 的取值集合的大小也为有限值, 因此存在 t_N 使得 $t > t_N$ 时,$C_n^\star(t) = C_i$, 且由于一一对应关系,$\mu^\star$ 收敛到 d_i. □

定理 3.3.2. *对任意智能体 i, j, 当两者均收敛时有 $\hat{C}_i = \hat{C}_j, \mu_i = \mu_j$, 即所有智能体关于最优分配达成共识.*

证明. 假设结论不成立. 对于一组收敛解 $(\mu_1, \mu_2, ..., \mu_N)$, 取 $i = \mathbf{argmin} C_i$. 由假设条件, 存在 j, 使得 $C_i < C_j$. 设 $C^\star = C(\nu_i), C_j = C(\nu_j)$. 由于我们算法的最优分配共享步骤, 每个智能体都会将自身的最优分配和邻居共享. 对于连通的通讯拓扑, 对于图中任意智能体

Algorithm 2: KS-COAL algorithm.

Input: Coalition structure $\mathcal{F}$.

Output: KSS solution $\mu^\star$.

/* Message passing */

Initialize Π_i by (3.9), $\Pi_i^\star = \emptyset$;

while *not terminated* **do**

  /* Sending message */

  for $\pi = (\mu, \Phi, k) \in \Pi_i$ *and* $|\Phi| \leq k$ **do**

    Send $\pi_{i\to j}$ to $\mathcal{N}_i$;

    Remove $\pi_{i\to j}$ from Π_i;

  /* Receiving message */

  forall $i \in \mathcal{R}$ **do**

    Receive $\{\pi_{j\to i}\}$ and update Π_i;

  /* Local optimization */

  for $\pi = (\mu, \Phi, k) \in \Pi_i$ **do**

    Compute ϕ_i^m and π^+ by (3.10);

    Save π^+ to Π_i;

    Update $\mu_i^\star$ by (3.11);

  /* Consensus */

  if $\hat{C}(\mu_j^\star) < \hat{C}(\mu_i^\star)$ **then**

    /* OAS */

    $\mu_i^\star \leftarrow \mu_j^\star$;

    /* RAR */

    Compute $\{\Phi_i(\mu_i^\star)\}$ and update Π_i by (3.12);

    Update $\mu_i^\star$ by (3.11);

i, j, 都存在一条长度为 L_j 的路径从 i 到 j, 即至多经过 L_j 次消息传递, z 最优分配将从 i 传递到 j, 此后 j 的最大代价将小于 $C^\star$ 因此在至多 $\mathbf{max}L_j$ 次消息传递后, 可以保证所有智能体的最小代价大于等于 C_i*. 这与 $(\mu_1, \mu_2, ..., \mu_N)$ 为收敛解矛盾. 因此假设不成立, 证毕. □

定理 3.3.3. *在 Alg.2下, 局部分配被保证收敛到 K-Serial 稳定解,*

证明. 对于每个机器人 i, 所有满足长度限制条件 $|\Phi_i| \leq k_i$ 的根变换 Φ_i, 将在算法收敛之前在机器人 i 的 k_i-hop 通信网络内被生成. 同时, 由于定理3.3.2, 算法收敛后, 所有智能体共享同一个最优分配 μ^*. 这意味着, 上述根变换同时作用于一个相同的分配, 并且所有作用后产生的新的分配的代价 C 大于位于该最优分配, 否则产生的新分配将替代该分配成为稳定后的最优分配. 换句话说,$\mu_i^\star$ 不能被任何 $|\Phi_i| \leq k_i$ 的根变换所改善. 根据 Def.6,$\mu^\star$ 是 KSS

解. □

推论 1. *如果在算法中始终对所有消息缓冲区中的消息计算实际代价, 则收敛后的局部分配是在实际代价意义上 K-Serial 稳定解, 相比于估计代价意义下的 K-Serial 稳定解, 该解将使得 CHO 问题具有更优的目标值.*

证明. 根据 K-Serial 稳定的定义, 在估计意义下的 K-Serial 稳定解不一定是在实际意义下的 K-Serial 稳定解, 因此其对应的实际代价大于等于实际意义下的 K-Serial 稳定解, 而 CHO 问题中的目标函数是按照各混合计划的实际代价计算的. □

推论 2. *如果估计代价函数始终小于实际代价函数, KS-COAL 每次迭代可以只计算当前处理的最优分配的实际代价, 而不影响解的 KSS 性质.*

证明. 和定理3.3.1的证明类似, 在上述前提条件下, 我们假设算法的收敛解 μ^* 不具有 KSS 性质, 则意味着存在一个根变换 Φ^+ 可以降低该收敛解的代价. 然而, 由于估计代价函数小于实际代价函数, 因此有根变换作用后的分配的估计代价小于收敛解的实际代价, 即 $\widehat{C}(\Phi^+(\mu^*)) \leq C(\Phi^+(\mu^*)) \leq C(\mu^*)$. 而在每次迭代中, 我们都保证最优分配的实际代价被计算, 因此根据算法的根节点替换步骤, 算法的最优分配将被更新为 $\Phi^+(\mu^*)$, 这与 μ^* 为收敛解矛盾. □

推论 3. *假设存在一个智能体 $i \in \mathcal{R}$ 满足其 K 取值为 $k_i = N$, 则 KS-COAL 返回的 KSS 解就是全局最优解.*

证明. 假设 $\mu^\star \neq \mu^\dagger$, 其中 $\mu^\dagger$ 是最优分配. 由于通信图是连通的, 因此一定存在一个长度为 N 的链式变换 Φ 来生成 $\mu^\dagger$. 由于 $\mu^\star$ 是 KSS, 也就是说 $\varrho(\mu^\dagger) \leq \varrho(\mu^\star)$, 这导致了矛盾. 因此,$\mu^\star = \mu^\dagger$ 成立. □

3.3.3 图神经网络加速规划

在上一节中, 我们提出了 KS-COAL 算法并证明了其收敛性和质量保证. 容易发现, 参数 $\mathcal{K}$ 显著影响所提出的协调算法的复杂性和效率. 随着 $\mathcal{K}$ 的增加, 消息数量呈指数增长, 尽管我们采用了 Max-Q 机制缓解了该问题, 但为了得到较高质量的 KSS 解, 大量的消息传递仍然是难以避免的. 此外, 我们发现, 在实际的算法运行过程中, 其中只有少数消息对整体效用的改进做出了贡献. 为了克服这些限制, 提出了一种基于图神经网络 (Graph Neural Networks,GNNs) 的神经加速器 (NAC) 来预测 Alg. 2中索引 $\mathcal{K}$ 的选择和良好的初始分配.

如图 3.5所示, 潜在的联盟结构被编码为一个带有顶点和边属性的图, 作为 GNN 的输入. 机器人和不同的任务, 即动态捕获和资源防御被建模为不同类型的顶点. 机器人到任务

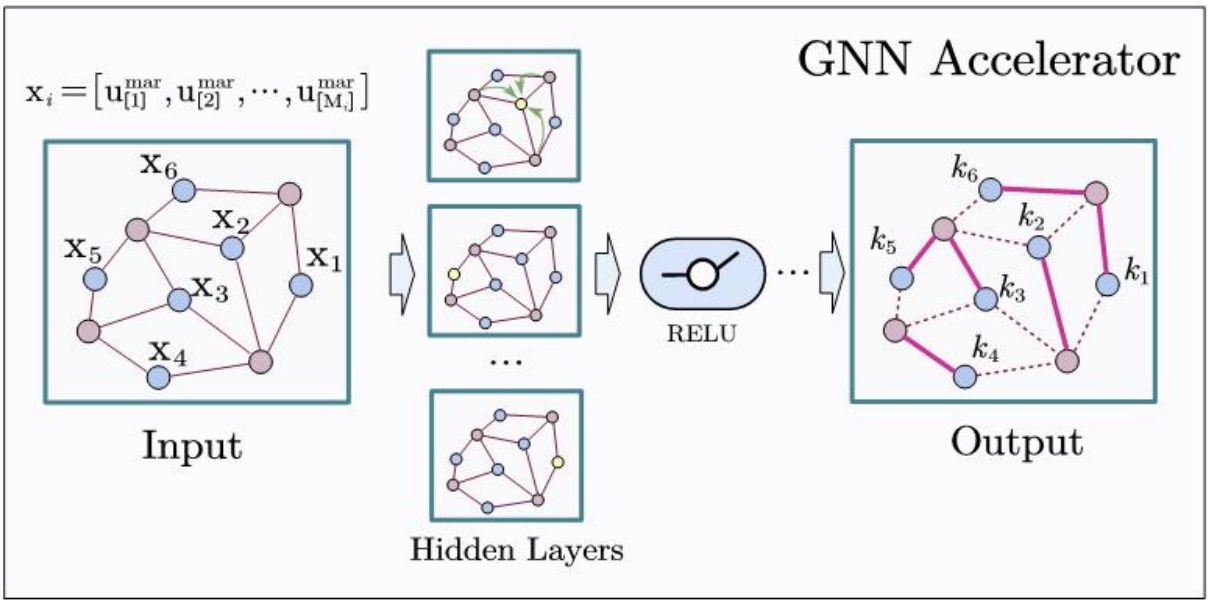


图 3.5. 图神经网络结构示意图

的无向边表示一个机器人是否可以参与一个任务. 此外, 每条边的属性是定义如下的边际效用的向量:

$$u(i, \omega_m) = \left\{ f(\mathcal{R}_m, \omega_m) - f(\mathcal{R}_m \backslash \{i\}, \omega_m), \forall \mathcal{R}_m \in \widehat{\mathcal{R}}_m \right\}, \tag{3.13}$$

其中 $i \in \mathcal{R}_m$ 是可以执行任务 $\omega_m \in \Omega$ 的任意联盟; $\widehat{\mathcal{R}}_m \subset 2^{\mathcal{R}}$ 是所有这样的联盟的集合; $u \in \mathbb{R}^{|\widehat{\mathcal{R}}_m|} \geq 0$ 是非负实数向量. 同时, 我们将边的属性聚合到 Agent 节点上, 从而得到每个 Agent 的嵌入向量表示:

$$\mathbf{x}_i = [u_{[1]}^{\mathtt{mar}}, u_{[2]}^{\mathtt{mar}}, \cdots, u_{[M_i]}^{\mathtt{mar}}] \tag{3.14}$$

请注意, 属性的维度可能会随着图的变化而变化. 上述图结构将由基于 GNN 的异构注意力图模型 (heterogeneous attention graph model, HAN) 处理, 该模型是基于 [33] 中针对具有不同大小的异构图量身定制的. 最后, 输出包括对机器人顶点上预测的索引 $\mathcal{K}$ 的属性和对每个机器人的所有出边进行排名的结果.

3.3.4 训练和执行

训练上述 NAC 所需的数据是通过使用 Alg.2 解决多个问题3实例得到的, 以获得在不同的 $\mathcal{K}$ 选择下的 KSS 解决方案. 用 $\mathcal{D} = (\mathcal{F}, \nu^\star)$ 表示这样的数据集. 与每个 $\mathcal{F}$ 相关的图可以编码为 Sec.3.3.3中描述的那样. 此外, 输出是每个 $\mathcal{F}$ 的最佳 $\mathcal{K}$ 选择以及给定最终分配 $\nu^\star$ 的每个边 (i, ω_m) 上的 0, 1 标签. 给定这些训练数据, 提出的 NAC 可以通过使用 *F1* 损失以监督方式进行训练, 以避免不平衡的数据. 现代 GNN 遵循邻域聚合范式, 通过聚合多个跳内的边来迭代更新边表示. 因此, 经过训练的 NAC 可以预测最佳的 $\mathcal{K}$ 选择, 以及推荐的初始解 ν_0.

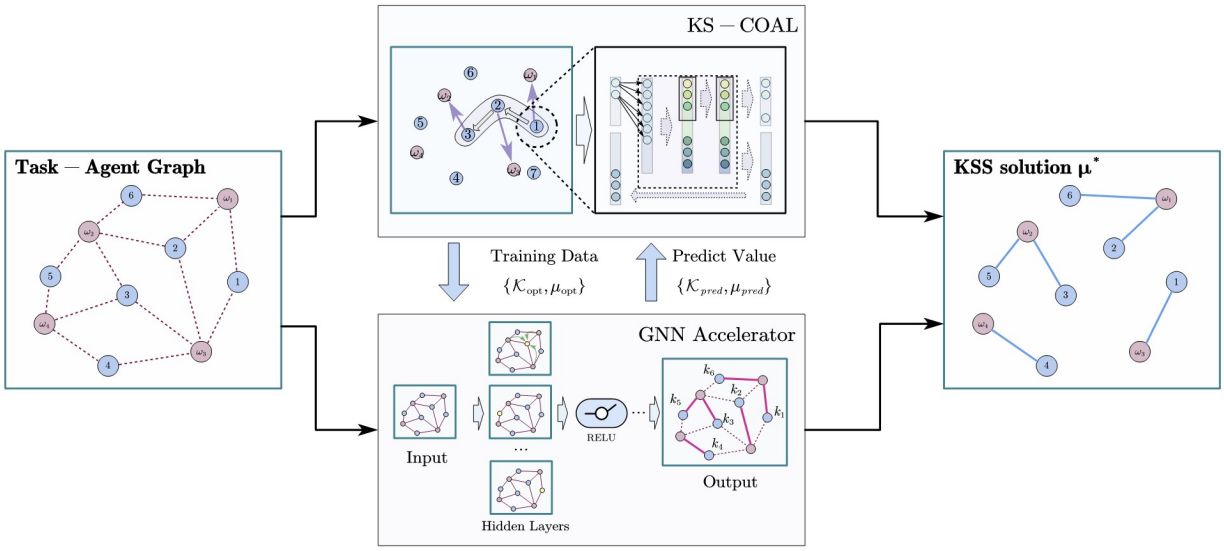


图 3.6. GNN-accelerator 与 KS-COAL 联合工作框架

在线执行过程中, 任务和机器人的数量不断变化. 相应的联盟 $\mathcal{F}$ 会定期更新, 其编码图被派生出来. 通过信息聚合, 可以以分布式的方式推断出最佳的 $\mathcal{K}$ 选择和推荐的解决方案 ν_0. 因此, 可以使用这种选择的 $\mathcal{K}$ 和特定的初始分配 ν_0 启动 Alg.2来开始通信. 这样, 在协调过程的早期就可以获得高质量的分配. 请注意, 定理3.3.3 和引理 3 中证明的质量保证得以保留.

