中国科学: 信息科学 2024 年 第 54 卷 第 11 期: 2623–2641
SCIENTIA SINICA Informationis
论文

基于动作链的形式化任务协同规划

刘泽森, 李忠奎, 国萌*

北京大学工学院, 北京 100871
* 通信作者. E-mail: meng.guo@pku.edu.cn

收稿日期: 2024–04–03; 修回日期: 2024–07–13; 接受日期: 2024–08–16; 网络出版日期: 2024–11–08

国家自然科学基金 (批准号: U2241214, 62203017, T2121002) 资助项目

摘要 基于形式化方法的多智能体任务规划因其丰富的任务形式和多样的系统功能而备受关注. 然而, 随着智能体数量增加, 规划复杂度呈指数级增长, 因此形式化方法在计算效率和集群规模上都受到了限制. 已有的改良方法中, 基于图搜索的方法对计算效率改进有限, 只能处理中等规模的集群; 而基于组合优化的算法忽略了智能体动作模型, 无法处理大规模协同任务. 本文针对以上局限, 提出了基于智能体动作链的偏序任务分配算法, 在保证正确性的基础上大幅提升规划性能. 进一步针对环境不确定性和在线新任务, 设计了自适应重规划算法和协作同步机制, 确保任务执行过程中的鲁棒性. 最后, 通过数值仿真和对比验证了算法的有效性和可靠性.

关键词 多智能体任务规划, 形式化方法, 线性时序逻辑, 在线自适应, 偏序集

1 引言

多智能体集群协作能大幅拓展单个智能体的功能, 提高智能体的效率, 从而在众多的领域得到应用 [1]. 在多智能体集群的研究与应用中, 任务规划是实现高效协同合作的重要基础 [2]. 集群任务规划方法可以分为几大类, 包括基于优化的方法 [3~5], 基于机器学习的方法 [6~8], 基于群算法的方法 [9,10] 和基于形式化语言的方法 [11,12] 等. 但大部分现有算法主要处理不带逻辑约束的规划问题, 在面对需要逻辑推理的复杂任务时, 会遇到建模上的困难. 而形式化语言具有灵活的表达内容和精确的任务描述能力, 方便指定复杂的高级任务 [13]. 因此在面对难以用简单目标函数描述的任务 [14] 和复杂的环境 [15] 时, 形式化方法应用更广泛. 在形式化方法中, 线性时序逻辑是一类常用的语言, 能用任务公式描述一系列需要满足复杂时序和条件的动作, 已被广泛应用于搜索救援 [16], 设备维护 [11] 和医疗服务 [17] 等领域. 形式化方法可以分为集中式和分布式两种, 集中式的方法更关注于提高集群的规模和计算效率, 而分布式的方法则关注于在受限通讯情况下的集群协作.

引用格式: 刘泽森, 李忠奎, 国萌. 基于动作链的形式化任务协同规划. 中国科学: 信息科学, 2024, 54: 2623–2641, doi: 10.1360/SSI-2024-0105
Liu Z S, Li Z K, Guo M. Collaborative planning of formal tasks based on action chains (in Chinese). Sci Sin Inform, 2024, 54: 2623–2641, doi: 10.1360/SSI-2024-0105

最具代表性的集中式框架是基于模型检查的算法. 首先, 通过 SPIN[18] 和 LTL2BA[19] 等工具软件, 将任务公式转换为确定性 Robin 自动机或非确定性 Büchi 自动机. 其次, 采用有限加权转移系统 [13], 马尔可夫决策过程 [20] 或 Petri 网 [21] 等描述智能体的模型, 在公式的自动机和智能体模型之间创建乘积自动机. 最后, 采用图搜索的方法在乘积自动机中找到有效且可接受的路径, 作为智能体的计划 [22]. 由于乘积自动机的状态数量会随着智能体数量的增长而指数爆炸 [19], 为此研究者们进行了两个方面的改进: 修改乘积自动机的生成方式, 比如采用采样的方法 [23] 只生成部分自动机, 或者基于可分解集的方法 [24] 生成较小的自动机; 或者放弃生成乘积自动机, 先将自动机内的序列拆分为子任务, 并利用独立性分析判断任务间的逻辑 [11], 再采用处理组合优化的算法进行任务分配, 比如分支定界法 [11], 遗传算法 [25], 启发类方法 [26] 等. 在上述方法中, 基于图搜索的方法由于需要构建乘积自动机, 计算速度慢, 集群数量小, 优化目标单一, 很难处理带有时间长度和协作需求的任务. 而基于组合优化类的算法计算速度快, 集群数量不限, 优化目标丰富, 但是只能规划公式中显含的任务, 缺乏对动作链的推理能力. 因此, 如何在保留推理能力的情况下, 实现大规模集群的高效率任务规划, 是形式化方法亟待解决的重要问题.

基于上述讨论, 本文对多智能体形式化任务的协同规划问题开展研究. 利用智能体和环境模型计算能协同动作的前置和后置执行条件, 形成动作链. 在避免生成乘积自动机的同时, 保留了图搜索算法根据执行条件推理补全潜在动作的能力. 并结合组合优化类算法, 设计了复杂时序约束和资源约束下基于改进分支定界法的任务分配方法. 进一步, 针对在线生成任务和环境不确定性, 构建了动态自适应规划框架. 采用局部搜索方法处理在线发布的任务, 并利用同步机制保证任务执行过程中的鲁棒性. 最后, 通过数值仿真试验和对比, 验证了算法兼顾规划效率和推理能力.

本文的主要贡献在于: 1) 提出了复杂度与智能体数量无关的动作链计算方法, 实现了形式化语言公式在环境和智能体模型中高效推理; 2) 提出了基于动作链的任意时间协同规划方法, 计算速度优于现有图搜索类算法; 3) 提出了基于任务量调整重规划时间的在线适应方案和主 - 从式同步方法.

2 问题定义

2.1 线性时序逻辑

线性时序逻辑 (linear temporal logic, LTL) 公式由一组原子命题, 布尔运算符和时态运算符组成, 其中原子命题是不可分割的基本命题, 只能处于真或假的状态中. LTL 的语法如 [13] 所述, 定义为: $\varphi \triangleq \top|p|\varphi_1 \wedge \varphi_2|\neg\varphi|\bigcirc\varphi|\varphi_1 U \varphi_2$, 式中 $\top \triangleq$ True, $p \in AP$, $\bigcirc$(下一刻), $\bigcup$(直到), 以及 $\bot \triangleq \neg\top$. 其他的时序运算符可由上述符号组成: $\diamond\varphi \triangleq \text{True}\bigcup\varphi$(最终), $\Box \triangleq \neg\diamond\neg\varphi$(总是), $\varphi_1 \Rightarrow \varphi_2 \triangleq \neg\varphi_1 \vee \varphi_2$(映射). 此外, 对于给定的 LTL 公式 φ, 存在一个非确定性 Büchi 自动机 (mondeterministic büchi automaton, NBA).

定义1 (NBA) NBA 是一个四元组 $\mathcal{B} \triangleq (S, \Sigma, \delta, (S_0, S_F))$, 其中 S 为状态; $\Sigma \triangleq AP$ 是原子命题集; $\delta : S \times \Sigma \to 2^S$ 为状态间转移关系; $S_0, S_F \subseteq S$ 分别为初始状态和接受状态.

无限单词 (word) w 是一个由原子命题组成的无限序列 $w \triangleq \sigma_1\sigma_2\cdots, \sigma_i \in 2^{AP}$. 单词 w 在 $\mathcal{B}$ 内的运行 (run) 的结果是一个无限序列 $\rho \triangleq s_0 s_1 s_2 \cdots$, $s_0 \in S_0$, 且满足转移关系: $s_{i+1} \in \delta(s_i, \sigma_i)$. 满足 $\inf(\rho) \cap S_F \neq \varnothing$ 的运行是可接受的 (accepting). 这里 $\inf(\rho)$ 是在 ρ 中出现无限次的状态的集合, 即运行会反复访问接受状态. φ 的语言是一个由所有满足 φ 的单词构成的集合, 即 $\mathcal{L}(\varphi) \triangleq Words(\varphi) = \{w|w \vDash \varphi\}$, 其中 $\vDash$ 表示满足关系.

一种特殊类别的 LTL 公式被称为语法协同安全公式 (syntactically co-safe LTL, sc-LTL), 可以由一组长度有限的单词满足. 它们只包含时态运算符 $\bigcirc, \bigcup$ 和 $\diamond$, 并且以肯定形式书写, 即否定运算符 $\neg$ 不被允许出现在时序运算符之前.

2.2 拟偏序集合

根据我们先前的工作 [17], 在一个 NBA $\mathcal{B}_\varphi$ 上可以得到多个拟偏序集 (relaxed partially ordered set, R-poset).

定义2 (拟偏序集) 拟偏序集 $P_\varphi \triangleq(\Omega_\varphi, \preccurlyeq_\varphi, \neq_\varphi)$ 是一个三元组: $\Omega_\varphi \triangleq \{\omega_\ell = (\ell, \sigma_\ell, \sigma_\ell^s), \forall \ell = 0, \ldots, L\}$ 是子任务集合, 其中 ℓ 是子任务 ω_ℓ 的索引; $\sigma_\ell \subseteq \Sigma$ 是任务的执行要求; $\sigma_\ell^s \subseteq \Sigma$ 是任务的前置要求. $\preccurlyeq_\varphi \subseteq \Omega_\varphi \times \Omega_\varphi$ 表示顺序关系, 如果 $(\omega_h, \omega_\ell) \in \preccurlyeq_\varphi$, 则只有在 ω_h 开始运行后, ω_ℓ 才能允许开始运行. $\neq_\varphi \subseteq 2^{\Omega_\varphi}$ 表示冲突关系, 如果 $\omega_h, \cdots, \omega_\ell \in \neq_\varphi$, 则 $\omega_h, \cdots, \omega_\ell$ 不能同时执行.

偏序 $P \triangleq (\Omega, \preccurlyeq, \neq)$ 的语言 $\mathcal{L}(P) \triangleq \{w | w \vDash P\}$ 是所有满足偏序的单词 w 的集合. 对于一组 LTL 公式集 Φ, 我们可以计算出其中每个公式的拟偏序集, 然后计算乘积. 最终乘积 P_{final} 的语言满足 $\mathcal{L}(P_{final}) \in \mathcal{L}(P_{o_1}) \cap \cdots \cap \mathcal{L}(P_U)$, 即 P_{final} 能满足整个公式集 Φ.

定义3 (拟偏序集的乘积 [27]) 给定一个有限的单词 w_0, 两个拟偏序集 P_1, P_2 的乘积被定义为一组新拟偏序集 $\mathcal{P}^r \triangleq \{P'_1, P'_2, \cdots\}$, 表示为 $\mathcal{P}^r = P_1 \otimes P_2$, 其中 $w_0 w \in \mathcal{L}(P_1), w \in \mathcal{L}(P_2)$ 是 $w_0 w \in \bigcup\limits_{P'_i \in \mathcal{P}^r} \mathcal{L}(P')$ 的充分必要条件.

