利用可持续信息高效解决在线多智能体路径规划问题的方法

[1, 3]Mingkai TANG, [1, 6] Boyi LIU, [1, 4]Yuanhang LI, [1, 2, 5]Hongji LIU, [1, 2, 7]Ming LIU, [1, 8]Lujia WANG

[1]香港科技大学 [2]香港科技大学（广州）{[3]mtangag, [4]yliog, [5]hliucq}
@connect.ust.hk, [6]by.liu@ieee.org, {[7]eelium, [8]eewanglj}@ust.hk

arXiv:2301.04446v1 [cs.MA] 11 Jan 2023

摘要

多智能体路径规划（MAPF）旨在让智能体移动至目标顶点，同时避免碰撞。在线 MAPF 问题中，任何时候都可能有新的智能体加入环境，而当前的智能体并不了解未来的智能体。现有在线方法无法复用先前规划上下文，因此会产生冗余计算并降低算法效率。为此，我们提出一种利用可持续信息解决在线 MAPF 的三级方法，以减少冗余计算。高层求解器可持续重规划算法（SR）负责管理规划上下文并模拟环境。中层求解器可持续基于冲突的搜索算法（SCBS）构建冲突树并维护规划上下文。低层求解器可持续反向安全区间路径规划算法（SRSIPP）是一种高效的单智能体求解器，它利用先前的规划上下文减少重复计算。实验表明，我们提出的方法在计算效率方面有显著提升。在其中一个测试场景中，在不同智能体数量设置下，我们的算法平均速度可达到 SOTA 的 1.48 倍。

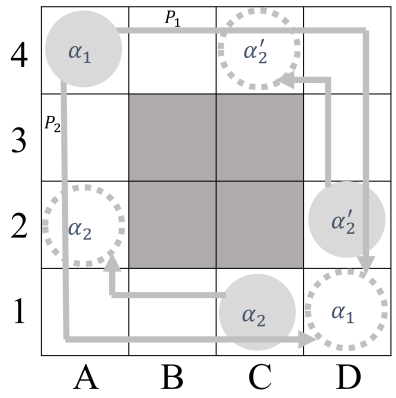


图 1：两个在线多智能体路径寻找实例示例。在这两个实例中，a_1 于时间点 0 出现，并需要前往 $D1$。P_1 和 P_2 是针对a_1代价相同的两条路径。在第一个实例中，a_2 于时间点 1 出现在 $C1$，其目标为 $A2$。在第二个实例中，a_2' 于时间点1出现在 $D2$，其目标为 $C4$。

1 引言

多智能体路径寻找问题（MAPF）旨在为一组智能体寻找路径，使它们从起始顶点移动到目标顶点，且彼此之间不会发生碰撞。MAPF 在许多实际场景中都有广泛应用，例如飞机牵引车（Morris 等人，2016）、仓库机器人（Wurman、D'Andrea 和 Mountz，2008）、电子游戏（Ma 等人，2017b）以及城市道路网络（Choudhury 等人，2022）。

对于多智能体路径寻找，大多数研究假设在系统运行之前就能完整掌握环境（Salzman 和 Stern，2020；Stern，2019；Ma 等人，2017a）。在这一假设下，可以预先计算出解，智能体在运行时只需按照预先计算好的计划采取行动。这类路径寻找问题称为离线多智能体路径规划。然而，在实际应用中，这一假设并不总能成立。系统运行期间，新的智能体可能会突然加入系统，因此智能体需要重新规划，以适应新的情况。近期，有人提出了在线多智能体路径寻找问题（Svancara 等人（2019））。该问题假设新智能体可能随时加入环境，而当前智能体并不了解未来的智能体。环境中的所有智能体都需要重新规划，以适应新的情况。

在线多智能体路径查找是一个具有重要实际意义的问题。例如，在现实仓库系统中，机器人常常会因充电或故障等因素进出工作区域。此外，机器人重新进入工作区域的时间点无法预测。由于新智能体的出现，仓库中的智能体需要调整其路径。