3.4 混合优化

如前所述, 将 $\mathcal{R}_m$ 中的代理组作为联盟应遵循混合规划 φ_m, 如公式 (2.2) 所定义, 以完成任务 ω_m. 本节介绍如何通过所提出的混合优化找到这样的混合规划.

3.4.1 混合优化问题

为简化表示, 令 $\mathbf{X} = x_{k_0^m} \cdots x_{k_T^m}$ 为离散时间步长 $t = 0, 1, \cdots, T$ 下的系统状态序列, 其中 T 为足够长的持续时间; $\Xi = \xi_{k_{t_1}^m}, \cdots, \xi_{k_{t_N}^m}$ 和 $\mathbf{P} = \rho_{k_{t_1}^m}, \cdots, \rho_{k_{t_N}^m}$ 为应用于系统的模式序列, 时间段为 $[t_0, t_1), [t_1, t_2), \cdots, [t_{N-1}, t_N)$, 其中 $t_n = nT_0, T_0 \in \mathbb{N}, \forall n = 0, \cdots, \lfloor \frac{T}{T_0} \rfloor$. $T_0 > 0$ 被选择为每个模式的持续时间下限, 以避免模式和参数过于频繁的切换. 此外, 令 $\mathbf{X}_t, \Xi_t, \mathbf{P}_t$ 分别表示 $x_{k_t^m}, \xi_{k_t^m}, \rho_{k_t^m}$. 那么混合优化问题可以表示为:

问题 4. 给定任务 ω_m 和相关联盟 $\mathcal{R}_m$, 找到最优序列 $(\mathbf{X}, \Xi, \mathbf{P})$, 解决下面的混合优化问题:

$$\begin{aligned} \min_{\Xi,\mathbf{P}} & \sum_{t=0}^{T} c_{\text{cont}}(\mathbf{X}_t,\, \mathbf{P}_t) \\ \textbf{s.t.} \quad & \mathbf{X}_0 = x_0,\ \mathbf{X}_T \in \mathcal{X}_{G_m}; \\ & \xi_{k_t^m} = \Xi_{t_n},\ \rho_{k_t^m} = \mathbf{P}_{t_n},\ \forall t \in [t_n, t_{n+1}); \\ & \mathbf{X}_{t+1} = h_{\xi_{k_t^m}}(\mathbf{X}_t,\, \mathcal{R}_m,\, \mathbf{P}_t),\ \forall t \in [0,\, T]; \end{aligned} \tag{3.15}$$

其中,$c_{\text{cont}}(\cdot)$ 是一个一般的函数, 包括控制成本和平滑度; $h_{\xi_{k_t^m}}(\cdot)$ 是每种模式下的系统动力学方程, 来自于公式 (2.1); 约束条件要求在 $[t_n, t_{n+1})$ 内保持模式和参数静态不变. ■

需要注意的是, 与式 (2.5) 中的原始组合混合优化问题不同, 上述混合优化问题的目标是找到特定任务 ω_m 的最优混合计划 φ_m 和相应的联盟 $\mathcal{R}_m$. 提出了一种称为启发式梯度引导混合搜索 (HGG-HS) 的新型混合搜索算法来解决问题 4. 该算法不是直接将上述问题输入到一般的非线性优化器中, 而是将以下两个步骤组合起来: (i) 基于 $A^\star$ 的离散搜索来获得最优模式序列, 以及 (ii) 基于梯度的优化来获得最优参数序列. 更具体地说, 混合搜索树定义为 $\mathcal{T} \triangleq (V, E, \nu_0, V_G)$, 其中 $V = \nu$ 是一组顶点,$\nu \in \mathcal{X}_{\texttt{safe}}$;$E \subset V \times V$ 是一组边; $\nu_0 \in V$ 是初始顶点;$V_G \subset V$ 是由 $\mathcal{X}_{G_m}$ 确定的目标状态集合. 此外, 设 $cost(\nu)$ 是节点 ν 的成本,$prev(\nu)$ 是 ν 的父节点.

3.4.2 启发式函数的设计

正如 [34] 所述, 启发式函数 $h : \mathcal{X} \to \mathbb{R}^+$ 的合理设计对 $A^\star$ 搜索算法的性能至关重要. 由于找到一个完美估算从给定顶点 $\nu \in V$ 到目标集合 V_G 的成本的精确启发式函数 $h^{\texttt{opt}}$ 是不切实际的, 因此本工作提出了两个不同抽象级别的精确启发式函数的近似: (i) 全局近似 $h^{\mathcal{G}}$ 作为实际成本的下界, 即 $h^{\texttt{opt}}(x) \geq h^{\mathcal{G}}(x), \forall x \in \mathcal{X}$. 例如, 欧几里得距离是一个常见的可接受启发式函数; (ii) 可微分的局部近似 $h^{\mathcal{L}}$ 具有与 $h^{\texttt{opt}}$ 在局部邻域内类似梯度的性质, 即:

$$|\nabla h^{\mathcal{L}}_{x_0}(x) - \nabla h^{\texttt{opt}}(x)| \leq E,\ \forall x \in \mathrm{U}_d(x_0) = \{x \in \mathcal{X} |\ |x - x_0| \leq d\}, \tag{3.16}$$

其中 $E > 0$. 此外, 平衡启发式函数 $h^{\mathcal{B}}$ 的递归定义如下:

$$h^{\mathcal{B}}(\nu) = \lambda\Big(h^{\mathcal{B}}(\tilde{\nu}) + \Delta h^{\mathcal{L}}(\tilde{\nu},\, \nu)\Big) + (1 - \lambda)\, h^{\mathcal{G}}(\nu), \tag{3.17}$$

其中 $\tilde{\nu} = prev(\nu)$; $\lambda \in [0, 1]$ 是一个权重因子; $\Delta h(\tilde{\nu}, \nu)$ 是从 $\tilde{\nu}$ 到 ν 成本的变化量, 它的估算值是沿着路径累计变化量, 即

$$\Delta h^{\mathcal{L}}(\tilde{\nu},\, \nu) = \sum_{\ell=1}^{L} \Big(h^{\mathcal{L}}_{x_\ell}(x_\ell) - h^{\mathcal{L}}_{x_\ell}(x_{\ell-1})\Big), \tag{3.18}$$

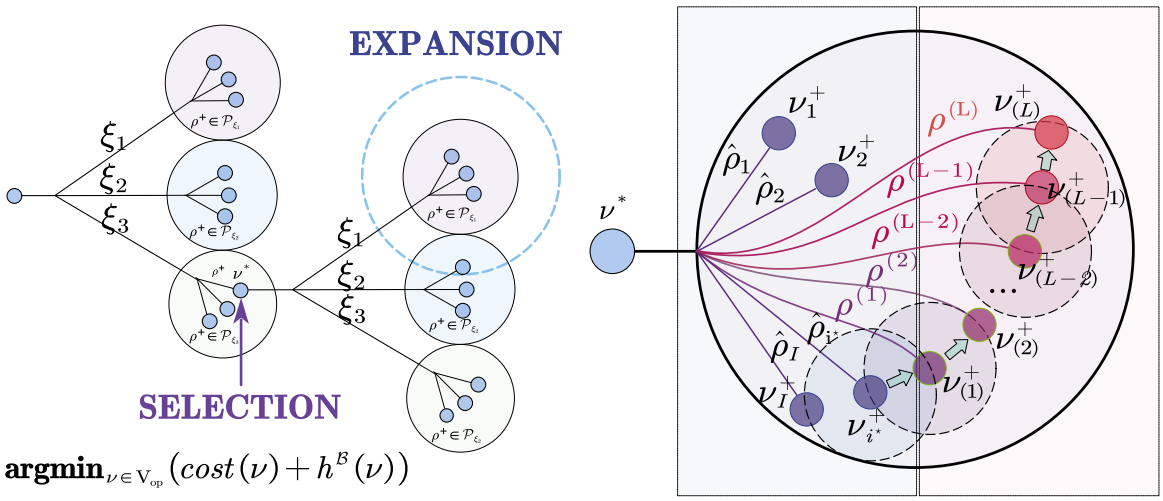


图 3.7. 3.4.3中描述的混合搜索算法: 选择和拓展 (**Left**); 迭代参数优化 (**Right**).

其中 $x_0 = \tilde{\nu}, x_L = \nu$, 且 $x_\ell \in \mathrm{U}_d(x_{\ell-1}), \forall \ell = 1, \cdots$

此外,λ 是影响 HGG-HS 贪心程度的参数. 当 $\lambda = 0$ 时,$h^{\mathcal{B}}$ 等同于 $h^{\mathcal{G}}$, 从而产生类似 $A^\star$ 的通用启发式搜索算法. 另一方面, 当 $\lambda = 1$ 时,$h^{\mathcal{B}}$ 仅依赖于局部近似 $h^{\mathcal{L}}$, 从而产生局部贪心搜索.

3.4.3 启发式梯度引导混合搜索

考虑到上面定义的平衡启发式方法, 混合搜索树 $\mathcal{T}$ 通过节点选择和扩展的迭代过程进行探索. 首先, 类似于 $A^\star$ 算法, 使用优先队列 $V_{\mathtt{op}} \subset V$ 存储要访问的顶点, 同时使用集合 $V_{\mathtt{cl}} \subset V$ 存储已经完全探索的顶点. 如图 3.7 所示, 提出的混合搜索算法包括以下阶段: (I) **选择**. 在 $V_{\mathtt{op}}$ 中选择预估成本最低的顶点, 即 $\nu^\star = \mathbf{argmin}_{\nu \in V_{\mathtt{op}}} cost(\nu) + h^{\mathcal{B}}(\nu)$, 其关联的状态为 $x^\star$. (II) **扩展**. 该顶点 $\nu^\star$ 经过以下三个步骤进行扩展: (i) 首先, 给定状态 $x^\star$, 选择可行模式 $\xi \in \Xi$; (ii) 然后, 在参数空间中通过迭代优化寻找模式 ξ 和 $x^\star$ 的一组候选参数集 $\{\rho^\star\} \subset \mathcal{P}_\xi$, 如下式所述; (iii) 最后, 给定上述 ξ 和 $\rho^\star$, 得到一组结果子顶点集合 $\mathcal{C}_\xi(\nu^\star) \triangleq \{\nu^+\}$, 其表示如下:

$$\begin{aligned} \mathbf{x}_{t+1} &= h_\xi(\mathbf{x}_t, \mathcal{R}_m, \rho^\star); \forall t \in [0, T_0]; \\ x_0 &= x^\star, \nu^+ = \mathbf{x}_{T_0}; \end{aligned} \tag{3.19}$$

其中 $\nu^+ \triangleq \mathtt{Expand}(x^\star, \rho^\star)$ 表示为简化符号而封装. 此外, 节点 ν^+ 的属性通过 $cost(\nu^+) = cost(\nu^\star) + \sum_{t=0}^{T_0} c_{\mathrm{cont}}(\mathbf{x}_t, \rho^\star)$ 和 $prev(\nu^+) = \nu^\star$ 进行更新. 对于 ν^+, 如果对所有满足 $[(\nu^+ - \nu)/\delta] = \mathbf{0}$ 的节点 $\nu \in V$, 都有 $cost(\nu^+) \leq cost(\nu)$ 且 $\nu \notin V_{\mathtt{cl}}$, i.e. 满足下列条件:

$$cost(\nu^+) \leq cost(\nu),\ \nu \notin V_{\mathtt{cl}}, \quad \forall \nu \in \{\nu \in V \| \ [(\nu^+ - \nu)/\delta] = \mathbf{0}\} \tag{3.20}$$

那么就把子节点 $\nu^+ \in \mathcal{C}_\xi(\nu^\star)$ 加入节点集合 V 和 $V_{\texttt{op}}$. 其中,$[\cdot]$ 是取整函数. 随后, 将边 $(\nu^\star, \nu^+)$ 加入边集合 E 并标记为相应的模式和参数 $(\xi, \rho^\star)$. (III) **终止**. 如果 $\nu^\star$ 的所有子节点都已被探索, 则将 $\nu^\star$ 从 $V_{\texttt{op}}$ 中移除并加入 $V_{\texttt{cl}}$ 中. 更重要的是, 如果 $\nu^\star \in V_G$, 则可以通过追溯父节点并检索其标签 (ξ, ρ), 得到解决问题4的最优序列 Ξ 和 $\mathbf{P}$. 因此, 混合搜索算法将以(3.15)中描述的混合规划的形式返回, 并附带指派任务的实际代价.

Algorithm 3: HGG-HS Algorithm.

Input: Task ω_m, Coalition $\mathcal{R}_m$.

Output: Hybrid plan $\varphi_m^\star$.