2.3 多智能体系统

考虑多智能体系统 $\mathcal{N} \triangleq \{1, \cdots, N\}$ 在工作空间 $W \in \mathbb{R}^2$ 中, 其中有 M 个感兴趣的区域表示为 $\mathcal{W} \triangleq \{W_1, \cdots, W_M\}$. 该环境可以定义 4 中的加权转换系统 (WTS). 在区域 m 内可能存放着一组资源, 记为 $R_m \triangleq \{r_1, \cdots, r_l\}, l \in \mathcal{R}$, 其中 $r_l \leqslant 0$ 代表资源种类 l 在区域 m 的数量, $\mathcal{R}$ 代表所有资源. 资源成本函数为 $C_r : \mathcal{R} \to R^+$. 由于多智能体系统是异构的, 每个智能体 n 有独立的模型 E_n, 其内容见定义 5. 每个智能体 $n \in \mathcal{N}$ 能够提供一组动作 $\mathcal{A}_n, n \in \mathcal{N}$. 所有动作的集合为 $\mathcal{A} \triangleq \bigcup\limits_{n \in \mathcal{N}} \mathcal{A}_n = \{a_1, \cdots, a_I\}$. 函数 $d_{\text{act}} : \mathcal{A} \to \mathbb{R}^+$ 表示动作需要执行时长, 其中动作 $a_l \in \mathcal{A}$ 可以由一组任意数量的智能体 $\mathcal{N}_l \subset \mathcal{N}$ 协同完成, 且累计执行时长保持不变即 $\sum\limits_{n \in \mathcal{N}_l} t_n = d_{\text{act}}(a_l)$. 另外, 函数 $C_{\mathcal{A}} : \mathcal{A} \to \mathcal{R}$ 表示执行 $\mathcal{A}$ 中某些动作会消耗环境中的资源.

定义4 (环境模型) 环境模型被表示为加权转换系统 WTS: $\mathcal{T} \triangleq \{\mathcal{W}, \to_{\text{t}}, AP, L_{\text{t}}, C_{\text{t}}\}$, 其中 $\mathcal{W}$ 是状态集; $\to_t \subseteq \mathcal{W} \times \mathcal{W}$ 是转移关系, AP 是原子命题, $L_t : \mathcal{W} \to 2^{AP}$ 是标记函数, $C_t : \mathcal{W} \times \mathcal{W} \to \mathbb{R}^+$ 是转移代价函数.

定义5 (智能体模型) 智能体 n 的模型被表示为自动机 $E_n \triangleq \{S_n, s_0^n, \mathcal{A}_n, \to_n, \lambda_n, C_{\mathcal{A}}\}$, 其中 S_n 是状态集, s_0^n 是初始状态, $\mathcal{A}_n$ 是智能体 n 的动作集, $\to_n : S^n \times \mathcal{A}_n \times S^n$ 是动作转移关系, λ_n 是动作的标记函数, $C_{\mathcal{A}} : W_m, \mathcal{A}_n \to R_m$ 是执行动作时的资源成本函数, 表示执行动作会消耗对应区域某一种资源.

例1 如图 1 所示, 一个 "志愿者" 智能体可被描述为一个自动机, 其中状态集 S^n: 初始状态, 运输水, 运输盐水, 运输食物, s_0^n 为 '初始状态'. 动作集 $\mathcal{A}_n$:获取水, 提供水, 转移关系 $\to_n$: 在 "运输水" 状态执行 "提供水" 动作后会转移到 "初始状态". 标记函数 λ_n:λ_n(获取水)=gw, 资源成本函数 $C_{\mathcal{A}}$: 动作 "获取水" 需要消耗一份水.

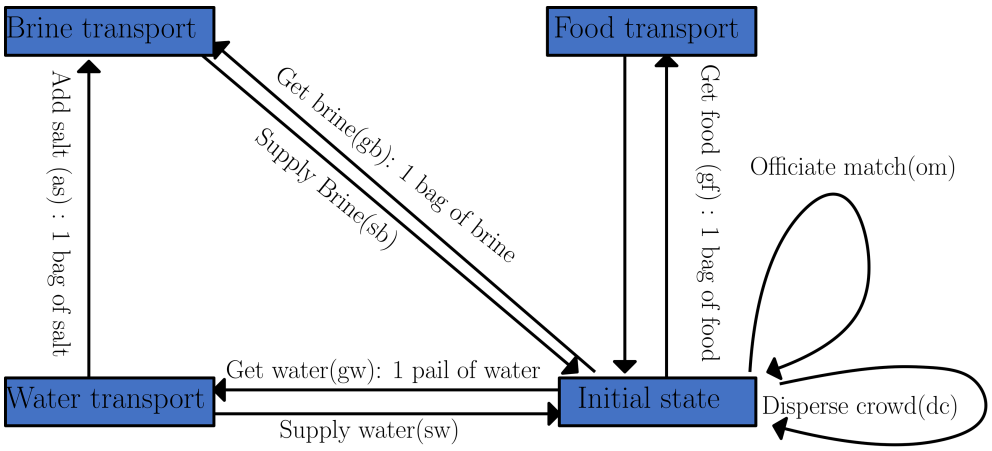

图 1　(网络版彩图) 志愿者智能体模型

Figure 1　(Color online) Agent model of the volunteer

2.4　问题定义

考虑以下两种原子命题: (i) 当任何智能体位于区域 $W_m \in \mathcal{W}$ 时, p_m 为真; 令 $\mathbf{p} \triangleq \{p_m, \forall W_m \in \mathcal{W}\}$; (ii) 当动作 a_i 在区域 W_m 执行时, $a_{i,m}$ 为真; 令 $\mathbf{a} \triangleq \{a_{im}, \forall a_i \in \mathcal{A}, \forall W_m \in \mathcal{W}\}$; 基于这些命题可以定义公式 $\varphi = sc - LTL(\mathbf{p}, \mathbf{a})$. 多智能体系统的任务可以定义为一个公式集 $\Phi \triangleq \{\varphi_1, \cdots, \varphi_O, \varphi_{O+1}, \cdots, \varphi_U\}$, 其中 $\varphi_o, \forall o \leqslant O$ 在执行之前给定, $\varphi_u, \forall O < u \leqslant U$ 在执行过程中在线给出. 为了满足 Φ 中的每个公式, 多智能体的动作序列被定义为

$$\mathcal{J} \triangleq (J_1, J_2, \cdots, J_N), \tag{1}$$

其中 $J_n \in \mathcal{J}$ 是单个智能体的任务序列, 序列中的元素表示需要执行的动作, 将在 3.2 节中详细定义. 为了评估动作序列 $\mathcal{J}$, 设计了三个维度的指标来评估执行效果, 最大完成时间 t_m, 资源成本 c_r 和智能体路径 c_t. 优化目标函数定义为

$$\min_{\mathcal{J}} \{\alpha t_m + \beta c_r + \gamma c_t\}, \tag{2}$$

其中 α, β 和 γ 是相应的系数.

因此, 问题可以描述为给定任务公式集 Φ, 规划并更新智能体的动作序列 $\mathcal{J}$ 以满足所有公式, 同时最小化完成式 2 中的综合成本.

如图 2 所示, 所提出处理方案由三部分组成. 首先, 通过偏序乘法将形式化任务公式转换为拟偏序集实现高效的任务分解, 具体算法在文献 [27] 中. 其次, 在离线规划部分, 通过将拟偏序集和智能体动作模型和环境模型联合搜索, 能生成集群的动作 - 路径链; 然后采用分支定界法对动作 - 路径链进行任务分配, 见第 3 章. 最后, 通过局部搜索处理在线新任务, 以及在线同步应对执行不确定性, 见第 4 章.

3　考虑复杂动作模型的离线规划

3.1　动作 - 路径链计算

首先, 针对给定的形式化语言公式集 Φ, 根据文献 [27] 所提出的偏序乘法, 将这些公式转换为拟

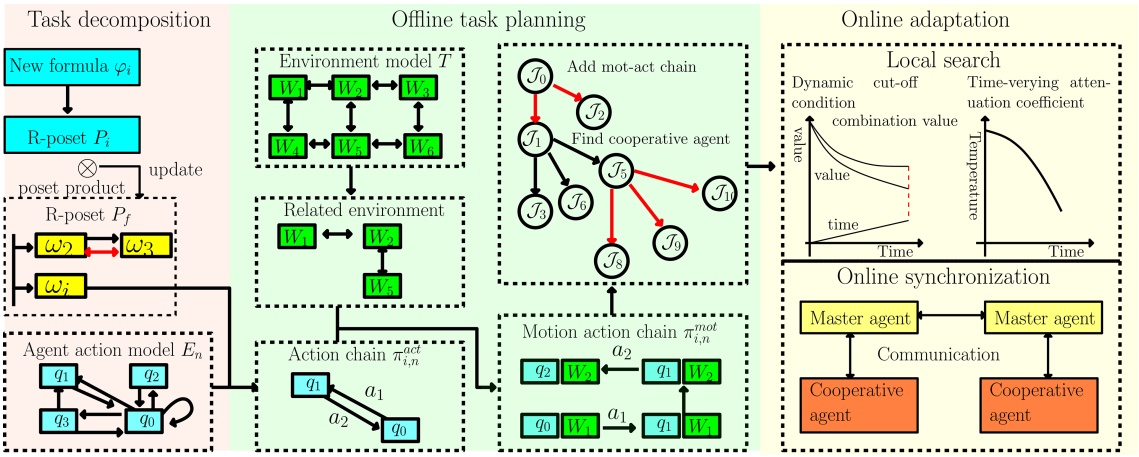

图 2 (网络版彩图) 算法整体架构

Figure 2 (Color online) Overall architecture of the algorithm

偏序集, 并获取它们的乘积 $P_{final} \triangleq (\Omega_f, \preccurlyeq_f, \neq_f)$. 通过在 $\preccurlyeq_f$ 和 $\neq_f$ 约束下执行任务 Ω_f, 公式集 Φ 将会满足. 为执行子任务 $\omega_i \triangleq (i, \{a_{i,m}\}, \{\}) \in \Omega_f$, 智能体需要在区域 W_m 执行动 a_i 作以实现任务所需要的原子命题 $a_{i,m}$. 但是对于一个处在状态 s^n_0, 且 $s^n_0, a_i, s^n_k \notin \rightarrow_n, \forall s^n_k \in S_n$ 的智能体, 它无法直接执行动作 a_i, 而必须先执行其他的动作以转移到可行的状态 s^n_{k1}, $(s^n_{k1}, a_i, s^n_{k2}) \in \rightarrow_n$. 同理, 在执行动作 a_i 之后, 若当前状态 $s^n_k \neq s^n_0$, 则还需要执行额外的动作以转移回初始状态 s^n_0. 而这些其他的动作由于资源分布的限制, 可能需要前往别的区域执行. 因此, 这些包涵智能体 E_n 和环境模型 $\mathcal{T}$ 信息, 且能满足子任务 ω_i 的动作 - 路径序列, 被称为智能体 n 能提供的并且满足子任务 ω_i 的动作 - 路径链.