最优地解决离线多智能体路径寻找问题是 NP难的（Yu 和 LaValle 2013；Ma 等人 2016）。与可在系统运行前计算路径的离线多智能体路径规划相比，在线多智能体路径查找还需要在新智能体出现时实时为所有智能体计算高质量路径。Svancara 等人提出了几种解决在线多智能体路径查找问题的方法。单智能体重规划算法会在每个新智能体出现时为其搜索一条最优路径，同时保持旧智能体的所有路径不变。单智能体重规划组算法会为同时出现的所有新智能体联合规划，且新智能体的路径规划不会影响旧智能体的路径规划。这两种算

法执行速度快，但所得解并非最优。Replan All算法会在新智能体出现时，在不考虑现有路径规划的情况下为所有智能体重新规划，因此可以获得高质量的解。然而，当智能体数量增多时，Replan All算法每次迭代规划的计算复杂度都很高。

考虑到重用先前规划迭代中的信息可以减少运行时间，我们提出了一种高效算法。在本文中，我们将这类信息称为可持续信息或规划上下文。我们的方法由三级算法组成。我们将高层算法命名为可持续重规划算法（SR）。它模拟环境并维护完整的规划上下文。中层算法可持续冲突搜索算法（SCBS）由 SR 调用，用于基于当前信息搜索多智能体路径规划解。SCBS 使用可持续反向安全区间路径规划算法（SRSIPP）作为低层求解器，进行单智能体规划。鉴于单个智能体在每次规划迭代中具有相同的目标点，但起始点不同，SRSIPP 采用反向方向搜索路径（从目标点到起始点），以重用先前的规划信息。

本文的主要贡献如下。
1. 我们提出了一种三级方法，用于重用先前的规划上下文，以减少在线多智能体路径查找的运行时间。2. 我们证明了所提方法的完备性和快照最优性。3. 我们针对 SOTA 进行了详细的算法性能比较实验。相对于 SOTA，平均加速比最高可达 1.48。

2 问题定义

在线多智能体路径查找问题的定义是：给定一个有向图 $G(V, E)$ 和一组 k智能体 a_1、 a_2、 a_3 ..... a_k，为每个智能体寻找一条无碰撞路径。智能体 a_i由三元组 $(t_i^s,\ v_i^s,\ v_i^g)$ 描述，表示智能体 a_i 在时间点 t_i^s 出现在起始顶点 $v_i^s \in V$，其目标是顶点 $v_i^g \in V$。在本文中，我们称智能体 i 从 t_i^s开始。我们不失一般性地假设 $0 \le t_1^s \le t_2^s \le \ldots \le t_k^s$。特别地，起始时间点为 0 的智能体可视为在环境开始运行前就已处于场景中的智能体，我们将针对这些智能体的规划称为在线多智能体路径查找问题的离线 部分。相反，我们将系统启动后的规划称为在线 部分。一开始，所有智能体都会规划从各自起始顶点到目标顶点的路径，但它们并不知道未来将启动的智能体的任何信息。之后，智能体在每个时间点按照各自的计划行动。当到达新智能体启动的时间点时，所有智能体都需要根据在线多智能体路径查找问题的新输入重新规划路径。令 m 表示新智能体启动的时间点数量， t_1^{new}、 t_2^{new}.....t_m^{new} 表示相应的时间点序列。 m 可能小于 k，因为可能有多个智能体在同一时间点启动。在线多智能体路径查找问题的解定义为一系列有效规划 $\Pi = \langle \pi^0,\ \pi^1,\ \pi^2 \ldots \pi^m \rangle$，其中 π^j 是 t_j^{new}时所有路径规划的集合。令 p_i^j 表示智能体 i 在 π^j中的路径规划，令 $p_i^j[t]$ 表示智能体 i 在 π^j的时间点t 所处的顶点。我们定义 $p_i^j[t_l :\ t_r]$ 为智能体 i 从时间点 t_l 到时间点 t_r的路径规划串接，即$p_i^j[u :\ v] = p_i^j[u] \circ p_i^j[u+1] \circ \ldots \circ p_i^j[v-1]$，其中 $\circ$ 是串接运算符。智能体 i的执行规划定义为 $Ex_i[\Pi] = p_i^1[t_1^{new} : t_2^{new}] \circ p_i^2[t_2^{new} :\ t_3^{new}] \circ \ldots \circ p_j^{m-1}[t_{n-1}^{new} :\ t_n^{new}] \circ p_j^n[t_n^{new} :\ \infty]$，表示智能体 i的实际路径。