Initialize $\mathcal{T} = (\{\nu_0\}, \emptyset, \nu_0), V_{\texttt{op}} = \{\nu_0\}, V_{\texttt{cl}} = \emptyset$;
while *not terminated* **do**
  /* Select node */
  $\nu^\star = \mathbf{argmin}_{\nu \in V_{\texttt{op}}} \{cost(\nu) + h^{\mathcal{B}}(\nu)\}$;
  /* Node expansion */
  forall $\xi^+ \in \Xi_m$ **do**
    forall $\hat{\rho}_i \in \mathcal{P}_{\xi^+}$ **do**
      $\nu_i^+ = \mathbf{Expand}(x^\star, \hat{\rho}_i)$
    $\nu_{i^\star}^+ = \mathbf{argmin}_{\nu \in \{\nu_{s_i}^+\}}(cost(\nu) + h^{\mathcal{B}}(\nu))$
    $\{(\rho^{(\ell)}, x_{\texttt{e}}^{(\ell)})\} = \mathbf{IterativeOptimization}(x_e^{(0)})$
    forall $\ell \in 1, \cdots, L$ **do**
      $\nu_{(\ell)}^+ = \mathbf{Expand}(x^\star, \rho^{(\ell)})$
    $\mathcal{C}_{\xi^+}(\nu^\star) = \{\nu_i^+, \forall i\} \cup \{\nu_{(\ell)}^+, \forall \ell\}$;
  $\mathcal{C}(\nu^\star) = \bigcup_{\xi^+ \in \Xi} \mathcal{C}_{\xi^+}(\nu)$
  forall $\nu^+ \in \mathcal{C}(\nu^\star)$ **do**
    if *condition* (3.20) *is true* **then**
      $V_{op} = V_{op} \cup \nu^+$
  /* Termination */
  if $\nu^\star \in \mathcal{X}_{G_m}$ **then**
    calculate and return the hybrid plan $\varphi_m^\star$

3.4.4 迭代优化模态参数

在上述扩展阶段中, 针对节点 $\nu^\star$ 中参数 $\rho^\star$ 的优化遵循一个两阶段的过程. 在第一阶段的原始扩展中, 参数最初从预定义的原始参数集合 $\hat{\rho}_1, \cdots, \hat{\rho}_I \subset \mathcal{P}_\xi$ 中选择. 然后, 可以通过 $\nu_i^+ = \texttt{Expand}(x^\star, \hat{\rho}_i)$ 生成一组子节点, 其中 i 取遍 $\hat{\mathcal{P}}_\xi$ 中的所有 $\hat{\rho}_i$. 在该集合中, 选择预估的总成本最低的子节点, 即

$$\nu_{i^\star}^+ = \mathbf{argmin}_{\nu \in \{\nu_i^+\}} \Big\{ cost(\nu) + h^{\mathcal{L}}(\nu^\star, \nu) \Big\} \tag{3.21}$$

并且关联的参数为 $\hat{\rho}_{i^\star}$. 在第二阶段的迭代优化中, 通过非线性优化来优化终态 $x_{\mathsf{e}}^{(\ell)}$ 和关联的参数 $\rho^{(\ell)}$, 其中 $\ell = 1, \cdots, L$ 表示迭代次数. 初始时,$x_{\mathsf{e}}^{(0)} = \nu_{i^\star}$ 和 $\rho^{(0)} = \hat{\rho}_{i^\star}$. 然后, 将以下过程应用于更新 $\rho^{(\ell)}$:

$$\begin{aligned} \rho^{(\ell+1)} = \underset{\rho \in \mathcal{P}_\xi}{\mathbf{argmin}} \Big\{ \sum_{t=0}^{T_0} c_{\text{cont}}(\mathbf{x}_t) + h^{\mathcal{L}}_{x_{\mathsf{e}}^{(\ell)}}(\mathbf{x}_{T_0}) \Big\}; \\ \mathbf{s.t.} \quad \mathbf{x}_0 = \nu^\star,\ \mathbf{x}_{T_0} \in \mathrm{U}_d(x_{\mathsf{e}}^{(\ell)}) \text{ in } (3.19), \end{aligned} \tag{3.22}$$

这个过程可以通过一般的非线性优化求解器 (例如 IPOPT [35]) 来解决, 因为所有状态都是通过参数 ρ 进行参数化的. 一旦得到 $\rho^{(\ell+1)}$, 则通过 $x_{\mathsf{e}}^{(\ell+1)} = \texttt{Expand}(\nu^\star, \rho^{(\ell+1)})$ 来更新相应的终止状态. 这个迭代过程会一直进行, 直到迭代 L 满足 $|x_{\mathsf{e}}^{(L)} - x_{\mathsf{e}}^{(L-1)}| < d$. 因此, 参数和终止状态对的集合由 $(\rho^{(\ell)}, x_{\mathsf{e}}^{(\ell)})$ 给出, 相应的子节点是 $\nu^+(\ell)$. 因此, 所有子节点 $\mathcal{C}_\xi(\nu^\star) = \{\nu_i^+, \forall i\} \cup \{\nu_{(\ell)}^+, \forall \ell\}$ 被发送到扩展阶段的下一步.

定理 3.4.1. *在混合搜索算法下, 对于搜索树中 $\mathcal{T}$ 中长度为 K 的路径 $\nu_0, \cdots, \nu_K$, 如果 λ 满足下列条件:*

$$\lambda_k \leq \frac{h_k^{\mathcal{G}}}{EJ/\epsilon + \Delta c_m + h_k^{\mathcal{G}}} \tag{3.23}$$

则对于任意的 $k = 1, \cdots, K$ 和参数 $\epsilon \geq 0$, 均有 $h_{k+1}^{\mathcal{B}} \leq (1+\epsilon) h_{k+1}^{opt}$. 进一步地, 如果对于搜索树中的所有路径, 各节点 λ 值均满足上述条件, 则混合搜索算法所求得的解的代价不会超过最优解代价的 $(1+\epsilon)$ 倍.

证明. 为简便起见, 设 $h_k^{\mathcal{B}}, h_k^{\mathcal{G}}, h_k^{\mathtt{opt}}$ 分别表示公式(3.17)中顶点 ν_k 的各启发式, $c_k = cost(\nu_k)$,$\Delta h_k^{\mathcal{L}} = \Delta h^{\mathcal{L}}(\nu_k, \nu_{k+1})$. 首先, 基于我们对启发式 $h^{\mathcal{G}}$ 的设定,$h_0^{\mathcal{B}} = h_0^{\mathcal{G}} \leq (1+\epsilon) h_0^{\mathtt{opt}}$ 成立. 假设对于任意的 k,$h_k^{\mathcal{B}} \leq (1+\epsilon) h_k^{\mathtt{opt}}$ 均满足. 那么,$\Delta h_k^{\mathcal{L}}$ 可以被限制如下:

$$\begin{aligned} \Delta h_k^{\mathcal{L}} &= \sum_{\ell=0}^{L-1} (h^{\mathcal{L}}_{x_{\mathsf{e}}^{(\ell)}}(x^{(\ell)}) - h^{\mathcal{L}}_{x_{\mathsf{e}}^{(\ell)}}(x^{(\ell-1)})) \\ &\leq h_{k+1}^{\mathtt{opt}} - h_k^{\mathtt{opt}} + EJ, \end{aligned}$$

那么, 其中 $|e(x)| = |\nabla h^{\mathcal{L}}_{x_0}(x) - \nabla h {x_0}^{\mathtt{opt}}(x)| \leq E$; J 是一个顶点和它的父顶点之间的最大距离. 那么, 可以推导出:

$$\begin{aligned} h_{k+1}^{\mathcal{B}} &= \lambda(h_k^{\mathcal{B}} + \Delta h_k^{\mathcal{L}}) + (1-\lambda)(h_{k+1}^{\mathcal{G}}) \\ &\leq \lambda\big((1+\epsilon) h_k^{\mathtt{opt}} + \Delta h_k^{\mathcal{L}}\big) + (1-\lambda)\, h_{k+1}^{\mathtt{opt}} \\ &\leq (1+\lambda\epsilon) h_{k+1}^{\mathtt{opt}} + \lambda\epsilon \Delta c_m + \lambda EJ, \end{aligned} \tag{3.24}$$

其中 $h_k^{\mathtt{opt}} < h_{k+1}^{\mathtt{opt}} + \Delta c$ 且 $\Delta c = c_{k+1} - c_k < c_m$. 如果关于 λ 的如下不等式成立:

$$\lambda \leq \frac{h_{k+1}^{\mathtt{opt}}}{EJd/\epsilon + \Delta c_m + h_{k+1}^{\mathtt{opt}}} \tag{3.25}$$

结合式 (3.24), 则有 $h^{\mathcal{B}}_{k+1} \leq (1+\epsilon)h^{\mathtt{opt}}_{k+1}$. 同时, 由于 $h^{\mathtt{opt}} > h^{\mathcal{G}}$, 上述不等式可以进一步放缩为 $\lambda \leq \frac{h^{\mathcal{G}}_{k+1}}{EJd/\epsilon+\Delta c_m+h^{\mathcal{G}}_{k+1}}$, 即式 3.23. 因此, 通过对 k 的归纳, 得到 $h^{\mathcal{B}}_{K} \leq (1+\epsilon)h^{\mathtt{opt}}_{K}$, 这确保了有一个有界的次优解 [34]. □

3.5 整体框架

如前文所描述的, 在本章中介绍的任务分配和混合优化两个层次以交错和并发的方式执行. 更具体地说, 联盟形成层寻找合适的切换操作来减少分配 μ 中式 (3.2) 的总成本. 在此过程中, 每当需要计算特定联盟 $\mathcal{R}_m$ 完成任务 ω_m 的实际成本 f_m 时, 混合优化层会寻找联盟 $\mathcal{R}_m$ 的最佳模式和参数序列 $(\Xi_m, \mathbf{P}_m)$, 以完成 ω_m, 同时最小化实际成本. 这个过程会一直重复, 直到算法求得在实际成本意义下的 Nash 稳定或 K-Serial 稳定任务分配为止. 结果分配为 $\mu^\star$, 每个任务 $\omega_m \in \Omega$ 的最优计划为 $(\Xi^\star_m, \mathbf{P}^\star_m)$. 因此,$\omega_m$ 的最终混合计划 (2.2) 为:

$$\varphi^\star_m = (\xi^\star_{k^m_{t_1}}, \mathcal{R}_m, \rho^\star_{k^m_{t_1}}) \cdots (\xi^\star_{k^m_{t_N}}, \mathcal{R}_m, \rho^\star_{k^m_{t_N}}), \tag{3.26}$$

其中,$\Xi^\star_m = \xi^\star_{k^m_{t_1}} \cdots \xi^\star_{k^m_{t_N}}$ 和 $\mathbf{P}^\star_m = \rho^\star_{k^m_{t_1}} \cdots \rho^\star_{k^m_{t_N}}$, 而总长度因不同的任务而异. 根据这些混合计划, 每个代理 $n \in \mathcal{N}$ 可以开始执行分配的任务 $\mu^\star(n)$, 通过遵循选择的参数与最优模式的序列. 由于未预料到的干扰和不确定性, 系统状态可能会与计划的轨迹不同, 在这种情况下, 应根据当前系统状态重新解决问题 1, 更新任务分配和混合计划.

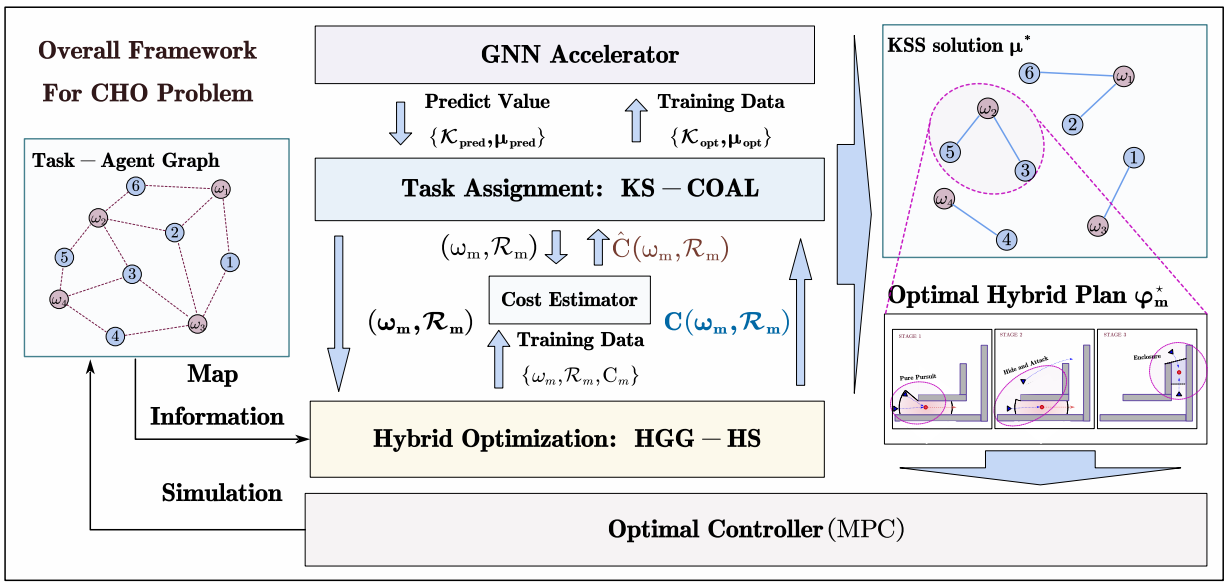


图 3.8. 组合-混合优化整体框架 (CHO Framework) 示意图

特别的, 如框架图3.8所示, 我们使估计代价函数以代价估计器 (Cost Estimator) 模块的形式工作, 允许估计成本函数具有可学习的参数, 并通过混合优化的运算结果为其提供训练数据样本.

理想情况下, 随着算法在特定场景规划经验的增长, 代价估计器所提供的估计成本和混合优化模块提供的实际成本的误差能够逐渐缩小. 可学习的估计成本函数具有多种可能的表达形式, 比如带少量参数的显示数值表达式, 或者采用神经网络 (Neural Network) 进行非线性拟合, 将任务和联盟信息编码为输入特征向量, 网络输出任务的估计代价. 显示数值表达式往往需要依赖于对问题结构性的认知, 根据的经验去设计, 好处是其参数拟合难度低, 缺点是对于复杂任务, 其表达能力较弱, 不具有良好的渐进性能. 基于神经网络的代价估计器可能具有更好的渐进性能, 能够拟合简单函数所不能处理的复杂非线性关系, 但缺点在于对数据量要求高.

理想情况是, 随着拟合的精度进一步提高, 我们能够在不降低解的质量的条件下, 逐渐减小任务分配层对混合优化层的调用次数, 大幅度提升算法运行效率. 特别地, 在提升代价估计器拟合精度的同时, 我们应该尽可能保证代价估计器所提供的的估计代价始终低于真实代价, 从而根据定理3.3.1和2, 在 NS-COAL 或者 KS-COAL 算法的稳定联盟形成过程中, 我们可以大量减少求解实际代价的需求量, 而解的质量保证却不受影响.

综上, 我们所提出的 CHO 框架在两个层次均给出了具有质量保证的具体实现, 并且通过合理地交替使用代价估计器和混合优化模块, 我们在一定程度上克服了组合优化和混合优化复杂度耦合导致的指数爆炸的问题.

第四章　实验

为了进一步验证所提出的方法, 本节展示了广泛的数值模拟实验.所提出的方法在 Python3 中实现, 并在一台装有 Intel Core i7-1280P CPU 的笔记本电脑上进行了测试.

4.1　合作搬运

4.1.1　实验设置

如第 2.5.1节介绍并在图 4.2中展示的那样, 大小为 $10m \times 10m$ 的工作空间布满了矩形障碍物. 在工作空间内随机分布着 16 个代理和 6 个箱子. 每个箱子的大小为 $1m \times 0.5m$, 圆形代理的半径为 $r = 0.1m$. 为了简单起见, 每个机器人具有一阶动力学, 并且可以在箱子的特定点提供有界的推力, 从而改变具有二阶平移不变动力学的箱子的状态.

需要注意的是, 混乱的工作空间引入了严重的几何约束, 使得运输任务比无障碍环境 [25; 26] 更加困难. 最初, 任务成本 $f(\cdot)$ 在式 (3.1)中被估计为代理到目标箱子的欧几里得距离之和以及到达其目标位置的距离. 启发式函数 $h^{\mathcal{G}}(x)$ 被设计为从状态 x 到 $\mathcal{X}_{G_m}$ 的最小距离, 由 $A^\star$ 搜索估计, 而 $h^{\mathcal{L}}(x)$ 则被设计为在局部几何约束下从当前状态 x 到最短路径上的下一个中间区域 $\hat{\mathcal{X}}$ 的运动代价, 其具体定义方式将在4.4.2节中进一步介绍.