定义6 (动作 - 路径链) 动作 - 路径链是一串动作 - 位置序列 $\pi^{\text{mot}}_{i,n} \triangleq (a_1, W_{m1}) \cdots (a_j, W_{mj})$, 表示依照顺序在区域内执行对应动作. 动作所消耗的资源与环境模型相匹配, 即 $C_{\mathcal{A}}(a_j, W_{mj}) \in R_j$. $\pi^{\text{act}}_{i,n} \triangleq [a_1, \cdots, a_j]$ 能使智能体模型的状态从 s_0 出发最终转移回 s_0, 称为动作链. 智能体 E_n 所有能提供给 ω_i 的动作链集合记为 $\Pi^{\text{act}}_{i,n}$, 动作 - 路径链的集合记为 $\Pi^{\text{mot}}_{i,n}$.

例2 考虑如图 1 所示的智能体模型和如图 3 所示环境模型 $\mathcal{T}$, 为了实现子任务 $(1, \{sw_{o_1}\}, \{\})$, 即在区域 o_1 执行动作 “提供水”(sw), 则智能体必须先执行动作 “获取水”(gw), 对应的动作链为 $[gw, sw]$. 考虑到 “水” 存在 r_1 区域, 对应的一种动作 - 路径链 $\pi^{\text{mot}}_{i,n}$ 为 $[(gw, r_1), (sw, o_1)]$.

给定子任务 ω_i 和智能体模型 E_n, 算法 1 给出了对应的动作 - 路径链的计算方法. 首先, 在第 1-9 行建立快速搜索框架以确定动作链集 $\Pi^{\text{act}}_{i,n}$. 即对于动作链 $\pi^{\text{act}}_{i,n}$, 从 E_n 的初始状态出发, 期间执行的动作满足任务 ω_i, 最终返回初始状态. 因此在第 1 行的初始化中, 要求从智能体初始状态 s_0 出发. 并且第 4 行要求遵循 E_n 中的转移约束, 最终回到初始状态 s_0, 因此动作链都符合动作模型 E_n. 第 6 行的 $\rho \vDash \sigma_i$ 和 $\rho \vDash \sigma^s_i$ 保证只有满足子任务的 σ_i, σ^s_i 的动作链才会被选入动作链集 $\Pi^{act}_{i,n}$ 中. 最终能遍历智能体状态 s 的所有转移路径. 然后在第 7~9 行中, 选取动作所对应的原子命题能满足子任务, 还使智能体状态返回到初始状态 s^n_0 的动作序列 π' 作为动作链. 同时如第 12~28 行所示, 在动作链集的基础上搜索动作 - 路径链集 $\Pi^{\text{mot}}_{i,n}$. 如第 13 行所示, 首先选取动作链中的任意动作所需要的相关资源的区域, 这一步能在保留必要信息的情况下大幅减小最终搜索空间. 然后在这些可移动的区域中, 在动作 - 路径链 π^m 中交替地尝试进行移动或执行任务. 其中, 当在第 22 行判断资源和动作在区域 W_m 相匹配时候, 算法会在第 23~25 行尝试在当前区域执行未完成的动作. 第 24 行的判断保证了只有满

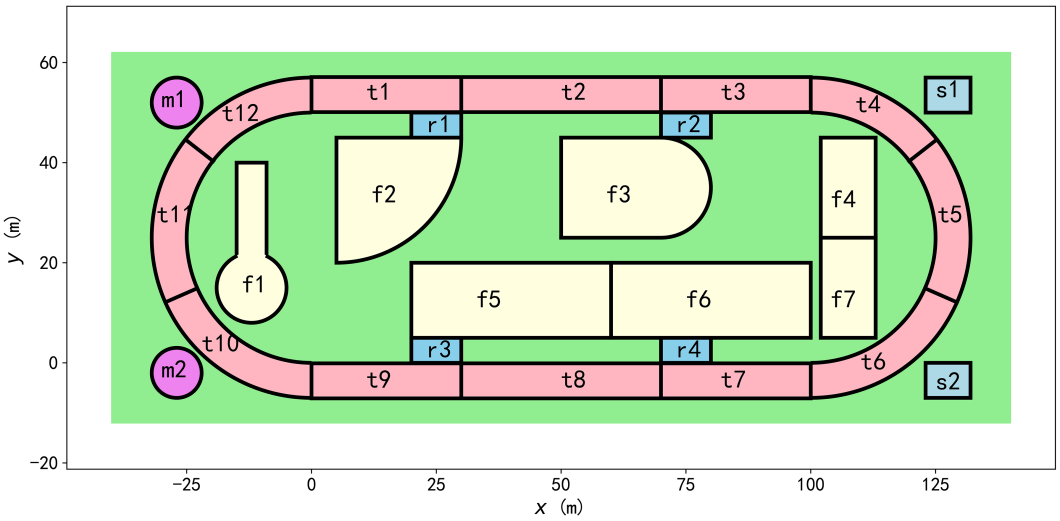

图 3　(网络版彩图) 智能赛场仿真环境

Figure 3　(Color online) Simulation environment of the "smart" track field

足子任务 σ_i 和 σ_i^s 的动作 - 路径链才会被加入动作 - 路径链集合 $\Pi_{i,n}^{\text{mot}}$ 中. 当 $\pi_{i,n}^{\text{mot}}$ 被执行完, 则子任务 ω_i 被完成. 因此, 由算法 1 得到的动作 - 路径链 $\pi_{i,n}^{\text{mot}}$ 能满足子任务 ω_i, 且符合智能体 n 的动作模型 E_n 和环境模型 $\mathcal{T}$ 的资源约束.

值得注意的是, 动作链 $\Pi_{i,n}^{\text{act}}$ 只与子任务的动作种类有关, 而动作 - 路径链 $\Pi_{i,n}^{\text{mot}}$ 和子任务的动作以及任务区域都有关. 特别地, 动作 - 路径链和智能体数量无关, 只和智能体种类有关, 因此计算量不会随着智能体数量增长而增长. 由于系统本身的动作类型和任务区域数量都是有限的, 这个算法可以提前将所有可能的动作 - 路径链遍历, 作为已知信息保留, 以减少规划时间. 但是当环境信息发生变化时, 则需要根据新的环境模型 $\mathcal{T}$ 重新计算动作 - 路径链 $\Pi_{i,n}^{\text{mot}}$.

基于动作链的偏序约束. 由于子任务由动作链中一连串的动作所替代, 拟偏序集中 P_φ 的偏序约束 $\preceq_\varphi, \neq_\varphi$ 将只约束动作链中, 而非其他的前置后置任务. 顺序约束 $\preceq_\varphi$ 在动作链对应的实际约束如下:

$$t_{i,m_1} \leqslant t_{j,m_2}, \forall(\omega_i, \omega_j) \in \preceq_\varphi, \tag{3}$$

其中, t_{i,m_1} 是动作链中满足 $\lambda(\pi_{i,n_1}^{\text{mot}}[m_1]) \models \sigma_i$ 的动作 $\pi_{i,n_1}^{\text{mot}}[m_1]$ 的开始时间, 而 t_{j,m_2} 是动作链中满足 $\lambda(\pi_{j,n_2}^{\text{mot}}[m_2]) \models \sigma_j$ 的动作 $\pi_{i,n_1}^{\text{mot}}[m_1]$ 的开始时间, n_1, n_2 是动作 - 路径链 $\pi_{i,n_1}^{\text{mot}}, \pi_{j,n_2}^{\text{mot}}$ 所对应的智能体序号. 冲突约束 $\neq_\varphi$ 在动作链对应的实际约束如下:

$$t_{i,m_1} + d(\pi_{i,n_1}^{\text{mot}}[m_1]) \leqslant t_{j,m_2}, \forall \Omega_{sub} \in \neq_\varphi, \exists \omega_i, \omega_j \in \Omega_{sub}, \tag{4}$$

其中, t_{i,m_1} 是动作链中满足 $\lambda(\pi_{i,n_1}^{\text{mot}}[m_1]) \models \sigma_i$ 的动作 $\pi_{i,n_1}^{\text{mot}}[m_1]$ 的开始时间, $d(\pi_{i,n_1}^{\text{mot}}[m_1])$ 为对应动作的执行时长.

3.2　基于动作链的任务分配

在获取到动作 - 路径链 $\Pi_{i,n}^{\text{mot}}$ 之后, 问题则转化为选择智能体来执行 $\pi_{i,n}^{\text{mot}}$, 并计算对应的执行时间. 即为满足环境资源约束, 运动约束, 同步约束和拟偏序集约束的组合优化问题. 考虑到动作 - 路径链具有逻辑意义上的连贯性, 需要同一智能体完整地执行所有的动作, 称为主智能体. 同时, 链中的协

算法 1 动作 - 路径链计算方法

输入 子任务 $\omega_i = (i, \sigma_i, \sigma_i^s)$, 智能体模型 E_n

输出 任务策略 $\Pi_{i,n}^{\text{mot}}$

搜索序列 $q = [(s_0^n, \pi = [], \rho = \{\})]$, 动作链集 $\Pi_{i,n}^{\text{act}} = \{\}$, 动作 - 路径链集 $\Pi_{i,n}^{\text{mot}} = \{\}$;
while q **do**
  $s, \pi, \rho = q.pop()$
  for $(s, a, s') \in \rightarrow_n$ **do**
    $\pi' = [\pi, a]$, $\rho' = \rho \cup \lambda_n(a)$
    if $\rho' \vDash \sigma_i, \rho' \vDash \sigma_i^s, s' = s_0$ **then**
      将 π' 加入到 $\Pi_{i,n}^{\text{act}}$
    else
      将 (s', π, ρ') 加入 q 中
    end if
  end for
  for $\pi^a \in \Pi_{i,n}^{\text{act}}$ **do**
    $\mathbf{W}_i = \{W_m | C_{\mathcal{A}}(a) \in R_m, \forall a \in \pi^a\}$,
    $q = [(W_m, 0, [(\varnothing, \mathrm{W}_m)])]$
    while q **do**
      $W_m, j, \pi^a = q.pop()$
      if $\pi^a[-1][1] \neq \varnothing$ **then**
        for $W_i \in \mathbf{W}_i, W_i \neq W_m$ **do**
          $(W_i, j, [\pi^a, (\varnothing, W_i)])$ 加入 q
        end for
      end if
      if $\pi^a[j]$ 能在 W_m 中执行:# 计划下一步动作 **then**
        if $j + 1 = |\pi^a|$ **then**
          if $\lambda_n(\pi^m) \vDash \sigma_i^s, \lambda_n(\pi^a[j], W_m) \vDash \sigma_i$ **then**
            将 $[\pi^a, (\pi^a[j], W_m)]$ 加入 $\Pi_{i.n}^{\text{mot}}$
          end if
        else
          $(W_m, j + 1, [\pi^a, (\pi^a[j], W_m)])$ 加入 q
        end if
      end if
    end while
  end for
end while
return $\Pi_{i,n}^{\text{mot}}$

作动作可以由其他智能体协同以减少执行时间长, 这些协同的智能体被称为协作智能体. 因此, 智能体 n_1 的动作序列 $J_{n1} \in \mathcal{J}$ 可以定义为

$$J_{n_1} \triangleq [\pi_{i,n_2}^{\text{mot}}[k], \cdots, \pi_{i,n_1}^{\text{mot}}], \tag{5}$$

其中 $\pi_{i,n_2}^{\text{mot}}[k] \triangleq (a_k, W_k)$ 是链中的元素, 表示智能体 n_1 将在区域 W_k 执行动作 a_k, 以协助完成智能体 n_2 的动作 - 路径链 $\pi_{i,n}^{\text{mot}}$. 其中 π_{i,n_1}^{mot} 则表示智能体 n_1 完整执行动作链 $\pi_{i,n}^{\text{mot}}$. 值得注意的是, 主 - 协作智能体的协作模式是智能体动作模型 E_n 的约束所要求的. 即在执行动作 a 前需要主智能体的当前状态 s 能够执行对应的动作, 即存在 $s' \in S^n$ 使得 $(s, a, s') \in \rightarrow_n$.