本文关注以下变体：

- 我们假设智能体从车库出发，这意味着新智能体可以选择在起始时间或之后进入起始顶点。在进入之前，它们会在车库中等待，且不会与其他智能体发生冲突。此外，我们采用在目标顶点消失的设定。在这两个假设下，只要离线部分可解，且每个智能体都有一条从其初始位置到目标位置的路径，问题就始终可解，正如 (Svancara 等人（2019）) 中的命题 2 所证明的那样。虽然我们采用了这两个假设，但所提出的方法也很容易扩展到其他起点和目标设定。

- 我们只考虑顶点冲突和边冲突。当两个智能体占据同一个顶点，或在同一时间沿同一条边反向穿行时，它们就会发生碰撞。

离线多智能体路径规划问题通常采用两个目标：最小化最大完成时间和最小化代价总和（*SOC*）。最大完成时间是所有智能体中最大的完成时间。然而，对于在线多智能体路径查找问题，最小化最大完成时间并不恰当，因为新智能体会不断加入环境，后加入的智能体更有可能影响这一目标。SOC 是所有智能体路径规划代价的总和。然而，不能直接根据 SOC 比较两个在线多智能体路径查找求解器，因为 SOC 无法衡量两个求解器的实际质量。例如，使用两个求解器 s_1 和 s_2 来求解图 1 中的实例。在时间点 0， a_1 从 $A4$ 出发。它有两条通往其目标 $D1$、代价相同的路径。假设 s_1 选择 P_1，而 s_2 选择 P_2。现在考虑 s_1， a_1 在时间点 1 前往 $B4$。在第一个实例中， a_2 出现， a_1 的路径不会受到影响，并会继续沿着长度为 6 的路径 $[B4, C4, D4, D3, D2, D1]$ 前行。然而，在另一个实例中， a_2' 出现， a_1 的路径将绕行 $[B4, A4, A3, A2, A1, B1, C1, D1]$，路径长度为 8。 s_2 上也会出现对称的情况。我们无法判断 s_1 是否优于 s_2，因为实际代价取决于未来的智能体，而这在较早的时间点无法预测。直接使用 SOC 无法评判求解器的质量。我们将求解器定义为快照最优求解器，前提是该求解器在假设未来不会出现新智能体的情况下，能够获得 SOC 意义上的最优路径。快照最优求解器的解质量优于非最优求解器。