为了在合作搬运问题中应用 HGG-HS 算法, 我们按照如图4.1所示的方式进行动作基元的设计. 如我们前面所描述的, 机器人通过在合作模式所指定的接触点施加一定大小呃

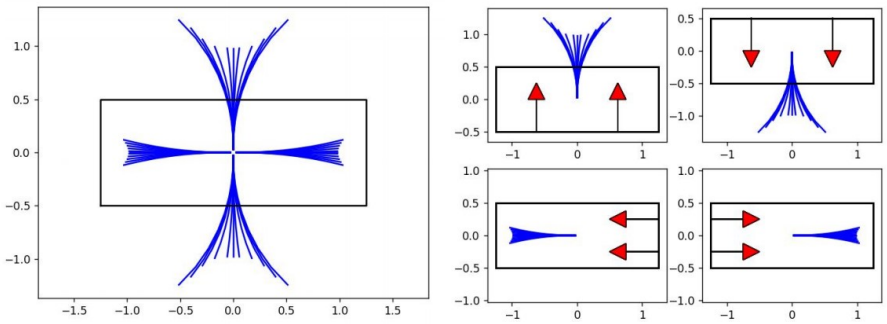


图 4.1. 两个不同场景的合作搬运仿真实验: 16 个机器人搬运 6 个箱子 (**Top**); 10 个机器人搬运 4 个箱子 (**Bottom**).

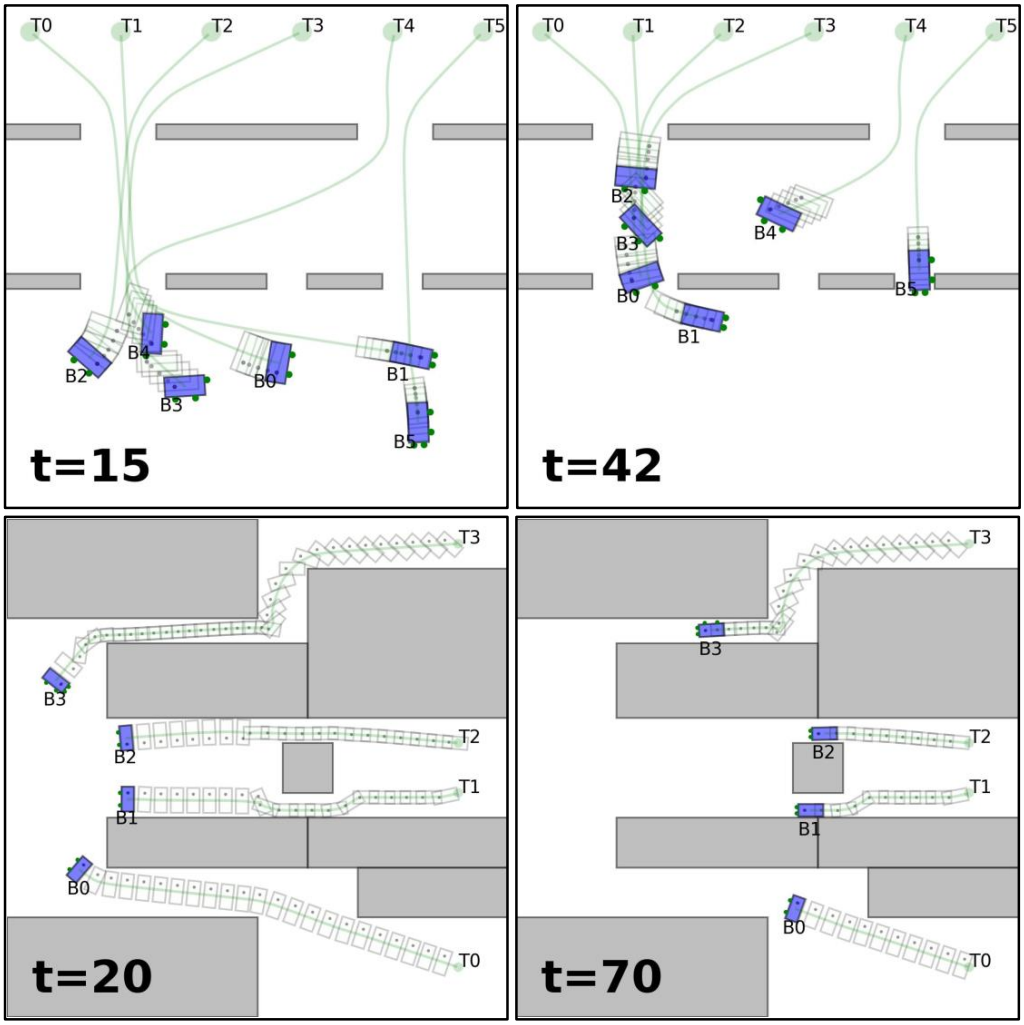


图 4.2. 两个不同场景的合作搬运仿真实验: 16 个机器人搬运 6 个箱子 (**Top**); 10 个机器人搬运 4 个箱子 (**Bottom**).

4.1.2 实验结果

系统状态的演变如图 4.2 所示．初始时,6 个任务和 16 个代理的联盟形成需要 9 秒, 在此期间, 在混合优化层中解决了 16 个混合搜索问题．可以看出, 不同的模式对总体成本有显著影响．例如, 通过选择使用 2 个代理以通过狭窄的通道的短侧推动模式, 箱子 5 达到其目标位置的速度更快; 分配了 3 个代理来推动箱子 4, 以便将其推过距离较远的 2 个通道; 通过几乎一条直线,4 个代理将箱子 5 推到其目标位置．为了进一步验证适用性, 考虑了另一个复杂情景, 并在图 4.2 中展示, 其中需要箱子进行许多急转弯才能到达目的地．因此, 在最终解决方案中可以找到更频繁的模式切换, 例如, 与箱子 3 相关的 4 个代理的联盟从“长边”模式切换到“短边”模式, 以便箱子可以在 $t = 45$ 秒通过第一个通道．然而, 由于从窄侧推动引入了更高的运动学不确定性, 因此成本更大, 同样的联盟在通过第二个通道后在 $t = 92$ 秒时切换到“对角线推动”模式．此外, 对于箱子 1 和 2, 在 $t = 70$ 时, 两个代

理从“长边”切换到“短边”通过中间通道.

4.2 动态抓捕

4.2.1 实验设置

如第 2.5.2 节介绍并在图 4.4 中展示的那样, 工作区包含随机放置的障碍物, 例如走廊和拐角, 使得捕获任务的决策难度比无障碍环境更高 [3; 30]. 10 个追捕者和 3 个逃脱者随机分布在一个大小为 $2.5m \times 2.5m$ 的工作区中, 为了简单起见, 所有智能体都满足单积分器动力学, 具有相同的最大速度为 $4m/s$. 对于追捕者来说, 系统状态始终可用. 追捕者的捕获半径设为 $0.15m$. 如第 2.5.2 节所述, 追捕者遵循三种参数化模式以协作方式捕获逃脱者. 对于联盟形成的第一层, 通过为每个追捕者分配最近的逃脱者来获得初始分配 ν_0. 在公式 (3.1) 中, 任务成本 $f(\cdot)$ 通过追捕者联盟到分配的逃脱者的加权平均距离来估计. 此外, 对于混合优化层, 启发式函数 $h^{\mathcal{G}}(\cdot)$ 被设计为到逃脱者的最小距离, $h^{\mathcal{L}}(\cdot)$ 则根据特定的局部环境进行设计, 例如, 对于封锁模式, 该启发式函数被定义为角落区域所形成的包围的面积.

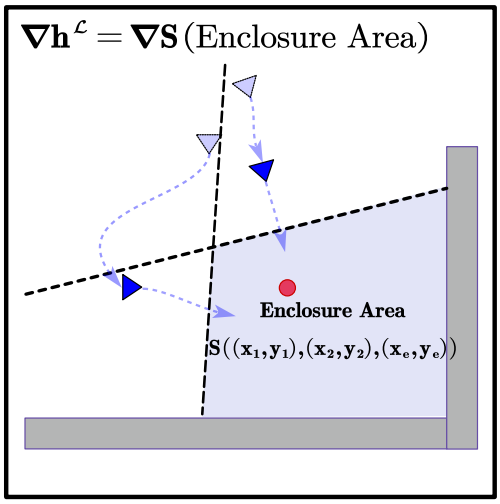


图 4.3. 围捕模式下的局部启发式函数设计

最后, 与“纯追逐”模式相关联的参数仅由目标逃脱者的位置确定; “藏匿攻击”模式的参数从给定的潜在位置集合中进行采样, 考虑到工作区布局和目标逃脱者; “封锁”模式的参数通过最大化优势区域确定 [31].

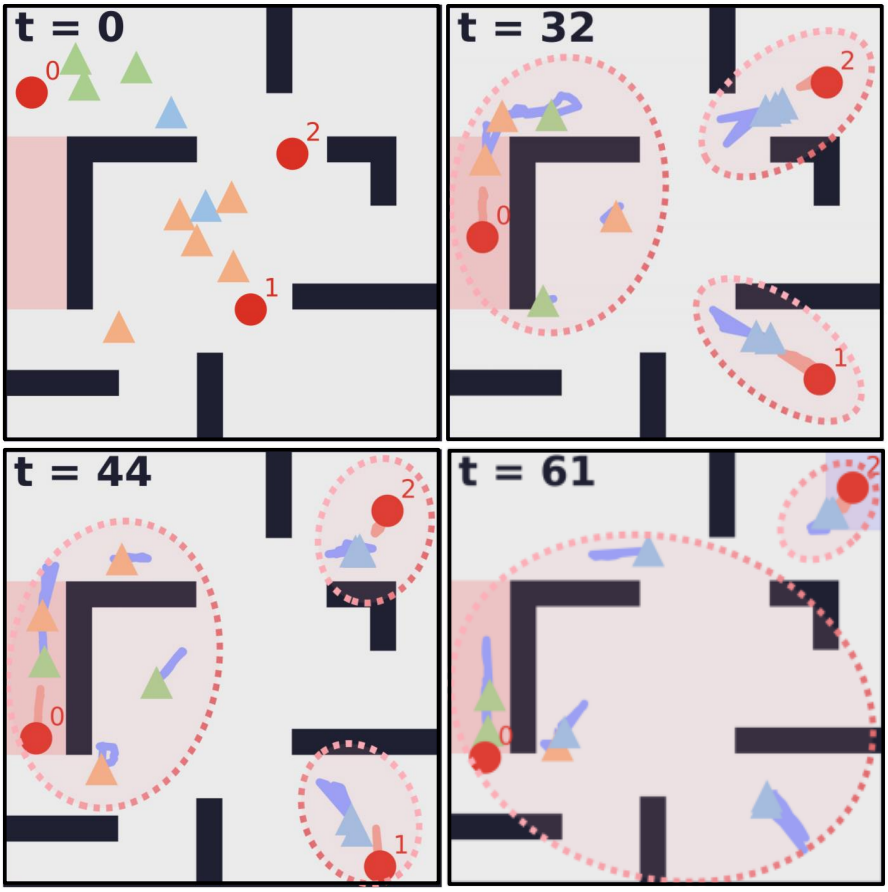


图 4.4. 动态抓捕仿真实验. 按照所属的联盟用虚线圈包围的追捕者 (三角形) 和逃避者 (红色圆).

4.2.2 实验结果

任务分配流程的第一层在每 15 个时间步长执行一次, 而混合优化的层在每 5 个时间步长更新一次, 以适应逃避者的快速移动. 在一个解下系统的演变在图 4.4中可视化. 最初, 追捕者和逃避者的位置随机初始化. 如图 4.5所示, 追捕者 $0, 1, 2, 3$ 和 6 最初被分配捕捉逃避者 0, 而追捕者 $4, 5$ 则是为逃避者 1 服务, 追捕者 $7, 8, 9$ 则是为逃避者 2 服务. 在 $t = 50$ 时, 设置参数为优势区域边界上的均匀分割点, 以模式“包围”捕获逃避者 1. 此后, 追捕者 $4, 5$ 以模式“纯追”加入了逃避者 2 的联盟. 然后, 逃避者 2 在 $t = 69$ 时在模式“包围”下被捕获, 此后, 追捕者 $4, 5$ 以模式“躲藏攻击”加入了逃避者 0 的联盟. 可以看出, 该模式的选择参数主要位于中间障碍物的四个角落. 在 $t = 75$ 时, 所有逃避者都被捕获, 期间总共进行了 3 次任务切换和 24 次模式切换.

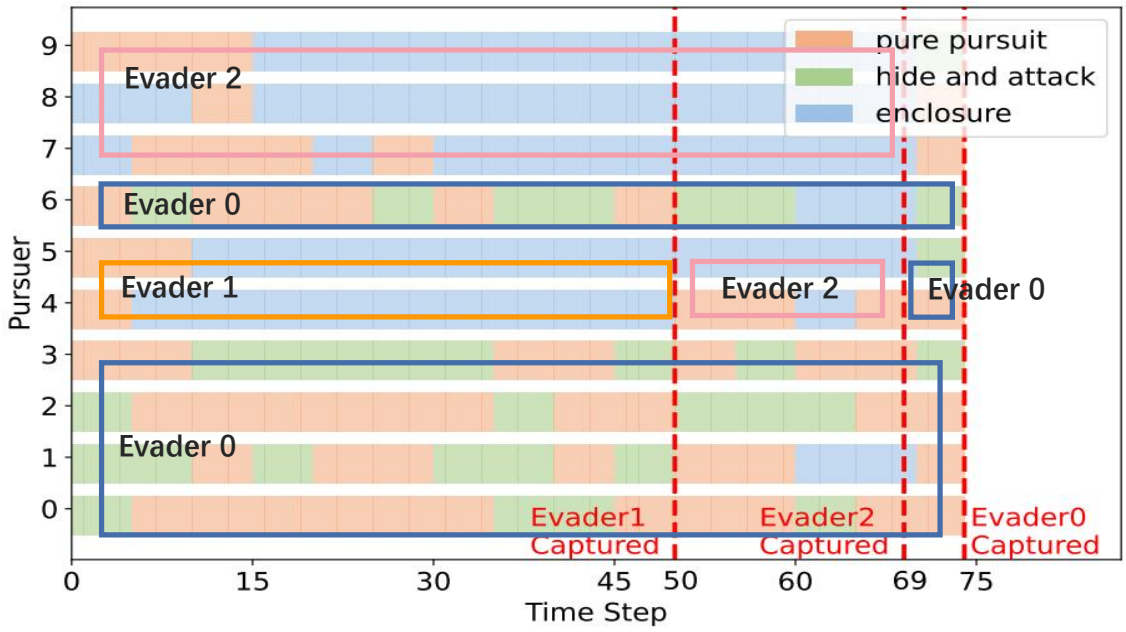


图 4.5. 在一次动态抓捕仿真实验中任务分配与合作模式的演化

4.3 数据分析与对比

本文介绍的组合-混合优化框架在协作两种场景上的有效性与两个基准方法进行了比较: (i) 贪心分配 (GA), 即将每个任务分配给最近的一个自由代理, 直到每个代理都有一个分配的任务; (ii) 固定模式 (FM), 即所有联盟都只遵循一种模式, 并且联盟形成层保持不变. 主要比较的指标是所有联盟混合计划的成本之和和完成所有任务的时间.