所提出的分支定界法的计算架构如图 4 所示, 算法的核心在于构建并维护解空间的搜索树, 不断

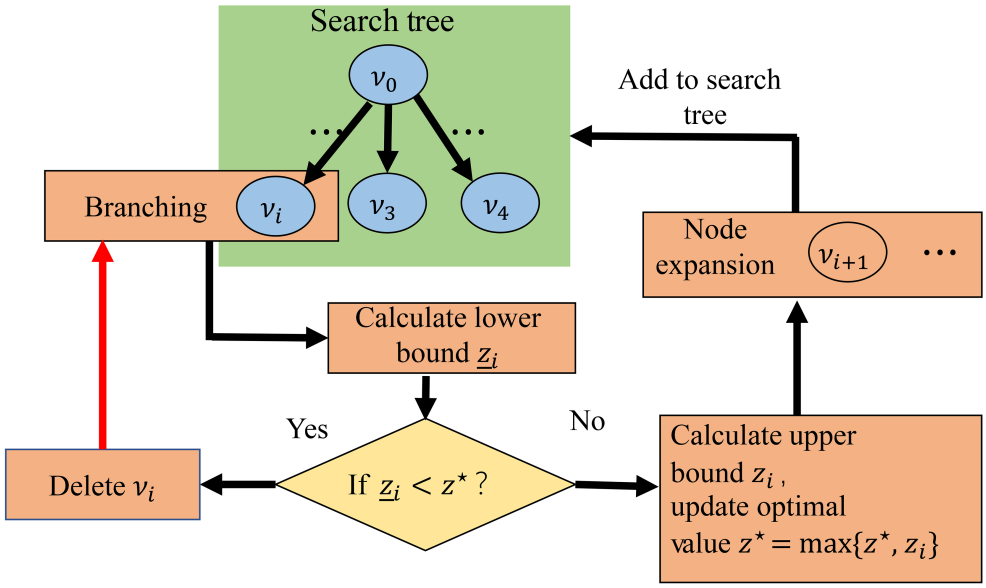

图 4 (网络版彩图) 分支定界法算法架构

Figure 4 (Color online) Architecture of the modified branch and bound algorithm

选择更有潜力的节点和分支进行拓展, 并减除下界过高的分支, 记录最好的上界. 由于任务序列 $\mathcal{J}$ 中主智能体需要完整执行动作 - 路径链, 而协作智能体只需协助单一动作, 之前的方法会生成大量不可行的子节点. 改进了节点的拓展方法, 以保证生成的子节点都是可行的. 同时, 并针对复合的优化目标设计了增量式的分支选择方法以加速计算.

3.2.1 基于动作路径链协同的节点拓展

搜索树的节点 $\nu_l \triangleq [\tau_1, \cdots, \tau_n]$ 内的元素与式 (5) 一致, 但为部分任务分配的结果. 初始节点没有分配任何任务, 即 $\nu_0 \triangleq [[], [], \cdots]$. 搜索树中, 没有指向其他节点的节点被称为叶节点, 生成某个叶节点所有可行的子节点的步骤被称为节点拓展. 对于一个节点 ν_l, 节点拓展可以分为两种情况: 添加一个未分配的子任务的可行的动作 - 路径链 $\pi^{\text{mot}}_{i,n}$; 或协同上一个分配的动作 - 路径链中的一个动作 $\pi^{\text{mot}}_{i,n}[k]$.

动作 - 路径链分配. 由于动作 - 路径链间存在偏序约束, 并且偏序约束对子任务执行顺序有要求, 首先可以对当前节点可分配的子任务进行如下的限制:

$$\Omega_f \triangleq \{\omega_k | \forall(\omega_j, \omega_k) \in \prec, \omega_j \in \Omega_l\}, \tag{6}$$

式中, Ω_l 为当前节点 ν_l 所有的已分配的任务. 每次分配只选择 Ω_f 中的子任务能保证子节点的任务顺序能一定满足偏序关系 $\prec$ 的约束. 在选取一个待分配的子任务 $\omega_i \in \Omega_f$ 后, 计算满足偏序关系的动作 - 路径链集:

$$\Pi^{Par}_i \triangleq \{\pi^{\text{mot}}_{i,n} | \lambda_n(\pi^{\text{mot}}_{i,n}) \vDash \Sigma^s_{fea}, \forall n \in \mathcal{N}\}, \tag{7}$$

其中, $\lambda_n(\pi^{\text{mot}}_{i,n})$ 表示动作 - 路径链对应的符号, $\Sigma^s = \bigcup_{\omega_i \in \Omega_f} \sigma^s_i$, 是所有当前可行的子任务的前置需求. 同时考虑已有动作对环境资源的消耗, 避免整体计划使用的资源数超过了当前区域的资源量:

$$\Pi^f_i \triangleq \{\pi^{\text{mot}}_{i,n} | C^m_{\mathcal{A}}(\mathcal{J}) + C^m_{\mathcal{A}}(\pi^{\text{mot}}_{i,n}) \leqslant R_m, \forall \pi^{\text{mot}}_{i,n} \in \Pi^{Par}_i, \forall W_m \in \mathcal{W}\}, \tag{8}$$

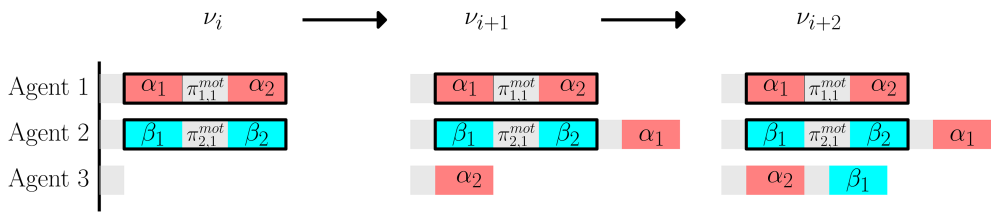

图 5 (网络版彩图) 动作链协同时的死锁情况

Figure 5 (Color online) Deadlock during the assignment of action chains

其中, $C^m_{\mathcal{A}}(\mathcal{J})$ 表示当前分配方案在区域 W_m 所消耗的环境资源, $C^m_{\mathcal{A}}(\pi^{\text{mot}}_{i,n})$ 表示此动作 - 路径链在区域 W_m 所消耗的环境资源. 只选择 Π^f_i 中的链进行添加, 能保证未执行的子任务的前置需求得到满足.

动作协同. 在搜索动作链中所需的协同智能体时, 若随机的选取已经分配的动作进行协同, 可能会出现不可行的死锁情况. 如图 5 所示, 对于已经分配了两个动作 - 路径链 $\pi^{\text{mot}}_{1,1}, \pi^{\text{mot}}_{2,1}$ 的节点 ν_i, 若先寻找 $\pi^{\text{mot}}_{1,1}$ 中动作的协同智能体, 再寻找 $\pi^{\text{mot}}_{2,1}$ 的协同智能体, 则可能会生成节点 ν_{i+2}. 而节点 ν_{i+2} 中含有两个冲突的动作时序约束 $\alpha_2 < \beta_1, \beta_1 < \alpha_2$, 从而产生死锁. 尽管这些节点可以在下界计算过程中判断为不可行, 但会带来额外的计算开销.

因此, 为避免不同链之间的协同动作发生死锁, 可只允许分配最新的动作 - 路径链 $\pi^{\text{mot}}_{i,n}$ 中未分配的动作 $\pi^{\text{mot}}_{i,n}[k]$ 给其他智能体, 这个可行智能体集合 $\mathcal{N}^f_{i,n}$ 同时受到其余动作的前置需求的约束:

$$\mathcal{N}^f_{i,n} \triangleq \{n' | \lambda_{n'}(\pi^m_{i,n}[k]) \vDash \Sigma^s_{fea}, n' \in \mathcal{N}, n' \neq n\}, \tag{9}$$

以上分配模式可以保证节点拓展生成的任务序列一定是可行的, 从而避免对不可行的节点进行精确计算, 提高执行效率.

3.2.2 增量式分支选择

在拓展节点后, 会选择最有可能获取到最优值的叶节点开始新一轮的循环, 这个选择的过程被称为分支选择. 由于每次节点拓展中都会获得多个叶节点, 需要设置排序指标来评估叶节点获取最优解的潜力. 即优先选择执行效率高, 且已剩余任务少的节点. 这里设置了增量式的评估方法, 即该评估值能基于父节点的评估值进行迭代计算, 以保证计算效率.

具体来讲, 对于空的初始节点 ν_0, 它的评估值设置为 $h_0 = 0$. 对于非空叶节点 ν_l, 它的父节点 ν_r 的评估值为 h_r. 若当前节点基于父节点分配了一个新的动作链 $\pi^{\text{mot}}_{i,n}$, 那么对应的评估值为

$$h_l \triangleq (h_r z_y + \alpha t(\pi^{\text{mot}}_{i,n}) + \beta c_r(\pi^{\text{mot}}_{i,n}) + \gamma c_t(\pi^{\text{mot}}_{i,n}))/z_r, \tag{10}$$

其中 z_r 是父节点的上界值, z_y 是双重父节点的上界值, h_r 是父节点的评估值. $t(\pi^{\text{mot}}_{i,n})$, $c_r(\pi^{\text{mot}}_{i,n})$, $c_t(\pi^{\text{mot}}_{i,n})$ 分别为相对父节点新分配的时间, 资源和路径消耗. 若当前节点 ν_l 是基于父节点 ν_r, 将动作链中的动作 $\pi^{\text{mot}}_{i,n}[k]$ 分配更多的协作智能体, 其评估值可计算为

$$h_l \triangleq (h_r z_y + \gamma c_t(\{\pi^{\text{mot}}_{i,n}[k]\}))/z_r, \tag{11}$$

因为, 协作动作不会添加额外的资源消耗, 且任务的协同总时长不变, 因此 (11) 式不需要计算 $\alpha t(\pi^{\text{mot}}_{i,n}[k])$ 和 $\beta c_r(\pi^{\text{mot}}_{i,n}[k])$.