3 方法论

我们的方法是一种三级方法。图2展示了该方法的架构。高层求解器是可持续重规划算法（SR），它模拟环境并维护所有智能体的规划上下文。

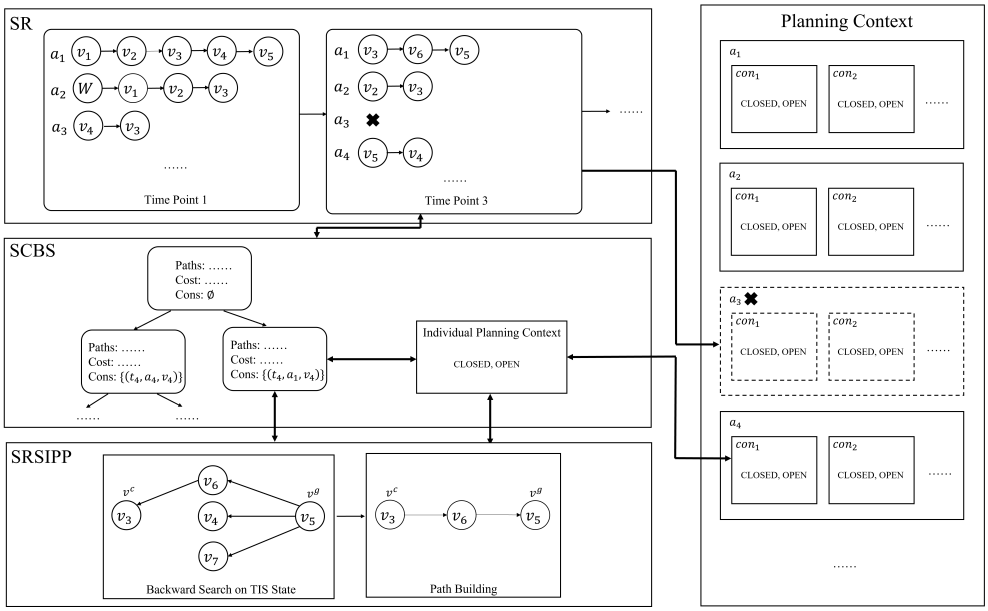


图2：三级方法的架构。SR算法是高层求解器，用于模拟环境并管理规划上下文。SCBS算法是中层求解器，用于构建冲突树并提取各智能体的规划上下文。低层求解器 SRSIPP 算法在 TIS状态上使用反向搜索进行单智能体路径规划。

中层求解器可持续基于冲突搜索算法（SCBS）为多个智能体规划最优路径，并管理规划上下文。低层求解器可持续反向安全间隔路径规划（SR-SIPP）则在一组约束下解决单智能体问题。

3.1 可持续重规划算法

SR算法是最高层的求解器。它可以模拟场景，并持续维护规划上下文。当一个或多个智能体出现时，该算法会为所有智能体重新规划。图2展示了SR算法的一个示例。图中的“W”表示在车库中等待的动作。

我们将 pc 定义为规划上下文，它是一个两级哈希表，用于保存所有规划上下文。其键为智能体的ID以及施加于该智能体的所有约束，值则由其低层求解器确定。我们将在后续小节中介绍具体的规划上下文。

伪代码如算法 1 所示。设 t^c 为当前时间点， v^c 为智能体所在的当前顶点， A 为已开始运行的智能体集合， A^+ 为在时间点 t^c出现的新智能体集合， Ex为执行计划。在第 1 至 8 行中，将环境模拟至时间点 t^c。如果智能体 a 在时间点 t^c之前到达其目标，则移除 A 和 pc 中的所有相关元素。否则，根据先前的执行计划 Ex获取智能体 a 的当前顶点 v^c。在第 10 行中，SCBS 算法计算最优路径规划 p。在第 11 行中，通过 p更新执行计划：删除从时间点 t^c开始的路径，并将新的路径计划接续到执行计划中。

算法 1：可持续重新规划

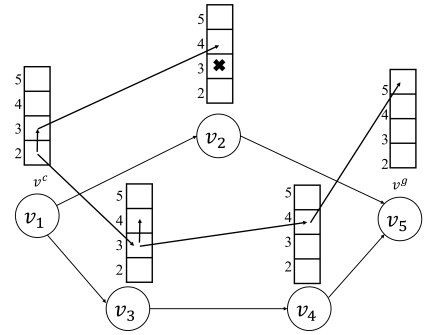
输入：原始智能体集合 A，新智能体集合 A^+，执行计划 Ex，当前时间点 t^c，规划上下文 pc