场景	方法	平均任务完成时间 (sec)	平均总代价
协同搬运	GA	127.8	20.6
	FM	132.7	26.1
	Ours	**108.5**	**19.0**
动态抓捕	GA	186	10.7
	FM	108	6.0
	Ours	**74**	**5.7**

表 4.1. Comparison with two baselines.

如表 4.1所述, 对于协作运输任务, 我们的算法在每个指标上都优于两个基准方法, 例如 GA 和 FM 方法的平均完成时间为 127.8s 和 132.7s, 明显高于我们的方法的 108.5s. 在这种情况下,FM 方法需要更长的时间, 因为不同模式的切换对于任务完成至关重要. 其次,

对于动态捕捉任务, 我们的方法需要约 74 个时间步骤来捕捉所有逃避者, 最短和最长捕捉时间分别为 70 和 80 步. 相比之下,FM 方法的捕捉时间要长得多, 为 108 步, 而 GA 方法的平均时间更长, 为 186 步. 有趣的是, 在 GA 方法下, 追逐者的子团队通常会同时接近和忽略其他逃避者, 导致分配不均衡.

4.4 补充实验

在本小节中, 我们进行了更多的补充实验, 用于验证我们在两个层次所提出的算法的有效性和运算效率.

4.4.1 KS-COAL 算法补充实验

我们针对动态抓捕问题对 KS-COAL 算法补充了新的实验. 之前所设置的实验场景包含较多的障碍物, 但总体的规模较小, KS-COAL 的分布式特点在大规模问题上的适用性尚未得到验证, 且图神经网络的加速特性未得到体现. 因此, 我们在更加大规模的无障碍物场景中应用了我们的算法, 并应用了图神经网络对我们的算法进行加速. 本实验同样采用 python3 实现. GNN 的实现基于 [36] 中的 *Pytorch geometric Graph*(PyG). 算法运行在 Intel Core i7-1280P CPU @ 2.0GHz 上, 并使用了 RTX3080 显卡.

具体地, 我们考虑了如图所示4.6的实验场景, 实验中包含三种机器人, 抓捕者 (SWAT)、逃避者 (TARGET) 和侦查者 (SCOUT), 三种机器人都具有一定的半径的观测范围, 其中抓捕者为地面无人车, 能够抓捕地面上的逃避者, 具有中等的移动速度和较小的观测范围; 侦察者为在稳定高度飞行的无人机, 具有较高的移动速度和较大的观测范围, 可以及时侦查到动态目标; 逃避者具有相对抓捕者较小的移动速率, 但其初始位置对于抓捕者和侦察者组成的联盟不可知. 同时, 为了丰富任务种类, 我们在考虑了在地图中存在随机生成的资源, 且其分布可以用混合高斯分布 (Gaussian Mixture Model) 来建模. 所有抓捕者和侦察者组成的联盟的总体目标是在保护地图上资源的同时, 尽可能多地捕捉逃避者. 对于该多目标优化问题, 我们通过恰当地设置不同任务基础效用的大小来配置各任务的权重. 而侦察者主要通过快速扫描地图来为抓捕者提供尽可能准确的地图资源分布信息和敌方目标的实时位置信息. 为了充分体现大规模场景下 KS-COAL 的性能, 我们在地图中设置了 100 个抓捕者,50 个目标单位, 和 20 个侦察者.

需要说明的是, 由于目标单位数目较多, 很可能出现部分目标对应的任务没有被分配任何智能体. 因此, 为了更加直观的数值显示, 我们将用任务的效用 U 而非代价 C 来评估任务的完成情况. 任务的效用的被定义为任务的基础效用 U_{base} 减去完成任务所需要的控制成本 C_{ctrl}. 当然, 以效用或代价作为目标函数本质上是相同的, 针对同一个联盟两者具有 $U = U_{base} - C$ 的关系, 但采用 U 可以给出更符合直观的实验数据. 相应的,KS-COAL 算

法中所有涉及到代价的部分将用效用代替, 并使相关不等号反号. 在该实验中, 我们对不同类型任务的估计效用按照显式的函数或者迭代过程获取, 由于其具体形式较为复杂, 且与本文其他内容相对独立, 对于我们分析 KS-COAL 的性能没有影响, 故我们将其放置于附录 B 中.

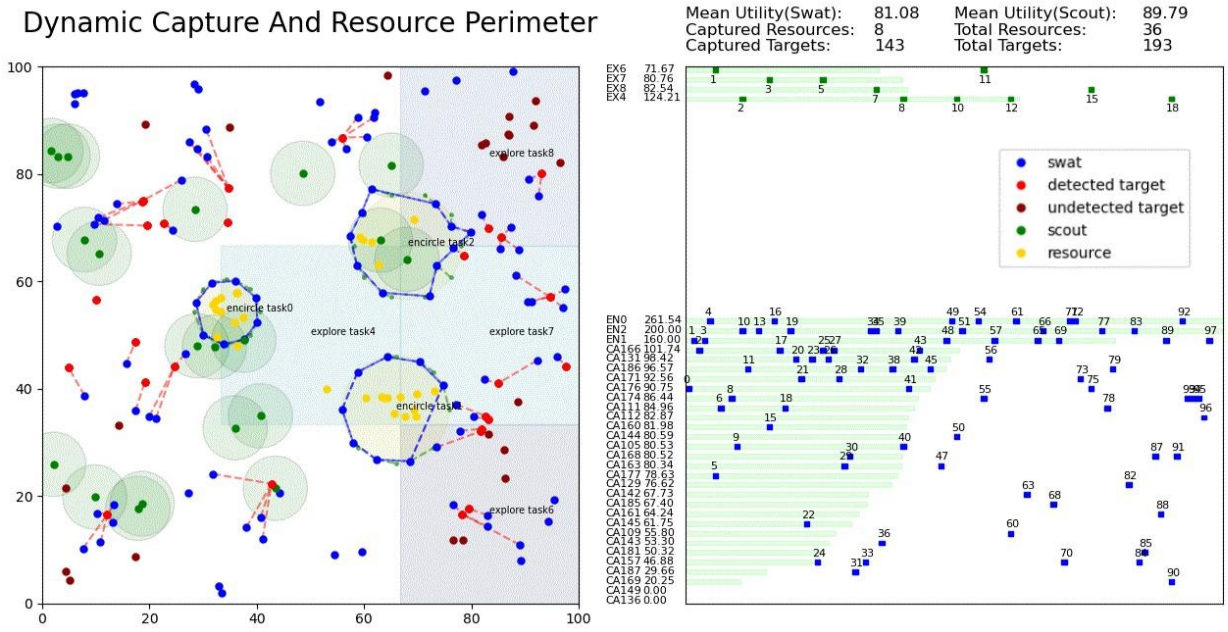


图 4.6. 动态捕捉与资源防守

实验的仿真图像如4.6所示, 左图中包含了上述所有的机器人种类和任务种类, 地图被划分为面积均等的 9 个子区域, 用于定义合作探索任务, 区域的颜色代表了其探索价值, 颜色越暗的区域探索价值越高. 右图展示了任务分配结果, 纵轴为任务 ID, 横轴为 SWAT 机器人 (蓝色小方块) 或者 SCOUT 机器人 (绿色小方块), 浅绿色条带的长度显示了对应任务的估计效用函数. 从图中可以看出, SWAT 机器人根据自身所在位置选择了合适的任务, 针对单个目标机器人, 通常形成了 2 4 个 SWAT 机器人的联盟进行合作抓捕, 更多的 SWAT 机器人通常导致边际效用下降. 针对地图中的三个资源聚落, SWAT 机器人形成了包围圈以防止目标机器人进入. 图4.7中展示了在仿真过程中的实时地图信息, 四个子图的具体含义见图片上侧描述文字. 可以看出, 随着 scout 对地图的周期性探索, 地图的不确定性得到有效控制, 并且资源分布的高斯中心被准确识别. 随着 SWAT 包围圈的形成, 单位时间内被目标获取的资源数目被有效控制.

我们将提出的方法与两种基线算法——GreedyNE 和 FastMaxsum 算法进行比较. 其中 GreedyNE, 即贪婪搜索算法等价于 K=1 的 KS-COAL 算法, 两者均能达成 Nash 均衡的稳定解. 而 FastMaxSum, 即快速最大和算法也是一种经典的分布式联盟形成算法. 同时, 为了评估 K 值和图神经网络加速器的影响, 我们的方法又分为 K=2, K=3, random, NACK,

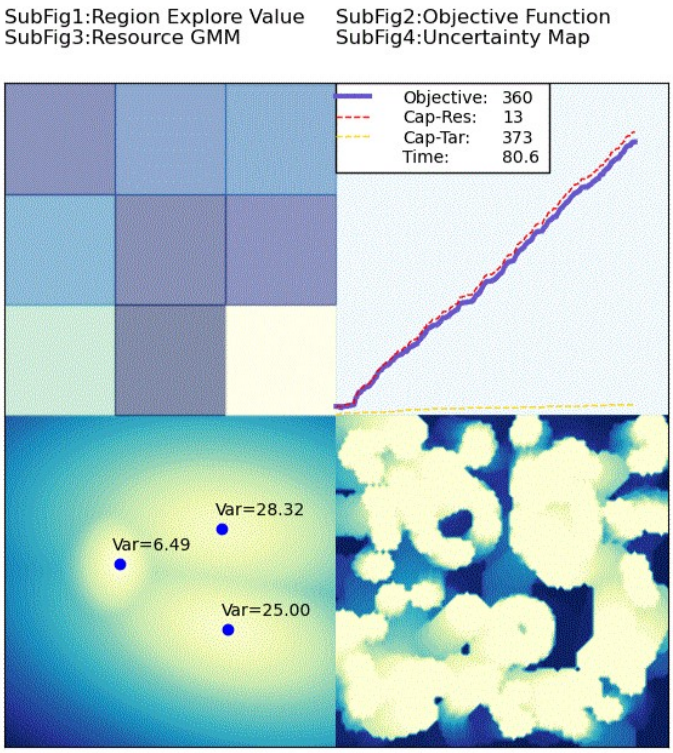
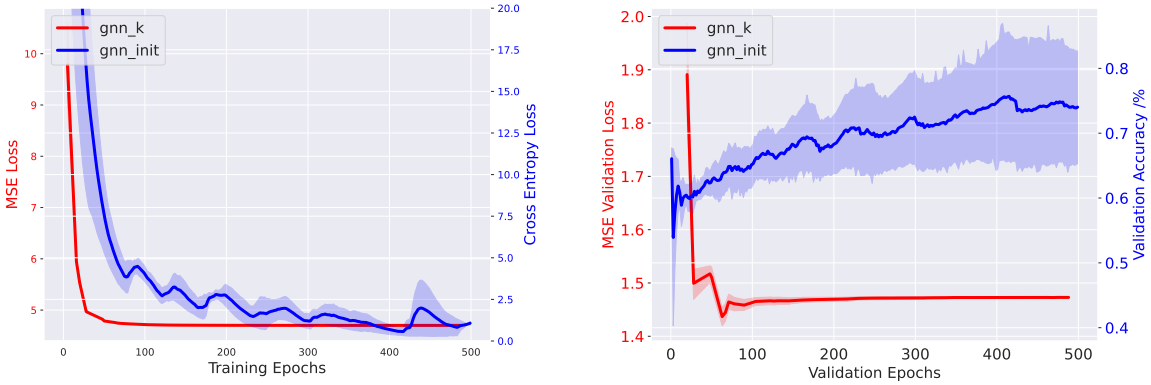


图 4.7. 实时信息显示

NACInit, NACKInit 六种设置．其中,random 组作为对照组, 我们将为 KS-COAL 提供随机的初始解; NACK 代表使用 GNN 预测最优的 K 值分布, 从而有侧重地进行局部优化; NACInit 代表使用 GNN 预测具有高效用的初始解, 在此基础上运行 K=3 的 KS-COAL 算法; NACKInit 则综合 NACK 和 NACInit, 同时预测 K 值分布和初始解．

在本实验中, GNN 的训练数据集由 K=3 的 KS-COAL 算法产生．GNNK 和 GNNInit 模型的训练过程图4.8所示．针对 K 值的预测, 我们采用均方差损失 (MSE Loss) 作为损失函数. 针对初始解的预测是一个多分类问题, 因此适合用交叉熵 (Cross Entropy) 作为损失函数. 从左图中可以看出随着训练的进行,GNNK 的 MSE 损失和 GNNInit 的交叉熵损失均能在数百个 epoches 内收敛到较小的稳定值. 右图展示了随训练同时进行的验证过程, 在验证集中,GNNK 的 MSE Loss 同样快速下降并收敛, GNNInit 的预测准确率则随着训练的进行波动上升, 在 500 个 epoches 的训练后达到了 74% 的准确率．此时准确率尚未收敛, 随着训练次数进一步增加, GNNInit 的预测初始解的准确率最终可以达到 90% 以上．



(a) Average total cost with different schemes. (b) Average time delay with different schemes.

图 4.8. Average cost with different schemes.

如图4.9所示, GreedyNE 方法具有最小的通信次数, 随着 K 值增大, 传递的消息量也随之指数上升. K=3 和 NACInit 具有最大的通信次数, 这是因为二者的均采用所有智能体 K=3 的设置来运行 KS-COAL 算法. 而采用了 NACK 方法的两组则具有较小的通信次数, 因为网络对于每个智能体节点预测出的 K 值主要分布在 1 3 之间, 相比于所有智能体 k=3 的组所需的消息量减小常数倍, 符合我们的预期.