以上分配方法的正确性可简略证明如下. 首先, 式 (7) 要求只有满足当前可行的任务集 Ω_f 的所有前置动作要求 Σ^s 的动作 - 路径链才会被选择, 即对所有 $(\omega_i, \omega_j) \in \prec_f$, 任务 σ^s_j 总会被 $\pi^{\text{mot}}_{i,n}$ 满足,

算法 2 动作 - 路径链计算

输入 更新后的拟偏序集 p, 未分配的子任务集 Ω_u, 已执行的子任务集 Ω_{finish}
输出 任务策略 $\Pi_{i,n}^{\text{mot}}$

1: 根据环境信息更新 Ω_u 的动作 - 路径链
2: 初始化温度 T, 设置初始节点为上一轮的解 $\mathcal{J}$
3: **while** 动态截止条件式 (14) 未满足 **do**
4: 若 $\Omega_{un} = \varnothing$, 从已分配的任务从取消一个链加入 Ω_{un}
5: **for** $(s, a, s') \in \rightarrow_n$ **do**
6: $\pi' = [\pi, a]$, $\sigma' = \sigma \cup \lambda_n(a)$
7: 从 Ω_{un} 中随机选取任务, 根据式 (15),(16) 找到邻域解 $\mathcal{J}'$
8: 根据温度 T 条件判断是否接受新的解
9: 根据式 (13) 进行变速率温度衰减
10: **end for**
11: **end while**
12: **return** 最优解

因此 $\mathcal{J}$ 能满足所有子任务的前置任务需求. 其次, 在每次节点拓展中, 子任务 ω_k 能被选择分配当且仅当所有需要在其之前执行的任务 ω_j 已经被分配了, 即式 (6) 中对于所有 $(\omega_j, \omega_k) \in \preccurlyeq$ 都有 $\omega_j \in \Omega_l$. 因此 $\preccurlyeq_f$ 的约束总是被满足. 另外约束 $\{\omega_k\} \in \neq_f$ 要求子任务不在同一时刻执行, 这一步会通过 4.2 节中的同步机制保证. 因此, $\mathcal{J}$ 的任务执行顺序能够满足 $\preccurlyeq_f$ 和 $\neq_f$ 约束, 并且执行方案满足任务公式.

4 在线适应

在线适应需要考虑两个关键问题: 当新任务在线生成时, 如何修改当前的任务序列; 由于移动和动作执行时间的不确定性, 智能体无法严格按照计算的时间执行. 因此提出了两个机制来保证在线适应能力: 1) 针对新任务公式, 利用局部搜索方法对任务序列进行更新; 2) 设计了基于动作 - 路径链的同步方案, 以保证在环境不确定性下任务执行的同步过程.

4.1 局部搜索

针对在线发布的新任务 φ_u, 设计了基于模拟退火的局部搜索算法来更新任务序列. 首先, 利用文献 [27] 所提出的偏序乘法, 将新任务转化为拟偏序集 P_u 后和原有偏序 P 相乘. 在获取到更新的 P 后, 基于之前的分配结果 $\mathcal{J}_0$ 对未分配的子任务 $\Omega_{un} \in \Omega$ 进行分配. 相比于经典的模拟退火算法 [28], 在算法 2 做出了两个核心的改动: 考虑计算时间 t_c 和收敛速率的变温度衰减系数 (第 9 行) 和截止条件 (第 3 行); 设计基于结构特征的邻域计算 (第 7 行).

变衰减系数. 设置了一个和优化值有关的衰减系数, 使算法能基于获取更优值的期望表现出不同的探索行为. 考虑当前重规划的计算时间 t_{re} 的优化目标函数为

$$z' \triangleq \alpha(t_m + t_c) + \beta c_r + \gamma c_t \triangleq z + \alpha t_c, \tag{12}$$

并且第 ℓ 次的衰减系数可以写为

$$T_\ell = T_{\ell-1} z^* / (z^* + \alpha t_c), \tag{13}$$

其中, z^* 为当前的最优值, αt_c 为加权下的计算时间消耗. 因此, 当计算消耗 αt_c 相对较小时, 温度会维持高位以保持较高的探索能力; 反之, 当 αt_c 相对 z^* 较大时, 温度会快速衰减, 使得结果在当前区域收敛到局部最优解.

动态截止条件. 当继续计算的时间消耗大于潜在目标函数下降的收益时, 算法截止. 即以下条件:

$$\alpha\Delta t_\ell - S_\ell > 0, \tag{14}$$

其中, S_ℓ 为当前平均下降速率, 更新方式为 $S_{\ell+1} = ((n-1)S_\ell + \Delta z_l)/n$, Δt_ℓ 是当前轮迭代的计算时间. 综合以上动态判别方式, 算法能在不同的优化目标之间表现出较好的适应性.

邻域计算: 对于一个部分解 $\mathcal{J}'_\ell$, 向其加入未分配的子任务 $\omega_i \notin \Omega_f$ 需要考虑到主, 协作智能体的结构, 以及 $\preceq_f, \neq_f$ 的时序约束. 否则随机的分配任务同样会生成大量存在任务次序冲突的不可行解. 对于一个智能体的任务序列 $J_n \in \mathcal{J}'_n$, $J_n \triangleq [\cdots, \pi^{\rm mot}_{j_1,n}, \pi^{\rm mot}_{j_2,n}, \cdots]$ 会带来新的时序约束, 即子任务 ω_{j_2} 需要在 ω_{j_1} 之后执行. 因此, 首先计算由于 $\mathcal{J}'_\ell$ 中分配关系所导致的时序关系:

$$\preccurlyeq_\ell \triangleq \preccurlyeq \cup \big\{(j_1, j_2) | \pi^{\rm mot}_{j_1,n} = J_n[k_1], \pi^{\rm mot}_{j_2,n} = J_n[k_2], \forall k_1 < k_2, \forall n \in \mathcal{N}\big\}, \tag{15}$$

然后在 $\mathcal{J}_\ell{}'$ 搜索所有满足偏序约束 $\preccurlyeq_\ell$ 的可插入次序 $[(n,k)]$, 表示可以将此任务插入到智能体 n 的任务序列的第 k 个执行:

$$\begin{aligned} &(j,\, i) \in \preccurlyeq_\ell,\, \pi^m_{j,n} = J_n[k_1],\, \forall k_1 < k,\, \forall n \in \mathcal{N}, \\ &(i,\, j) \in \preccurlyeq_\ell,\, \pi^m_{j,n} = J_n[k_2],\, \forall k_2 > k,\, \forall n \in \mathcal{N}, \end{aligned} \tag{16}$$

在从 $[(n,k)]$ 随机选取 n, k 后, 可类似于式 (8), 从 $\Pi^{\rm mot}_{i,n}$ 中选取满足环境约束的动作 - 路径链:

$$\Pi^f_{i,n} \triangleq \{\pi^{\rm mot}_{i,n} | C^m_{\mathcal{A}}(\mathcal{J}) + C^m_{\mathcal{A}}(\pi^{\rm mot}_{i,n}) \leqslant R_m, \forall \pi^{\rm mot}_{i,n} \in \Pi^{\rm mot}_{i,n}, \forall W_m \in \mathcal{W}\}, \tag{17}$$

最终选择 $\pi^{\rm mot}_{i,n} \in \Pi^f_{i,n}$ 加入到 J_n 的第 k 个位置, 完成动作 - 路径链和主智能体的选择. 然后按顺序对链 $\pi^{\rm mot}_{i,n}$ 中的动作从 $[(n,k)]$ 中寻找协作智能体, 并且在加入新动作之前, 用 (15) 式更新 $\preccurlyeq_\ell$, 并用 (16) 式更新 $[(n,k)]$.

以上自适应算法的正确性可简要证明如下. 在采用算法 2 获取邻域解时, 对于未加入的子任务 ω_i, 其对应的可加入组合 $[(n,k)]$ 即作为智能体 n 的第 k 动作执行, 可由式 (15) 和式 (16) 确定. 其中, 式 (15) 同时考虑拟偏序集原有的顺序关系以及由于分配方案导致的顺序关系, 从而避免由于协作任务导致的死锁. 式 (16) 所计算的组合 $[(n,k)]$ 能保证子任务 ω_i 既满足已有的顺序关系, 且不会添加新的带有冲突的顺序关系. 因此, 最终的分配方案 $\mathcal{J}$ 满足拟偏序集的顺序关系, 从而满足任务公式, 且执行时不会发生死锁.

4.2 在线同步

对于动作 - 路径链 $\pi^{\rm mot}_{n,i}$, 若 $\pi^{\rm mot}_{n,i}[h:(h+1)]$ 是能满足子任务 ω_i 中 σ_i 的核心动作, 需要遵循偏序约束 $\preccurlyeq_f, \neq_f$. 同时, 链中的每个动作若有多个智能体参与, 则需要一个连续的时间段完成. 由于环境的不确定性, 通常智能体无法严格按照计算的时间到达目标区域执行动作. 这会导致协同任务执行时间与偏序约束 $\preccurlyeq_f, \neq_f$ 发生冲突. 因此, 基于动作链中主、协智能体的区别提出了在线同步机制.

如图 6 所示, 在线同步方式遵循以下原则: 1) 主智能体不需要考虑协作智能体的状态, 2) 主智能体需要考虑核心动作的偏序条件是否满足, 3) 协作智能体不考虑偏序条件, 需要等待主智能体开始执行后才能执行对应任务, 4) 若协作智能体到达时对应任务已经完成, 则直接转入执行下一个任务.