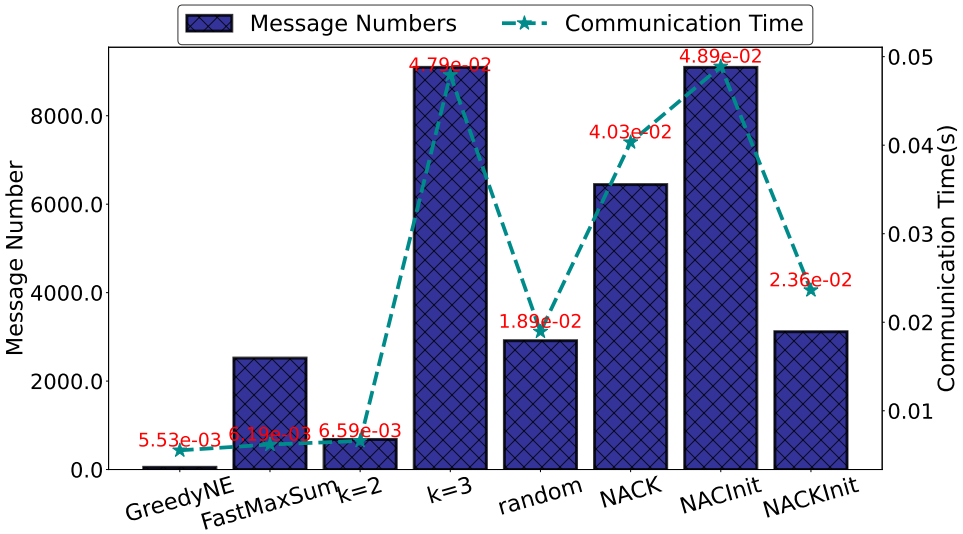


图 4.9. 静态实验消息传递

如图4.10所示, NACKInit 具有最好的性能, 其兼具了收敛速度与解的质量. 在不足 0.2s 内 NACKInit 的静态收益就快速增长到了 187. 与之相比, GreedyNE 尽管收敛迅速, 但是所得的 Nash 稳定解质量较差, 陷入了初始解附近的局部最优, 稳定收益仅有 150 左右. 而 FastMaxSum 则表现最差, 不仅在算法初期存在收益震荡, 而且其稳定收益也仅有 148.

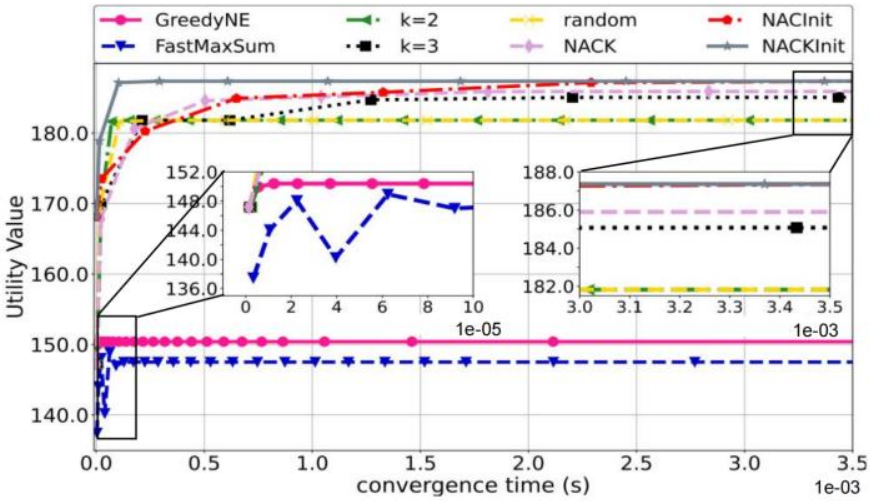


图 4.10. 静态实验的效用增长

NACInit 组也能收敛到和 NACKInit 相同的稳定收益, 但收敛时间为 NACKInit 的 10 倍左右, 这意味着其均匀的 K 值分布带来了大量的无效运算, 没有对稳定收益的增长提供关键作用. 有趣的是, 尽管 NACInit 和 NACKInit 的解也至多具有 K(3)-Serial 稳定的质量保证, 但其最终得到的稳定解的收益却高于 k=3 组的 185, 我们推测是因为通过对大量高质量分配方案的学习, GNN 跳出贪婪初始解或者随机初始解对应的 K-Serial 稳定解的局部最优范围, 找到了质量更高的局部最优解. 综上,NAC 方法基本取得了我们预期的效果, 并且在某些指标上取得了超越预期的预测准确度.

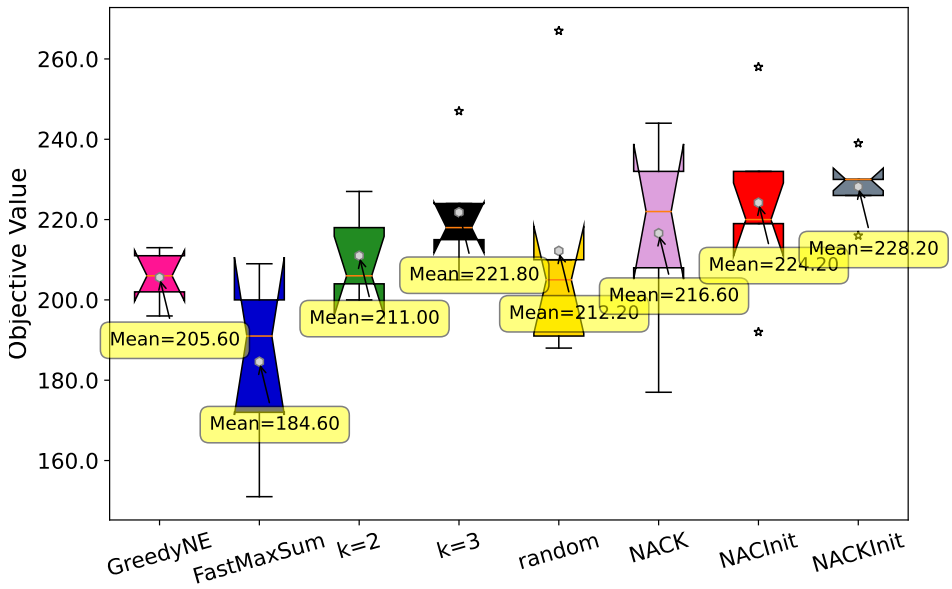


图 4.11. 动态实验的平均效用

图4.11展示了动态实验的结果, NACKInit 具有最大的平均目标函数值 228.20, 而 Fast-

MaxSum 算法的平均目标函数值仅为 184.60. NACK 和 NACInit 组仅此于 NACKInit, 高于其他对照组, 说明了结合 K 值选择和初始解预测是非常有效的, 两者产生了叠加效应. 上述动态实验结果与我们静态实验的结果基本相吻合, 证明了在我们恰当分配任务的权重时, 任务的静态平均效用提升将带来更高的动态收益.

4.4.2 HGG-HS 算法补充实验

我们针对合作搬运场景对 HGG-HS 做了进一步测试, 主要对比的基准方法是 Hybrid A*.

Hybrid A* 算法是一种图搜索算法, 改进于 A* 算法, 最早由斯坦福 Dmitri Dolgov, Sebastian Thrun 等人在 2010 年提出, 并在 (DARPA) 的城市挑战赛中得以运用. 与普通的 A* 算法区别在于,Hybrid A* 规划的路径考虑了车辆的运动学约束, 即满足了车辆的最大曲率约束. Hybrid A* 算法的启发式包括两种: non-holonomic without-obstacles 和 holonomic with obstacles. 前者考虑了车辆的运动约束, 但不考虑障碍物, 一般使用 RS 曲线,Dubins 曲线; 后者将车辆当成网格地图上面的点, 但考虑了障碍物. 通常而言,Hybrid A* 算法被用于解决车辆的路径规划, 考虑了车辆的前进和倒退模式切换. 将 Hybrid A* 应用于路径规划问题需要进行类似于我们在4.1.1中的处理, 将多机器人的高维控制量离散化并简化为动作基元. 同样的, 在 Hybrid A* 的每一步节点拓展中, 我们会针对每个接触模式都拓展一定数量的动作基元, 并检查是否碰撞. 与汽车模型不同的是, 对于合作搬运规划问题, 如2.5.1节所描述的, 不同接触模式下, 系统将具有显著不同的动力学约束, 包括箱子质心加速度方向范围和曲率范围. 同时, 由于机器人的数目和接触位置可以有多种组合, 这导致合作搬运问题具有大量了可选模式. 综上, 我们修改了 Hybrid A* 的模式和动作基元设计, 用作我们衡量 HGG-HS 算法性能的重要基准.

为了将 HGG-HS 算法用于合作搬运问题, 此处我们需要进行一些细节的补充. 由于 HGG-HS 算法需要局部启发式函数 $h^{\mathcal{L}}$ 作为引导, 因此对于我们的方法, 需要采用 A* 预搜索的方式来搜索一系列可能的中间目标位置. 在 A* 预搜索阶段, 我们不需要保证中间目标位置具有最优性, 而只是进行粗略的搜索, 并通过计算相邻两个中间目标位置之间不同模式对应的动力学代价来给出较优的模式可选项, 作为 HGG-HS 在进行节点拓展时的重要启发. 具体地, 此处我们用 Hermite 三次插值多项式来进行动力学代价的估计, 插值基函数和插值轨迹如下:

$$
\begin{aligned}
&h_0 = (1+2\frac{t-0}{1-0})(\frac{t-1}{0-1})^2, \quad h_1 = (1+2\frac{t-1}{0-1})^2 \\
&H_0 = (t-0)(\frac{t-1}{0-1})^2, \quad H_1 = (t-1)(\frac{t-0}{1-0})^2 \\
&x = x_0h_0 + x_1h_1 + cos(\psi_s)H_0 + cos(\psi_g)H_1 \\
&y = y_0h_0 + y_1h_1 + sin(\psi_s)H_0 + sin(\psi_g)H_1
\end{aligned}
\tag{4.1}
$$

从而, 我们可以得到一条估计的状态转移轨迹为:

$$X(t) = (x(t), y(t), \psi(t) = \mathbf{arctan}(\frac{y'(t)}{x'(t)})) \tag{4.2}$$

从估计的轨迹中, 可以计算估计的控制代价和障碍物损失代价:

$$\begin{aligned} J_u &= \Sigma_{i=0}^{N}(\ddot{x}^2 + \ddot{y}^2) \\ J_o &= \Sigma_{i=0}^{N}\Sigma_{p=1}^{P}\mathcal{L}_p(x(t), y(t), \psi(t)) \end{aligned} \tag{4.3}$$

其中,$\mathcal{L}_p$ 是盒子与障碍物的碰撞损失代价. 该碰撞损失代价可以用如图4.12的方式进行计算. 根据凸集的超平面分离定理, 我们只要能够找到一个投影方向使得两个凸集的投影交集为空集合, 则可以说明两个凸集可以被一个超平面分开, 也就是两个凸集不相交. 因此, 一个自然的想法是, 对于相交也就是碰撞的情况, 我们也可以用两个凸集的交集大小来评估碰撞损失的大小. 具体的, 由于我们问题假设中所有障碍物均为矩形, 因此只需要选择两个矩形一共四条边的方向作为投影方向, 就可以获得 4 个投影交集长度, 如下所示:

$$\begin{aligned} \mathcal{L}_{proj,1} &= \max\left(\min\left(x_{1,max}, x^r_{2,\max}\right) - \max\left(x_{1,\min}, x^r_{2,\min}\right), 0\right) \\ \mathcal{L}_{proj,2} &= \max\left(\min\left(y_{1,max}, y^r_{2,\max}\right) - \max\left(y_{1,\min}, y^r_{2,\min}\right), 0\right) \\ \mathcal{L}_{proj,3} &= \max\left(\min\left(x_{2,max}, x^r_{1,\max}\right) - \max\left(x_{2,\min}, x^r_{1,\min}\right), 0\right) \\ \mathcal{L}_{proj,4} &= \max\left(\min\left(y_{2,max}, y^r_{1,\max}\right) - \max\left(y_{2,\min}, y^r_{1,\min}\right), 0\right) \end{aligned} \tag{4.4}$$

而最终的障碍物损失函数则被定义为投影交集大小的最小值:

$$\mathcal{L}_p = \mathbf{min}(\mathcal{L}_{proj,1}, \mathcal{L}_{proj,2}, \mathcal{L}_{proj,3}, \mathcal{L}_{proj,4}) \tag{4.5}$$

其理论依据是我们只需要将最小值优化到 0, 根据凸集的超平面分离定理, 就能保证两个物体不碰撞. 并且, 按照该方式定义的障碍物损失函数在局部具有良好的可微性质, 适合用于基于梯度的非线性优化.

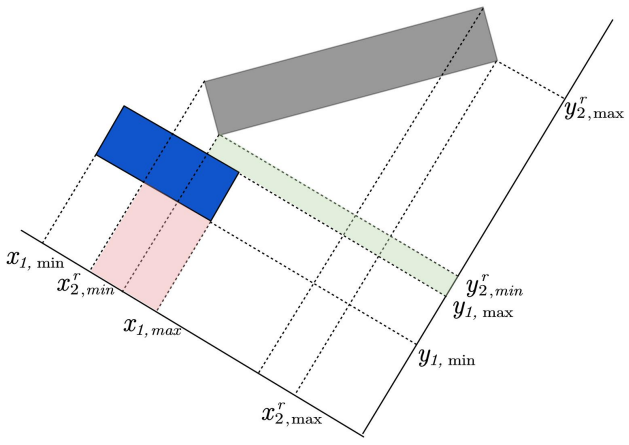


图 4.12. 矩形碰撞损失估计

如此以来, 我们就可以将局部启发式定义为:

$$h^{\mathcal{L}}(x) = \mathbf{min}_k J(x, \hat{\mathcal{X}}_k) = \mathbf{min}_k J_{s,k} + J_{s,k} \tag{4.6}$$

其中 $\mathcal{X}_k$ 是当前状态一定范围内的可选的局部目标, 通过预搜索确定.

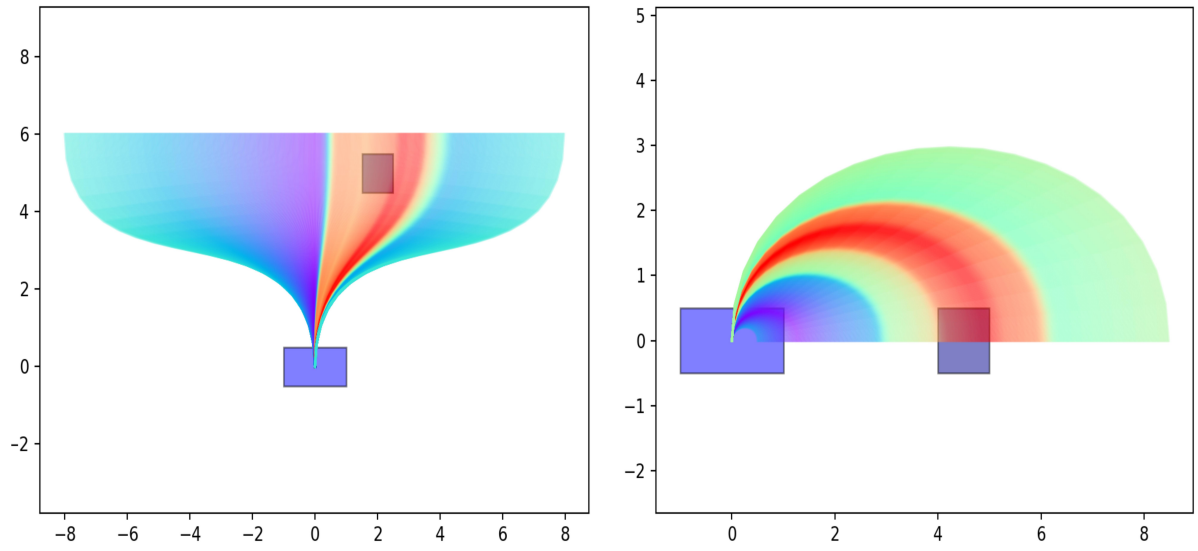


图 4.13. 箱子在长边推动模式下向不同状态迁移的估计动力学代价

对比实验在一个特定的地图中进行, 每次实验中我们随机生成一个可行的箱子初始位置和目标位置, 重复十次实验对数据取平均值.

方法	预搜索时间 (sec)	搜索总时间 (sec)	节点拓展次数	路径代价	模式切换次数
Hybrid A*	0.0	22.354	444.6	25.3498	2.0
HGG-HS	1.6	13.99	12.0	20.0	0.4

表 4.2. 和两种基线方法比较.

数据如表 4.2所示, 我们的方法在各项指标显著优于 Hybrid $A^{\star}$, 从表中可以看出, 我们的方法所需的搜索总时间为 13.99 秒, 这个时间包含了预搜索时间和混合搜索的时间, 与之形成对比的是 Hybrid A* 算法的总时间为 22.354 秒. 由于 Hybrid A* 不需要进行预搜索, 因此预搜索时间为 0. 这说明我们的算法在给定的任务下的搜索成本大幅度优于 Hybrid A* 算法. 在节点拓展次数方面,Hybrid A* 需要的平均拓展次数为 444.6, 而与之形成对比的是我们的方法只进行了 12 次节点拓展, 这说明在局部启发式的引导下, 我们的搜索算法能够在单次节点拓展的过程中, 通过迭代优化获得更大的单次步长. 由此看出, 在预搜索

给出的粗略规划的基础上, HGG-HS 往往只需要大体上跟随粗略规划, 但当粗略规划的结果不可行时, 用局部的基元拓展来修复其中不可行或者代价过高的部分, 在路径代价方面, Hybrid A* 的代价为 25.3, 而 HGG-HS 的代价为 20.0, 这符合我们将优化嵌入搜索的目标. 由于为了保证 Hybrid A* 的搜索速度, 我们往往不能选择步长过小的动作基元, 但这也限制了 Hybrid A* 通过启发式搜索获得的解的质量. 尽管 HGG-HS 同样受制于这一点, 但通过优化我们可以访问到在粗糙离散的动作基元之间的局部最优参数, 而这种局部最优效应的累计则反映在总的路径代价上, 体现了在搜索中合并优化的另一优势. 另外, 在模式切换次数方面, HGG-HS 在十个随机情景中仅仅花费了平均 0.4 次的路径模式切换次数, 远小于 Hybrid A* 的 2.0 次. 模式切换意味着机器人需要转移到盒子上新的接触位置, 这将带来额外的时间成本. 需要说明的是, 这部分代价并未包含在路径代价中, 这进一步证明了 HGG-HS 算法在解决合作规划问题时的优越性.

第五章 结论与展望

本文提出了一种面向协作任务的组合-混合优化 (CHO) 框架, 通过构建两个交错并行的优化层级, 该框架同时解决了任务联盟的动态形成和协作行为的混合优化. 对每一个层级, 我们都进行了通用的形式化问题建模, 并给出了高效的算法实现, 算法被证明具有良好的质量保证和可接受的时间复杂度, 在解的质量和求解效率之前取得较好的均衡. 两个层次在我们的框架中完成了深度的耦合, 相比于顺序进行两个层次的计算, 我们的框架进一步降低了不同联盟执行各种任务的整体成本.

更具体地, 在联盟形成层, 我们提出了集中式的 Nash 稳定联盟形成算法 NS-COAL 以及分布式的 K-Serial 稳定联盟形成算法 KS-COAL, 通过理论证明给出了其质量保证, 两种联盟形成算法都通过有效的筛选机制来避免无意义的混合优化问题求解. 同时, 我们基于图神经网络设计了神经加速器, 大幅度地提升了 KS-COAL 算法在大规模问题上的求解效率, 并且可以该加速器同样可以被分布式部署在真实的多机器人平台中.

在混合优化层, 我们将多智能体合作规划问题形式化为了关于离散的模式决策和连续的模式参数的优化问题, 并提出了启发式梯度引导混合搜索算法 (HGG-HS) 算法, 充分利用了图搜索算法的最优性保证和优化方法在高维问题中的优势, 使用全局和局部两种近似水平的启发式函数来引导搜索, 并具有可接受的时间复杂度和可证明的质量保证.

此外, 由于我们的框架提供了可增量学习的代价估计模块, 随着算法框架在特定场景中的规划经验的不断丰富, 该框架将具有更好的渐进性能.

针对上述算法框架, 我们研究了协作运输和动态抓捕这两个非平凡应用, 通过基准测试和对比实验有效地说明了耦合任务分配与协作行为规划的必要性.

未来的工作主要包括以下几个方向:

1. 模型和数据结合的模型预测: 针对真实环境中的多机器人合作问题, 我们往往难以对系统物理模型进行精确建模, 其原因可能是物体物理属性不完全可知, 或者环境中存在较大的随机干扰. 这将对模型预测控制的性能造成负面影响. 通过将模型和数据结合, 能够缓解纯模型方法在应对复杂物理模型时的欠拟合, 同时减小学习方法对数据的需求.

2. 动态物理约束: 对于多机器人合作任务, 机器人与机器人之间或机器人与环境之间可能存在动态的物理约束, 一方面, 这可能缩小了混合优化的决策变量的自由度, 但另一方面, 通过机器人主动地改变约束类型, 团队可能能够完成更加复杂的任务. 动态

物理约束将对运动规划算法的设计提出更高的要求.

3. 深度学习加速运动规划: 由于神经网络的强大表达能力, 深度学习能够有效改善搜索、采样和优化三种运动规划方法各自的缺陷. 相比于直接由神经网络端到端地输出决策序列, 利用神经网络加速运动规划算法具有更高的质量保证.

参考文献

[1] Thomas Bock. Construction robotics. *Autonomous Robots*, 22(3):201–209, 2007.

[2] Marko Krizmancic, Barbara Arbanas, Tamara Petrovic, Frano Petric, and Stjepan Bogdan. Cooperative aerial-ground multi-robot system for automated construction tasks. *IEEE Robotics and Automation Letters*, 5(2):798–805, 2020.

[3] Alyssa Pierson, Zijian Wang, and Mac Schwager. Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers. *IEEE Robotics and Automation Letters*, 2(2):530–537, 2016.

[4] Asif Khan, Bernhard Rinner, and Andrea Cavallaro. Cooperative robots to observe moving targets. *IEEE transactions on cybernetics*, 48(1):187–198, 2016.

[5] Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Štolba. Cooperative multi-agent planning: A survey. *ACM Computing Surveys (CSUR)*, 50(6):1–32, 2017.

[6] Maria Gini. Multi-robot allocation of tasks with temporal and ordering constraints. In *AAAI Conference on Artificial Intelligence*, 2017.

[7] Raphaël Massin, Christophe J Le Martret, and Philippe Ciblat. A coalition formation game for distributed node clustering in mobile ad hoc networks. *IEEE Transactions on Wireless Communications*, 16(6):3940–3952, 2017.

[8] Isaac E Weintraub, Meir Pachter, and Eloy Garcia. An introduction to pursuit-evasion differential games. In *IEEE American Control Conference (ACC)*, pages 1049–1066, 2020.

[9] Jesus Tordesillas and Jonathan P How. Mader: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics*, 38(1):463–476, 2021.

[10] Venkata Ramana Makkapati and Panagiotis Tsiotras. Optimal evading strategies and task allocation in multi-player pursuit-evasion problems. *Dynamic Games and Applications*, 9:1168–1187, 2019.

[11] Valentin N Hartmann, Ozgur S Oguz, Danny Driess, Marc Toussaint, and Achim Menges. Robust task and motion planning for long-horizon architectural construction

planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6886–6893, 2020.

[12] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks 2015*, pages 31–51, 2015.

[13] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi De Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[14] Ernesto Nunes and Maria Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[15] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986.

[16] Krzysztof R Apt and Andreas Witzel. A generic approach to coalition formation. *International game theory review*, 11(03):347–367, 2009.

[17] Talal Rahwan, Tomasz P Michalak, Michael Wooldridge, and Nicholas R Jennings. Coalition structure generation: A survey. *Artificial Intelligence*, 229:139–174, 2015.

[18] Lovekesh Vig and Julie A Adams. Multi-robot coalition formation. *IEEE transactions on robotics*, 22(4):637–649, 2006.

[19] Fredrik Präntare and Fredrik Heintz. An anytime algorithm for optimal simultaneous coalition structure generation and assignment. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–31, 2020.

[20] Tomasz Michalak, Talal Rahwan, Edith Elkind, Michael Wooldridge, and Nicholas R Jennings. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230:14–50, 2016.

[21] Francesco Maria Delle Fave, Alex Rogers, Zhe Xu, Salah Sukkarieh, and Nicholas R Jennings. Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *2012 IEEE International Conference on Robotics and Automation*, pages 469–476. IEEE, 2012.

[22] Cristino De Souza, Rhys Newbury, Akansel Cosgun, Pedro Castillo, Boris Vidolov, and Dana Kulić. Decentralized multi-agent pursuit using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4552–4559, 2021.

[23] Enrica Soria, Fabrizio Schiano, and Dario Floreano. Distributed predictive drone swarms in cluttered environments. *IEEE Robotics and Automation Letters*, 7(1):73–80, 2021.

[24] Xinmiao Sun and Christos G Cassandras. Optimal dynamic formation control of multi-agent systems in constrained environments. *Automatica*, 73:169–179, 2016.

[25] Elio Tuci, Muhanad HM Alkilabi, and Otar Akanyeti. Cooperative object transport in multi-robot systems: A review of the state-of-the-art. *Frontiers in Robotics and AI*, 5:59, 2018.

[26] Konstantinos I Alevizos, Charalampos P Bechlioulis, and Kostas J Kyriakopoulos. Bounded energy collisions in human–robot cooperative transportation. *IEEE/ASME Transactions on Mechatronics*, 27(6):4541–4549, 2022.

[27] Ramkumar Natarajan, Howie Choset, and Maxim Likhachev. Interleaving graph search and trajectory optimization for aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 6(3):5357–5364, 2021.

[28] Wolfgang Hönig, Joaquim Ortiz-Haro, and Marc Toussaint. Benchmarking sampling-, search-, and optimization-based approaches for time-optimal kinodynamic mobile robot motion planning.

[29] Mehmet Dogar, Andrew Spielberg, Stuart Baker, and Daniela Rus. Multi-robot grasp planning for sequential assembly operations. *Autonomous Robots*, 43:649–664, 2019.

[30] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. Multiplayer reach-avoid games via pairwise outcomes. *IEEE Transactions on Automatic Control*, 62(3):1451–1457, 2016.

[31] Alexander Von Moll, David W. Casbeer, Eloy Garcia, and Dejan Milutinović. Pursuit-evasion of an evader by multiple pursuers. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 133–142, 2018.

[32] Qinyuan Li, Minyi Li, Bao Quoc Vo, and Ryszard Kowalczyk. An anytime algorithm for large-scale heterogeneous task allocation. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 206–215, 2020.

[33] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019.

[34] Shuhao Cui, Xuan Jin, Shuhui Wang, Yuan He, and Qingming Huang. Heuristic domain adaptation. *Advances in Neural Information Processing Systems*, 33:7571–7583, 2020.

[35] Andreas Wächter. Short tutorial: Getting started with ipopt in 90 minutes. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2009.

[36] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[37] Alexander Von Moll, David W. Casbeer, Eloy Garcia, and Dejan Milutinović. Pursuit-evasion of an evader by multiple pursuers. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 133–142, 2018.

[38] Tomoki Toda, Alan W Black, and Keiichi Tokuda. Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(8):2222–2235, 2007.

[39] Vishnu S. Chipade and Dimitra Panagou. Multiagent planning and control for swarm herding in 2-d obstacle environments under bounded inputs. *IEEE Transactions on Robotics*, 37(6):1956–1972, 2021.

附录 A

非线性优化器 IPOPT 参数设定

本文在 HGGHS 算法的迭代优化过程中用到了 IPOPT 作为主要的非线性优化求解器. Ipopt 是一个开源的软件包, 用于大规模非线性优化. 它可以被用于解决如下所示的通用的非线性优化问题 (Nonlinear Optimization Problem) 形式:

$$\begin{aligned} \textbf{min} \quad & f(x) \\ \textbf{s.t.} \quad & g^L \leq g(x) \leq g^U \\ & x^L \leq x \leq x^U \end{aligned} \tag{5.1}$$

Ipopt 所能求解的非线性优化问题的形式如下: 给定优化变量 $x \in \mathbb{R}^n$（可能带有上下界,$x_L \in (\mathbb{R} \cup -\infty)^n$ 和 $x_U \in (\mathbb{R} \cup +\infty)^n$）, 其中 $x_L \leq x \leq x_U$,$f : \mathbb{R}^n \to \mathbb{R}$ 是目标函数, $g : \mathbb{R}^n \to \mathbb{R}^m$ 是一般非线性约束函数. 函数 $f(x)$ 和 $g(x)$ 可以是线性或非线性的, 凸或非凸的（但必须具有二阶连续可微性）. 约束函数 $g(x)$ 具有下界和上界, $g_L \in (\mathbb{R} \cup -\infty)^m$ 和 $g_U \in (\mathbb{R} \cup +\infty)^m$, 其中 $g_L \leq g(x) \leq g_U$. 注意, 可以通过设置 $g_{L,i} = g_{U,i} = g_{\bar{i}}$ 来指定形式为 $g_i(x) = \bar{g}_i$ 的等式约束.