具体来讲, 当主智能体 n 到达目标区域准备执行动作 $\pi^{\rm mot}_{i,n}[k]$, 若该动作是核心动作即满足 $\lambda(\pi^{\rm mot}_{i,n}[k]) \models \omega_i$, 则必须在所有前置任务 ω_j 开始前等待, 其中 $(\omega_j, \omega_i) \in \preceq$. 而且在所有非并行任务 ω_k 执行时等待, 其中 $(\omega_k, \omega_i) \in \neq$. 另外, 若该动作不是核心动作, 则可以直接开始执行. 通过该同步过程, 偏

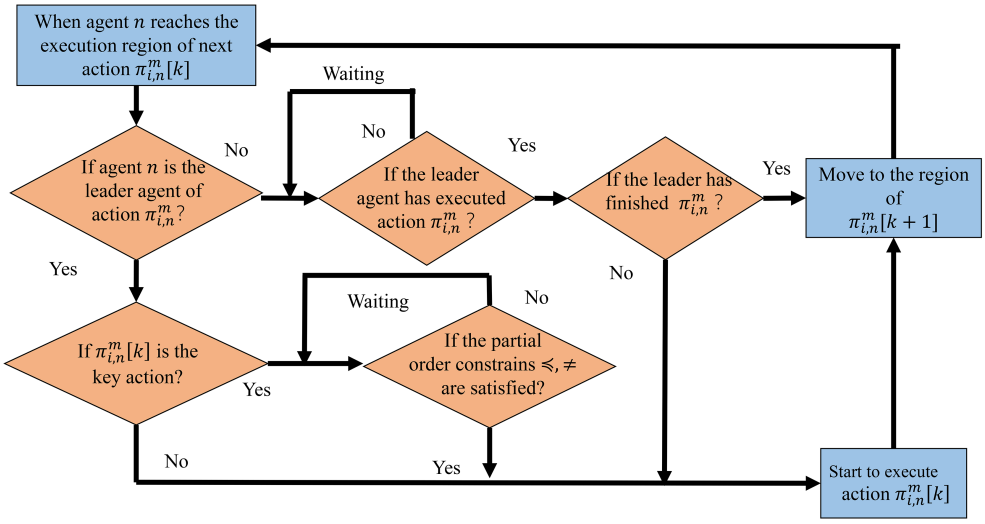

图 6　(网络版彩图) 在线同步算法图

Figure 6　(Color online) Diagram of the online synchronization algorithm

序约束 $\preceq$ 和 $\neq$ 被同时满足. 可见该算法不严格要求协作智能体和主智能体同时开始任务. 主智能体能在协作智能体未到达的情况下直接开始, 且在任务的执行过程中允许协作智能体在线加入. 若当协作智能体集群在对应动作完成时才到达, 则会直接加入下一个动作. 因此, 智能体集群可在不进行严格同步的情况下实现任务协作与协同, 同时满足偏序集约束.

在通信方面, 主智能体之间需要全局通信, 进行任务协同和在线同步. 而协作智能体只需在到达任务目标区域后与主智能体进行局部通信. 此外, 该算法不要求主智能体间存在完全通信网络, 即负责动作 - 路径链 $\pi_{i,n}^{\mathrm{mot}}$ 的主智能体 n, 只需和有相关时序关系动作链的负责主智能体进行通讯, 这些相关的智能体集 $\mathcal{N}_n$ 可以通过任务方案 $\mathcal{J}$ 计算, 即

$$\mathcal{N}_n \triangleq \{n'|\pi_{j,n'}^{\mathrm{mot}} \in \mathcal{J}[n'], \forall((\omega_i,\omega_j) \vee (\omega_j,\omega_i)) \in \preceq_f\} \cup \{n'|\pi_{j,n'}^{\mathrm{mot}} \in \mathcal{J}[n'], \forall(\omega_i,\omega_j) \in \neq_f\}, \tag{18}$$

其中第一部分为偏序约束相关任务的负责主智能体, 第二部分为互斥约束相关任务的负责主智能体.

4.3　复杂度分析

算法 1 针对智能体类型 E 计算单个子任务的动作链的复杂度为 $O(|E|^2)$, 其中 $|E|$ 为智能体动作模型的状态数量; 动作 - 路径链的复杂度为 $O(|E|^4|\mathcal{T}|^2)$, 其中 $|\mathcal{T}|$ 是环境模型的状态数量. 因此, 在有 L 种智能体的集群中, 计算子任务数量为 n 的拟偏序集的动作 - 路径链的复杂度为 $O(nL|E|^4|\mathcal{T}|^2)$. 值得注意的是, 复杂度只与智能体种类有关, 而与智能体数量无关, 因此算法 1 的复杂度不会随着智能体数量而增长. 在分配过程中, 分支定界法的首次解的复杂度为 $O(NMC)$, 其中 N 为智能体数量, M 为子任务数量, C 为最长的动作链的长度. 在线添加 W 个子任务后, 局部搜索获得首次可行解的计算复杂度为 $O(NMW)$. 因此对于有 m 个最大长度为 n 的任务公式集, 整体框架的计算流程为: 计算子公式的拟偏序集 $O(mn\cdot 2^n + m\cdot n!)$, 计算偏序乘积 $O(n^2m^3)$, 计算子任务的动作 - 路径链 $O(nmL|E|^4|\mathcal{T}|^2)$, 计算任务分配 $O(nmNC)$. 和第 4 章算法的计算复杂度相比, 主要的增加项是 $O(nmL|E|^4|\mathcal{T}|^2)$, 该项为多项式复杂度, 且与智能体数量无关, 因此考虑复杂动作模型并不会显著增加算法的计算时间.

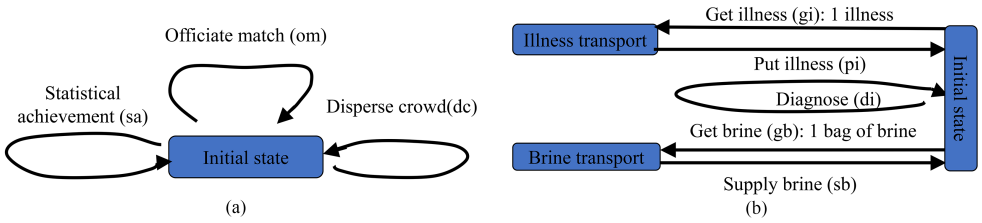

图 7 (网络版彩图) 裁判智能体 (a) 和运动医生智能体 (b) 模型

Figure 7 (Color online) Agent model of the referee (a) and the sport doctor (b)

5 仿真对比

5.1 仿真背景

设计了一个智能赛场环境. 其中有三类智能体: “志愿者” “裁判” “运动医生”. 志愿者智能体的模型如图 1 所示, “裁判” “运动医生” 模型如图 7(a), (b) 所示, 它们的一部分动作如 “诊断 (di)” 不需要消耗资源, 另一部分动作如 “获取盐水 (gs)” 则会消耗对应的资源, 这些智能体, 动作和资源的详细定义见表 A1. 赛场环境为如图 3 所示体育场, 其中红色区域 $t1,\cdots,t12$ 为径赛区, $r1,\cdots,r4$ 为服务区, $m1,m2$ 为材料区, $f1,\cdots,f7$ 为田赛区, $s1,s2$ 为储存区. 环境中有 4 种资源, 水, 盐, 盐水和伤员. 其中在区域 $s1$ 有 10 份水, 5 份盐, 10 份食物, 在区域 $s2$ 中, 有 5 份水, 2 份盐水, 5 份食物. 以下所有的计算是在一个 16 核, 2.6 GHZ, 32 GB 内存的电脑中完成的.

5.2 案例

在表 A2 中给定一系列任务公式, 这些公式没有显式地写出需要执行的动作的前置以及后置需求, 需要算推理得到. 其中, $\varphi_1\sim\varphi_5$ 为事先给定的离线任务, 表示确定的任务要求. $\varphi_6\sim\varphi_{10}$ 为在线任务, 下标对应的区域会在线确定. 按照复杂度由低到高, 设计了 4 个仿真案例以验证算法的性能和效果.

案例 1: 离线发布 φ_1,φ_2; 在 30 s 发布 φ_6; 智能体集群由 8 个 “志愿者”, 6 个 “裁判”, 3 个 “运动医生” 构成; 优化参数: $\alpha=1,\beta=3,\gamma=3$.

案例 2: 离线发布 $\varphi_1,\varphi_2,\varphi_3$; 在 100 s 发布 φ_6, 在 280 s 发布 $\varphi7$. 智能体集群由 8 个 “志愿者”, 6 个 “裁判”, 3 个 “运动医生” 构成. 优化参数: $\alpha=0.5,\beta=0.1,\gamma=10$.

案例 3: 离线发布 $\varphi_1,\varphi_2,\varphi_3,\varphi_4,\varphi_5$; 在 80 s 发布 φ_6, 在 180 s 发布 $\varphi7$, 在 400 s 发布 φ_{10}. 智能体集群由 16 个 “志愿者”, 12 个 “裁判”, 6 个 “运动医生” 组成. 优化参数: $\alpha=1,\beta=3,\gamma=3$.

案例 4: 离线发布 $\varphi_1,\varphi_2,\varphi_3,\varphi_4,\varphi_5$; 在 80 s 发布 φ_6, 在 180 s 发布 φ_7, 在 400 s 发布 φ_9. 智能体集群由 32 个 “志愿者”, 24 个 “裁判”, 12 个 “运动医生” 组成. 优化参数: $\alpha=0.5,\beta=0.1,\gamma=10$.

5.3 仿真结果

首先详述中等复杂的案例 2 的仿真结果, 如图 8 所示, 这是 400 s 时刻的任务路径图. 尽管在任务 φ_1 中, 只要求在 $r1$ 执行提供水 (sw) 的动作, 但是算法根据智能体模型计算得到了动作链: $[gw,sw]$, 需要先执行的获取水 (gw) 动作, 以将初始状态移到 “运输水” 状态. 并根据环境的资源分布, 生成了动作 - 路径链: $[(gw,s1),(sw,r1)]$, 选择了存在水的 $s1$ 区域执行获取水 (gw) 动作.

特别地, 当环境区域出现伤员时, 对算法提出了任务 φ_6, 要求执行动作获取伤员 (gi). 动作链 $[gi,pi]$ 要求在执行 gi 后, 额外执行放置伤员 pi 的动作. 根据更新后的环境信息, 计算得到了动作 - 路