Ipopt 实现了一种内点线搜索过滤方法来实现非线性问题. 在本文的实验中, 我们通过调用 python 的 CasAdi 库附带的 Ipopt 优化器实现来完成一般的非线性优化问题的求解. CasADi 是一个用于自动微分和数值优化的符号框架, 便于我们使用易于理解的直观方式书写非线性优化问题的. 如第3.4.4节中所描述的, HGG-HS 算法在迭代优化过程中, 不是进行单次的非线性优化问题的求解, 而是进行多次的迭代优化, 直到满足特定条件时终止. 具体地, 我们通过为每一次优化设置初始值和添加关于决策变量的额外约束来实现. 我们约束由决策变量决定的轨迹末端节点只能在初始值的半径为 d 的邻域内进行优化, 特别的, 对于物体搬运问题而言, 半径 d 通常被设置为 0.4m. 同时, 为了避免不必要的优化精度, 我们设定优化器的"max_iter" 参数, 即最大迭代次数为 100, 设定"acceptable_tol" 参数, 即可接受的收敛容许误差为 1e-4. 另外, 参数"warm_start_init_point" 设置为 true, 即开启初始点的热启动, 以增强优化器寻找高质量解的能力,

值得一提的是, 本文采用 Ipopt 作为优化器的主要原因是出于框架的泛用性考虑, 但针对具体的问题, 往往可以通过对模型的适当简化来将优化问题转化为某种可以高效求解的优化问题, 从而避免非线性优化带来的高时间成本问题.

附录 B

KS-COAL 补充实验中的任务效用函数设置

合作探索与侦察

不失一般性地, 工作空间被划分为一组区域, 表示为 $\mathcal{W} = r_m$. 它可以基于特征, 例如我们感兴趣的区域, 也可以基于几何网格, 例如三角形或均匀网格的划分. 每个区域都包含一项探索任务, 供 SCOUT 机器人团队执行. 访问区域 r_m 的潜在收益由以下公式给出:

$$\chi(r_m) = u_c(r_m) + \rho_{\mathtt{tg}}(r_m) + \rho_{\mathtt{rs}}(r_m), \tag{5.2}$$

其中 $u_c(r_m) = 1 - \lambda^{-t_e}$ 表示关于区域 r_m 特性的不确定性, t_e 是上次 SCOUT 机器人访问区域 r_m 的时间, $\lambda \in (0, 1)$ 是设计参数; $\rho_{\mathtt{tg}}(r_m), \rho_{\mathtt{rs}}(r_m)$ 衡量每个区域内目标和资源的密度. 这些信息可以通过 SWAT 或 SCOUT 机器人进行在线感知和共享. 如果 $\chi(r_m)$ 检测到高于预定阈值, 则会在 SCOUT 机器人之间广播探索任务 ω_m.

多个 SCOUT 机器人的勘探策略相对简单: 将区域按机器人数目划分为多个子区域, 并且每个代理在一个子区域上扫描. 当联盟内的所有机器人都完成了各自区域的探索任务时, 勘探任务就完成了. 鉴于上述策略, 相关的效用函数 $f_{\mathtt{exp}}$ 为

$$f_{\mathtt{exp}}(\widehat{\mathcal{R}}_{\mathtt{o}}, \omega_m) \triangleq \frac{\chi(r_m)}{\tau(\widehat{\mathcal{R}}_{\mathtt{o}}, r_m)}, \tag{5.3}$$

其中,$\widehat{\mathcal{R}}\mathtt{o} \subset \mathcal{R}\mathtt{o}$ 是可能被分配到探索任务 ω_m 的侦察机器人的子集; $\chi(r_m)$ 在式 (5.2) 中定义; $\tau(\widehat{\mathcal{R}}\mathtt{o}, , r_m)$ 通过以下公式估算任务的持续时间:

$$\tau(\widehat{\mathcal{R}}_{\mathtt{o}}, r_m) \triangleq \mathbf{max}_{i \in \widehat{\mathcal{R}}_{\mathtt{o}}} \left\{ t_i(s_i, r_m) + \frac{A_m}{2\widehat{N}_s R_i v_i} \right\}, \tag{5.4}$$

其中,$t_i(s_i, , r_m)$ 衡量代理 $i \in \widehat{\mathcal{R}}\mathtt{o}$ 从其位置 s_i 到区域 r_m 的所需时间; A_m 是区域 r_m 的总面积; $\widehat{N}s = |\widehat{\mathcal{R}}\mathtt{o}|$; $2R_i v_i$ 是代理 i 每单位时间探测的区域面积, 其中 $R_i, , v_i$ 是其检测半径和参考速度. 因此, 总持续时间由最慢的机器人决定. 换句话说, 如果一个机器人从远处加入大的联合体, 则其边际效用较低.

合作捕捉

一旦 SCOUT 机器人通过探测发现目标机器人, 如其位置和速度, 相关信息将会广播给突击机器人. 接收到这些信息后, 每个目标机器人都将被分配一个抓捕任务. 显然, 如果有

多个突击机器人进行协作, 则一个目标机器人可以更容易地被捕获. 对于多个追捕者和一个逃避者, 使用类似于 [37] 的协作捕捉策略.

更具体地, 考虑形成联盟以捕捉一个目标机器人 $j \in \mathcal{R}_{\mathtt{S}}$ 的突击机器人子集 $\widehat{\mathcal{R}}_{\mathtt{C}} \subseteq \mathcal{R}_{\mathtt{C}}$. 每个机器人 $i \in \widehat{\mathcal{R}}_{\mathtt{C}}$ 都会遵循纯追踪策略, 通过不断调整方向以朝向目标机器人并以恒定速度 v_i 移动. 另一方面, 目标机器人总是尝试最大化自身存活时间, 通过朝向远离 $\widehat{\mathcal{C}}_j \triangleq \bigcap_i \mathcal{C}_{ij}$ 中的最远点来躲避, 即所有突击机器人 $i \in \widehat{\mathcal{R}}_{\mathtt{c}}$ 的阿波罗尼斯圆 $\mathcal{C}_{ij}$ 的交集区域和目标机器人 j. 阿波罗尼斯圆 $\mathcal{C}_{ij}$ 的定义为:

$$\mathcal{C}_{ij} = \left\{ s \in \mathbb{R}^2 \,|\, \|s - s_i\| \leq \frac{v_i}{v_j} \|s - s_j\| \right\}, \tag{5.5}$$

其中 $(s_i, v_i), (s_j, v_j)$ 是机器人 i, j 的位置和参考速度. 在 [37] 中已经表明, 通过上述运动策略, 目标机器人最终将被 $\widehat{\mathcal{C}}_j$ 捕获.

当目标机器人被捕获并变得不动时, 捕获任务就完成了. 因此, 部署联盟 $\widehat{\mathcal{R}}_{\mathtt{C}}$ 参与机器人 j 的捕获任务 ω_j 的效用由以下公式给出:

$$f_{\mathtt{cap}}(\widehat{\mathcal{R}}_{\mathtt{c}}, \omega_j) \triangleq \Gamma - \mathbf{min}_{i \in \widehat{\mathcal{R}}_{\mathtt{c}}} \left\{ t_i(\widehat{\mathcal{C}}_j, s_i) \right\}, \tag{5.6}$$

其中,$\Gamma > 0$ 是一个设计参数, 用于平衡不同的任务; $\widehat{\mathcal{C}}_j$ 是先前定义的交叉区域; $t_i(\widehat{\mathcal{C}}_j, , s_i)$ 估计机器人 i 从其位置 s_i 到达捕获区域 $\widehat{\mathcal{C}}_j$ 所需的最短时间.

通过形成包围来保护资源

SCOUT 机器人可能会检测到按下式描述的 GMM 分布进行随机生成的资源 ξ_t:

$$\xi_t \sim \mathtt{GMM}(\{(\alpha_k, \mu_k, \Sigma_k)\}), \tag{5.7}$$

即由未知均值和协方差的 K 个分量的高斯混合模型 (GMM) 组成. 给定检测到的资源位置, 通过最大似然估计 [38], 可以确定隐含的高斯聚类 G_k. 然后, 为了进一步防止目标机器人窃取这些资源, SWAT 机器人可以通过包围策略形成圆圈环绕这些聚类, 类似于 [39]. 具体而言, 考虑一个 SWAT 机器人子集 $\widehat{\mathcal{R}}_{\mathtt{C}} \subseteq \mathcal{R}_{\mathtt{C}}$ 和一个聚类 G_k, 其包围任务用 ω_k 表示. 首先, 计算以 μ_k 为圆心, 半径为 $3\sigma_k$ 的圆作为防御范围. 然后, 位于 $\widehat{\mathcal{R}}_{\mathtt{c}}$ 中的 SWAT 机器人导航到这个圆周上, 并保持彼此之间的等距离, 即每个 SWAT 机器人保卫圆周的一个等长的部分. 之后,SWAT 机器人保持静止, 并捕捉任何进入其捕捉半径的目标机器人. 在给定的时间后, 包围任务完成并释放代理.

通过以上运动策略, 更多的 SWAT 机器人会导致每个机器人防守的部分更小, 从而产生更高的捕获率. 然而, 如果这个数量超过了一个阈值, 并且它们的捕获区域大部分重叠,

那么增加另一个 SWAT 机器人的好处很快就会减少．因此, 我们提出以下效用函数:

$$f_{\mathsf{enc}}(\widehat{\mathcal{R}}_{\mathsf{c}},\, \omega_k) \triangleq \mathbf{sat}\left(\sum_{i\in\widehat{\mathcal{R}}_{\mathsf{c}}} \frac{d_{\mathsf{c},i}}{3\pi\sigma_k}\right), \tag{5.8}$$

其中,$\widehat{\mathcal{R}}\mathsf{c} \subseteq \mathcal{R}\mathsf{c}$ 包含联盟内的 SWAT 机器人; $d_{\mathsf{c},,i}$ 是机器人 i 的捕获半径; $3\sigma_k$ 是形成的圆的半径; 因此,$d_{\mathsf{c},i}/(3\pi\sigma_k)$ 估计了机器人 $i \in \widehat{\mathcal{R}}_{\mathsf{c}}$ 防御的圆的部分; $\mathbf{sat}(\cdot)$ 是输出区间 $[0,,1]$ 的剪切函数．因此, 在联合防御内的 SWAT 机器人数量达到时, 它们防御的圆的部分总和最接近但小于 1．此后, 如果有更多的 SWAT 机器人加入, 总效用保持不变．

其中,$\widehat{\mathcal{R}}\mathsf{c} \subseteq \mathcal{R}\mathsf{c}$ 包含联盟内的 SWAT 机器人; $d_{\mathsf{c},,i}$ 是机器人 i 的捕获半径; $3\sigma_k$ 是形成的圆的半径; 因此,$d_{\mathsf{c},i}/(3\pi\sigma_k)$ 估计了机器人 $i \in \widehat{\mathcal{R}}_{\mathsf{c}}$ 防御的圆的部分; $\mathbf{sat}(\cdot)$ 是输出区间 $[0,,1]$ 的剪切函数．因此, 在联合防御内的 SWAT 机器人数量达到时, 它们防御的圆的部分总和最接近但小于 1．此后, 如果有更多的 SWAT 机器人加入, 总效用保持不变.

致谢

本科四年即将走向尾声．很幸运能够在北大度过这段生命中无比珍贵的时光．四年中，有难眠的焦虑，也有丰收的欣喜；有一时的消沉，也有激扬的热情．有人说，我们这一届最不幸，上学半年就遇上了疫情，疫情结束时却即将毕业．想来确实是这样，但无论如何，至少我们所接收的学术训练仍然是合格且充分的，在这段艰难的岁月里，这大约是难能可贵的．回首过往，求学道路上的十余载不乏坎坷，幸得良师益友相助，方能顺利走到今日，完成这篇沉淀了我四年来的学习和思考的论文．

在此，首先要感谢我的导师，国萌老师，您是我进入机器人科研道路的引路人．进入课题组的一年来，我从对科研仅有模糊认识，到初步有能力撰写学术论文，离不开您的悉心指导与恳切交流，在一次次富有激情和思维碰撞的讨论中，我收获的不仅仅是学识的增长，还有您思考问题的方式，您面对生活和工作的乐观态度．

同时也要感谢实验室的同学们，特别是陈俊峰师兄，在我们的密切合作中给了我很大的帮助，让我能够快速地理解科研的基本范式，进入到科研的正确节奏中．还有冯禹铭师弟，也在这项研究中给了我许多必要的帮助．希望在以后的道路上，我们能够互相帮助，共同前行．

感谢我的家人，在我成长的道路上，你们始终坚定地支持我对未来的选择，鼓励我去尝试，去探索未知．养育之恩，无以回报，惟愿你们永远健康快乐．

感谢所有在我大学四年中给予我包容和支持的人，感谢所有和我分享快乐与温暖的人，虽然可能我们只有数面之缘，但生活中偶然的相逢和善意也值得珍惜．没有你们，我的大学生活也一定黯然失色．

最后，我想以《燕园情》作结：

我们今天东风桃李，用青春完成作业；

我们明天巨木成林，让中华震惊世界．

燕园情，千千结，问少年心事，眼底未名水，胸中黄河月．

我们心中一致的理想追求，尽管可能一时被焦虑所遮蔽，但我仍然相信，有些事物不会轻易更易．当我们再次相逢，心中一定还会响起这亲切的旋律．

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明: 所呈交的学位论文, 是本人在导师的指导下, 独立进行研究工作所取得的成果. 除文中已经注明引用的内容外, 本论文不含任何其他个人或集体已经发表或撰写过的作品或成果. 对本文的研究做出重要贡献的个人和集体, 均已在文中以明确方式标明. 本声明的法律结果由本人承担.

论文作者签名:

日期: 年 月 日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定, 即:

- 按照学校要求提交学位论文的印刷本和电子版本;
- 学校有权保存学位论文的印刷本和电子版, 并提供目录检索与阅览服务, 在校园网上提供服务;
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文;

论文作者签名:

日期: 年 月 日