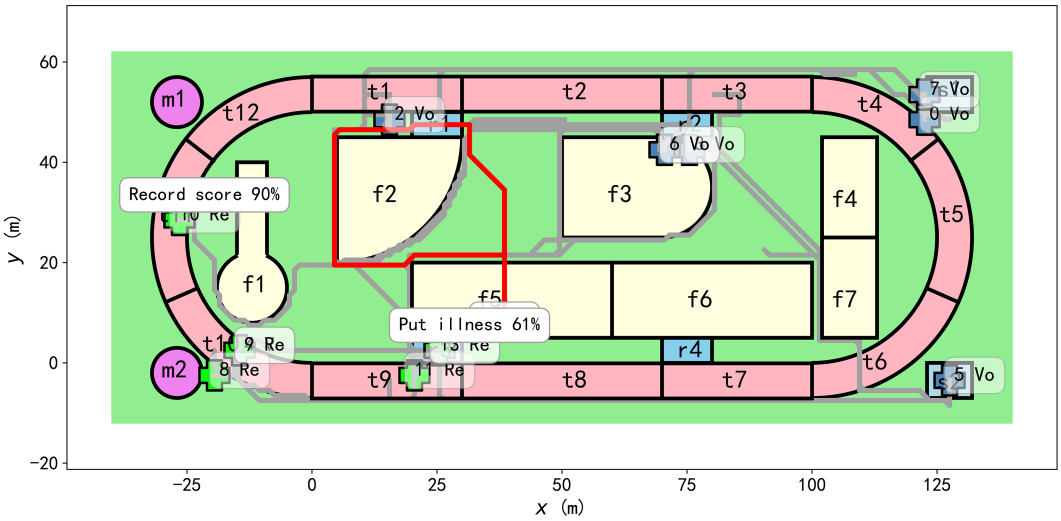

图 8 (网络版彩图) 案例 2 在 400 s 时的执行路径

Figure 8 (Color online) Trajectories of all agents in Case 2 at 400 s

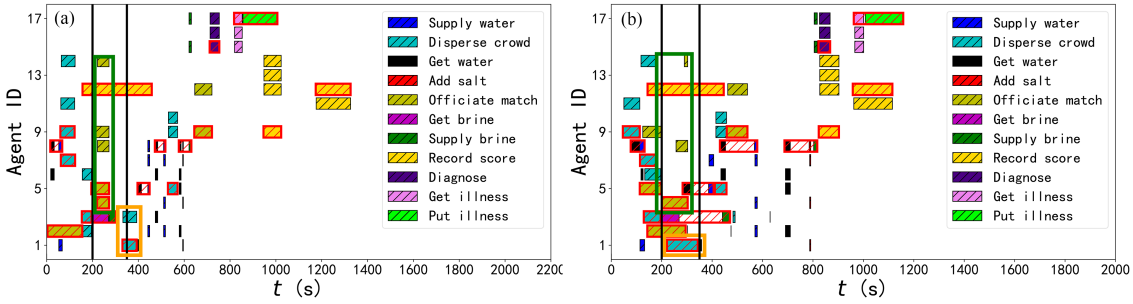

图 9 (网络版彩图) 任务的计算时间 (a) 和真实执行时间 (b) 的甘特图

Figure 9 (Color online) Gantt charts of the planned execution (a) and the actual execution (b)

径链为 $[(gi, f6), (pi, r3)]$, 如图中特别标红的路径所示. 偏序乘法最终分析得到了 19 个子任务, 考虑算法 2 计算得到的动作 - 路径链后, 拓展到了 26 个动作, 并最终在分支定界法中确定动作的协同数量, 最终智能体执行了 64 次协同行为.

任务执行时间的计算结果和真实执行结果如图 9 所示. 离线算法计算了 27.71 s, 其中增量式更新的方法节约了 2% 的计算时间. 每一个动作链对应多个动作, 其中主智能体将完整执行, 而协作智能体只协作单个动作. 主智能体所对应的动作链被红色框框出. 在 200 s 和 350 s 处, 在线添加两次任务, 局部搜索算法基于当前的执行情况和之前的分配结果进行了重新计算. 重规划平均耗时 7.09 s, 最终得到的最大执行值. 与计算值相比, 由于路径规划等的不确定性, 真实的到达时间出现一定的波动, 但在线同步方案很好地保证了任务的同步性. 在图中被绿色框出的灰绿色 “主持比赛” 动作中, 由于 4 号主智能体相比其他智能体提前到达, 它提前开始执行任务, 最终它的真实执行时间比计划时间长. 相反, 智能体 8, 14 的真实执行时间比计划时间更短了. 对于橙色框出的浅蓝色 “疏散人群” 动作, 在智能体 3 到达之前, 主智能体 1 已经工作了足够的时间. 因此动作视为提前完成, 智能体 3 直接去执行下一个任务.

图 10(a) 记录了案例 2 的局部搜索的过程, 可以看到, 由于当前温度较高, 算法很容易接受一个

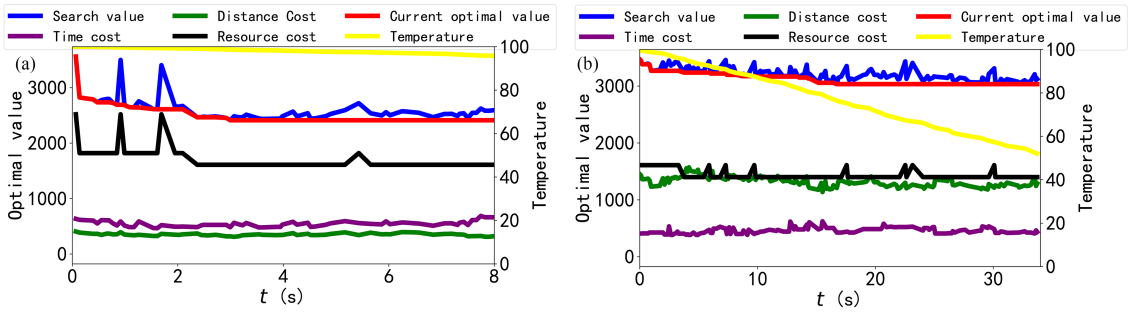

图 10 (网络版彩图) 案例 2 (a) 和案例 4 (b) 的模拟退火搜索过程

Figure 10 (Color online) Searching results of simulated annealing search process in Case 2 (a) and Case 4 (b)

较差的结果, 并尝试了多个领域. 但在 3 s 后的搜索中, 都没有找到更优的解, 优化值下降速率很慢. 因此, 动态截止条件判断继续计算获得更优值的收益要小于时间消耗对目标值的影响, 从而提前结束. 相反, 图 10(b) 为案例 4 的模拟退火计算过程, 由于下界速率超过时间增长, 算法选择以更多的计算时长换取更好的结果. 同时, 在开始阶段, 计算消耗相比最优值很小, 温度下降的速度慢, 算法进行了更大范围的探索. 不同案例间计算时间的变化表明, 所提出的动态截止条件能灵活选择合适的搜索时间. 在计算时间 30 s 左右时, 搜索时间和最优值的比值到达了 1.5%, 模拟退火的温度参数下降速度加快, 使得算法快速收敛到当前的最优解. 这也表明动态温度下降能根据目标函数合理调节算法的收敛速度.

5.4 可拓展性分析

在表 1 中记录了每个案例的数据. 可以注意到, 随着任务公式的增加, 最终需要执行的子任务数量也随之增加, 动作数量总是超过子任务数量, 这即是由智能体模型得到的动作链推理添加的动作. 随着智能体数量的增加, 任务复杂度的增加, 离线和在线的计算时间都随之增加. 和案例 3 相比, 在最复杂的案例 4 中设置了 68 个智能体. 离线规划耗时 1680.3 s, 增量式的分支选择方法降低了约 4% 的计算时间. 局部搜索的单次求解时间只随着复杂度线性增长, 但重规划的耗时增长较快. 案例 4 的重规划耗时 107.1 s, 远大于案例 3 的 18.35 s, 但是获取到了更优的优化目标. 这是由于设了动态退火和动态截止条件, 算法选择更多的计算时间以获取更优值. 最后, 动作链的计算时间是由于智能体种类和环境的复杂度决定的, 与智能体规模大小和任务公式长度无关. 因此每个案例的计算耗时均为 0.01 s 左右, 表明了基于动作链的算法既能保留复杂模型下的推理能力, 又不会增加优化算法的复杂度, 这与第 4.3 节的结论相符.

5.5 算法对比

将所提出的方法与文献中几种最常见的方法进行了比较. 具体来说, 比较了以下四种方法. 基础乘法 [13]: 处理多智能体问题的标准解决方法. 计算所有智能体模型与环境模型以及自动机模型的乘法自动机, 然后搜索可行路径. 混合整数规划 [22,29]: 将拟偏序集的分配问题建模为混合整数规划的优化解决方案. 偏序关系通过优化函数中的约束表示, 使用开源求解器 GLPK [30]. 采样 [23]: 不计算完整乘积自动机的模型, 而基于采样策略, 在智能体模型中选取和任务公式相关的动作, 并在环境模型中选取和任务公式相关的区域, 逐步构建相关的搜索空间. 可分解集 [31]: 将 NBA 拆分为可并行的部分, 并找到并行的节点作为可分解集. 构建每一个智能体和 NBA 的乘积自动机, 并利用可分解集连接这

表 1　仿真结果

Table 1　Simulation results

	Case 1	Case 2	Case 3	Case 4
Number of tasks	9	19	24	26
Number of actions	11	26	33	36
Optimal value	1227.8	2617.7	4657.7	3197.3
Time of offline calculation	10.12 s	27.71 s	79.19 s	1680.3 s
Time saved by incremental computation	0.208 s	0.518 s	1.732 s	55.31 s
The computation time of replanning	6.320 s	14.18 s	18.35 s	107.1 s
The computation time of the first solution	0.023 s	0.026 s	0.065 s	0.073 s
The calculation time of action-chain	0.012 s	0.011 s	0.019 s	0.012 s

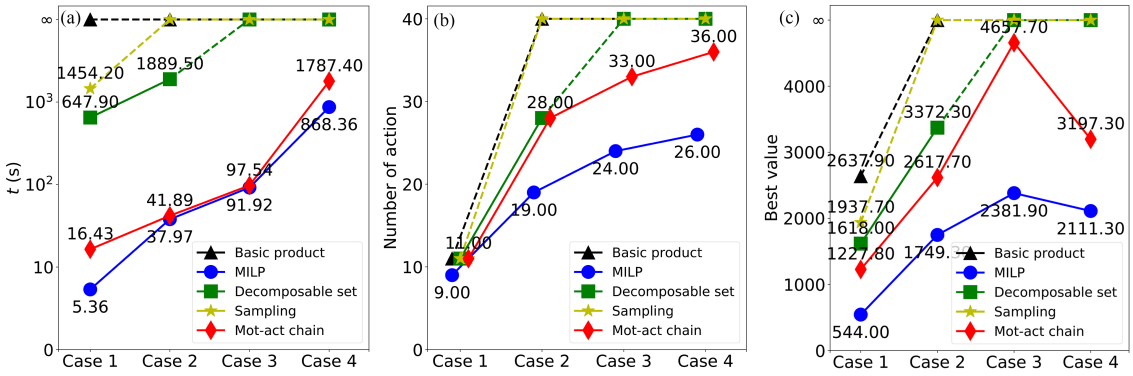

图 11　(网络版彩图) 不同基线算法的规划时长 (a), 动作数 (b) 和最优值 (c) 对比

Figure 11　(Color online) Comparison of the planning time (a), the number of actions (b) and the optimal values (c) among baseline methods

些较小的乘积自动机.

在图 11(a) 中记录了不同算法在各个案例中的规划时长. 其中, 由于基础乘法的复杂度随着智能体数量指数增加, 在最简单的案例 1 中, 生成了一个状态数超过 10^4 的乘积自动机, 因此所有的案例都不可行. 为便于比较, 在其余图中提供了案例 1 中单个智能体的计算结果. 在所有方法中, 混合整数规划和我们的动作链计算方法具有数量级上的优势, 能处理复杂的案例 3 和 4. 另外, 在图 11(b) 中记录了各算法由 LTL 公式计算得到的动作数量. 其中, 图搜索类的可分解集, 采样法和我们的动作 - 路径链方法都能结合智能体模型推理出潜在的需要执行的动作, 但可分解集和采样法均无法处理更为复杂的案例. 而混合整数规划方法只能执行 LTL 公式中提及的动作, 缺乏推理能力, 最终执行的动作数量均少于其他算法. 最后, 在图 11(c) 中记录了各算法的最优值. 其中, 混合整数规划能获得更低的最优值, 但是由于其忽略了智能体动作模型而执行了更少的动作, 规划方案是不充分的. 其次, 我们的动作 - 路径链算法则充分发挥了智能体间的协同作用, 计算得到第二低的最优值. 剩余方法中, 可分解集方法能处理较为复杂的案例二, 其余由于复杂度的原因都无法得到可行解. 综上, 所提出的基于动作 - 路径链的方法既保留了图搜索类方法对智能体模型的推理能力, 又能保持和混合整数规划相近的计算速度.

6 总结

本文针对复杂智能体模型, 环境资源和动态环境的协同任务规划问题, 提出了一种基于动作链的高效的任务规划方法. 通过计算搜索智能体模型中与任务相关的动作链, 保留了图搜索类方法对潜在任务的推理能力. 并设计了基于动作链的离线分配方法, 达到了组合优化类算法同等规划速度. 最后设计了在线适应方法, 实现了响应速度和优化目标间的权衡. 通过仿真对比, 本文所提的算法计算效率优于大多数图搜索方法, 接近混合整数规划, 且保留了潜在动作的推理能力.

在未来的研究中, 计划加入以一定概率模型动态出现的任务, 将马尔可夫决策过程和鲁棒优化加入到算法当中.

参考文献

1 Tamio A, Pagello E, Parker L, et al. Advances in multi-robot systems. IEEE Trans Robot Autom, 2002, 18: 655–661
2 Alejandro T, Eva O, Antonín K, et al. Cooperative multi-agent planning: A survey. ACM Comput Surv, 2017, 50: 1–32
3 Zhe C, Javier A, Xiaoshan B, et al. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. IEEE Robot Autom Lett, 2021, 6: 5816–5823
4 Hou X L, Yang J N. A two-level scheme for multi-objective multi-debris active removal mission planning in low earth orbits. Sci China Inf Sci, 2022, 65: 152–201
5 Wu G H, Wang T Y. Large-scale constellation tt&c resource scheduling algorithm based on adaptive simulated model cling. Acta Aeron Astron Sinica, 2023, 44: 265–286 [伍国华, 王天宇. 基于自适应模拟退火算法大规模星座测控资源调度方法. 航空学报, 2023, 44: 265–286]
6 Wells A M, Dantam N T, Shrivastava A, et al. Learning feasibility for task and motion planning in tabletop environments. IEEE Robot Autom Lett, 2019, 4: 1255–1262
7 Omidshafiei S, Pazis J, Amato C, et al. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In: Proceedings of International Conference on Machine Learning (ICML), PMLR, 2017. 2681–2690
8 Chen M, Zhou T L. Data fusion using bayesian theory and reinforcement learning method. Sci China Inf Sci, 2020, 63: 170209
9 Keshanchi B, Souri A, Navimipour N J. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. J Syst Software, 2017, 124: 1–21
10 Yan M, Yuan H M, Xu J, et al. Task allocation and route planning of multiple uavs in a marine environment based on an improved particle swarm optimization algorithm. Eurasip J Adv Sig Process, 2021, 2021: 1–23
11 Liu Z S, Guo M, Li Z K. Time minimization and online synchronization for multi-agent systems under collaborative temporal logic tasks. Automatica, 2024, 159: 111377
12 Dimarogonas D V. Multi-agent plan reconfiguration under local LTL specifications. Int J Robot Res, 2015, 34: 218–235
13 Baier C, Katoen J P. Principles of Model Checking. Cambridge, MA: MIT Press, 2008
14 Yu X Y, Yin X, Li S Y, et al. Security-preserving multi-agent coordination for complex temporal logic tasks. Control Eng Pract, 2022, 123: 105–130
15 Tian D Y, Fang H, Yang Q K, et al. Two-phase motion planning under signal temporal logic specifications in partially unknown environments. IEEE Trans Ind Electron, 2022, 70: 7113–7121
16 Tumova J, Dimarogonas D V. Decomposition of multi-agent planning under distributed motion and task LTL specifications. In: Proceedings of IEEE 54th Annual Conference on Decision and Control, New York, 2015. 7448–7453
17 Li L, Chen Z Y, Wang H. Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints. IEEE Robot Autom Let, 2023, 8: 4991–4998
18 Mordechai B A. A primer on model checking. ACM Inroads, 2010, 1: 40–47
19 Gastin P, Oddoux D. Fast LTL to Büchi automata translation. In: Proceedings of International Conference on Computer Aided Verification (CAV), Paris, 2001. 53–65
20 Ding X, Smith S L, Belta C, et al. Optimal control of markov decision processes with linear temporal logic constraints.

IEEE Trans Autom Control, 2014, 59: 1244–1257
21 Kloetzer M, Mahulea C. Accomplish multi-robot tasks via petri net models. In: Proceedings of International Conference on Automation Science and Engineering (CASE), Gothenburg, 2015. 304–309
22 Leahy K, Serlin Z, Vasile C L, et al. Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS). IEEE Trans Robot, 2022, 38: 2516–2535
23 Kantaros Y, Michael M, Zavlanos M M. Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. Int J Rob Res, 2020, 39: 812–836
24 Faruq F, Laccrda B, Hawes N. Simultaneous task allocation and planning under uncertainty. In: Proceedings of International Conference on Intelligent Robots and Systems (IROS), Madrid, 2018. 3559–3564
25 Li H, Han C Q, Liu Y R. Mission planning for small satellite constellations based on improved genetic algorithm. Chin J Space Sci, 2019, 39: 129–134
26 Peng Z H, Cai J Q. A multi-mode multi-skill project scheduling reformulation for reconnaissance mission planning. Sci China Inf Sci, 2022, 65: 169201
27 Liu Z S, Guo M, Bao W M, et al. Fast and adaptive multi-agent planning under collaborative temporal logic tasks via poset products. Research, 2024, 7: 3–37
28 Van Laarhoven P J M, Aarts E H L. Simulated Annealing: Theory and Applications. Springer Netherlands: D. Reidel Publishing Company, 1987
29 Luo X S, Zavlanos M M. Temporal logic task allocation in heterogeneous multi-robot systems. IEEE Trans Robot, 2022, 38: 3602–3621
30 Makhorin A. GLPK (GNU linear programming kit). http://www.gnu.org/s/glpk/glpk.html, 2008
31 Schillinger P, Bürger M, Dimarogonas D V. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. Int J Robot Res, 2018, 37: 818–838

Collaborative planning of formal tasks based on action chains

Zesen LIU, Zhongkui LI & Meng GUO*

College of engineering, Peking University, Peking 100871*, China*
* Corresponding author. E-mail: meng.guo@pku.edu.cn

Abstract Multiagent task planning based on formal methods has attracted considerable attention due to its ability to handle diverse task specifications and complex systems. However, as the number of agents increases, the planning complexity grows exponentially. Existing search-based methods can only handle medium size fleets while integer program-based algorithms cannot incorporate collaborative actions. To address these issues, this paper proposes a novel planning scheme based on action chains, where tasks are assigned via partial order optimization. Furthermore, an online adaptation and synchronization algorithm is proposed to handle contingent tasks that are generated online and uncertainty during task execution. Extensive numerical simulations are conducted to validate the effectiveness and reliability.

Keywords multiagent task planning, formal method, linear temporal logic, online self-adaptation, partially ordered set

附录 A 附录

表 A1 智能体, 动作, 资源和感兴趣区域的定义

Table A1 The definition of agent, action, resource and interested region

Agent	**Capacity**			**Label**
Volunteer	Get Brine, Supply Brine, Add Salt, Get Water, Supply food, Get food, Officiate Match, Disperse Crowd			Vo
Referee	Officiate Match, Statistical Achievement, Disperse Crowd			Ju
Sport Doctor	Get Illness, Put Illness, Diagnose, Get Brine, Supply Brine			Do
Resource	Water	Food	Brine	Illness
Action	**Requirement**	**Action**	**Requirement**	
Get Brine	Brine: 1, Time: 70 s	Get Food	Food: 1, Time: 30 s	
Supply Brine	Time: 30 s	Officiate Match	Time: 150 s	
Add Salt	Time: 10 s	Disperse Crowd	Time: 120 s	
Get Food	Time: 30 s	Diagnose	Time: 120 s	
Get Water	Water: 1, Time: 30 s	Get Illness	Illness: 1, Time: 100 s	
Put Illness	Time: 150 s	Statistical Achievement	Time: 300 s	
Supply Food	Time: 30 s	-	-	

表 A2 案例中的 sc-LTL 任务公式

Table A2 The sc-LTL formulas in the cases

Formula	**Description**
φ_1	$\diamond sw_{r1} \wedge \diamond sw_{r2} \wedge \diamond(dc_{t3} \wedge \diamond(om_{t3} \wedge \diamond sa_{t3}))$: Supply water(sw) at region $r1, r2$; disperse the crowd(dc) at $t3$ and then officiate the match(om), and finally statistical achievement(sa).
φ_2	$\diamond(dc_{t4} \wedge \neg om_{t4} \wedge \diamond om_{t2}) \wedge \diamond dc_{t5} \wedge \diamond sf_{r3}$: Disperse crowd(dc) at region t_4. After that, officiate match(om) at region t_4; disperse crowd(dc) at region t_5, and supply food(sf) at region r_3.
φ_3	$\diamond(dc_{t3} \wedge \diamond sb_{r1} \wedge \diamond sb_{r2} \wedge \diamond sb_{r3}) \wedge \diamond(om_{t3} \wedge \diamond sa_{t9})$: Disperse crowd(dc) at region t_3, then supply the brine(sb) at region r_1, r_2, r_3; officiate match(om) at region t_3, then statistical achievement(sa) at region t_9.
φ_4	$\diamond(dc_{f4} \wedge \neg Do_{f4} \wedge (\diamond om_{f7} \wedge \neg Do_{f7}))$: Disperse crowd(dc) at region f_4, then officiate match(om) at region f_7, the sport doctor(Do) should not enter related regions.
φ_5	$\diamond sa_{f1} \wedge \diamond sb_{f1}$: Statistical achievement(sa) at regin f_1 and supply brine(sb) at region f_1.
φ_6	$\diamond(di_i \wedge \neg gi_i \wedge \diamond gi_i) \wedge \neg Vo_i \wedge \neg \mathrm{Re}_i \bigcup gi_i$: Diagnose the illness(di) at region i, then put the illness(ti) to region i; the referee(Re) and volunteer(Vo) should not enter region i until the illness is removed.
φ_7	$\diamond(dc_i \wedge \neg om_i \wedge \diamond(om_i \wedge \diamond sa_j))$: Statistical Achievement (sa) at regin j.
φ_8	$\diamond sf_i$:Supply food(sf) at region i.
φ_9	$\diamond sw_i$: Supply water(sw) at region i.