1: **对于** 智能体 a in A **执行**
2: **如果** a 在时间之前到达目标 t^c **则**
3: $A \leftarrow A \setminus \{a\}$
4: 移除 pc 中 a 的所有规划上下文。
5: **else**
6: 通过 Ex 更新 $a.v^c$ 。
7: **结束 if**
8: **结束 for**
9: $A \leftarrow A \cup A^+$。
10: p， $pc \leftarrow SCBS(A,\ t^c,\ pc)$ // 算法 2
11: 通过 p 更新 Ex。
12: **返回** A， Ex， pc

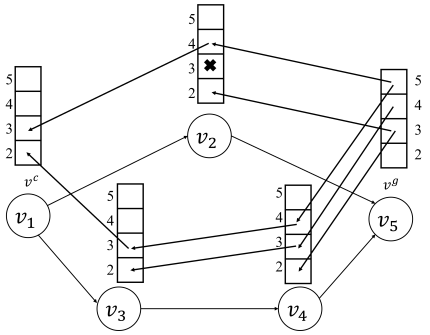
3.2 可持续冲突搜索算法

可持续基于冲突搜索算法（SCBS）计算多智能体路径规划，并以可持续的方式维护规划上下文。它是在基于冲突的搜索算法（CBS）（Sharon 等人，2015）的基础上扩展而来的。SCBS 与 CBS 的主要区别在于对规划上下文的处理。

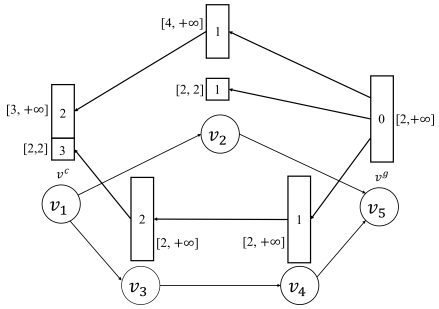
SCBS 算法的一个使用示例如图 2 所示。每次进行低层搜索之前，都会根据智能体及其相关约束，提取与该低层搜索直接相关的规划上下文。我们将这部分规划上下文称为个体规划上下文，并用 ipc 表示。在低层搜索过程中，



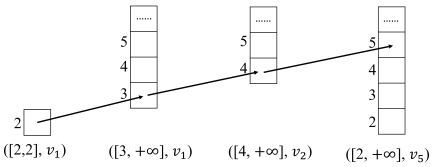
(a) 对 TS 状态进行正向搜索。



(b) 对 TS 状态进行反向搜索。



(c) 对 TIS 状态进行反向搜索。方框中的数字是到 v^g的代价。



(d) 基于 TIS 状态进行路径构建。

图 3：(a)–(c) 是在相同图和约束下的三种不同搜索方法。(d) 基于 (c)。方框旁的数字表示时间点或时间区间。

ipc 会被修改以适应新的实例。低层搜索完成后，新的 ipc 将被放回 pc，以便在后续迭代中使用。

算法 2 展示了伪代码。在第 5–7 行和第 25–27 行，函数 $GetIPC$ 会从 pc 中筛选出 ipc。使用低层求解器后，新的 ipc 会被放回 pc。

3.3 可持续反向安全区间路径规划算法

现在介绍可持续反向安全区间路径规划算法（SRSIPP）。SRSIPP 是一种基于 A*（Hart、Nilsson 和 Raphael, 1968）和 SIPP（Phillips 和 Likhachev，2011）的单智能体求解器，旨在复用先前的个体规划上下文，以降低搜索的复杂度。在讨论 SRSIPP 算法时，所有符号中都省略智能体索引 i 和当前规划迭代次数 j，例如 $t_{i,\ j}^s$、t_i^g，$_j$、v_i^s、v_i^g 到 t^s、t^g。令 v^c 表示智能体当前所在的顶点，$s^c = (t^c,\ v^c)$ 表示智能体的当前状态。

在在线多智能体路径查找中，智能体可能会在执行计划的过程中重新规划。尽管每次规划的起始顶点不同，但终点保持不变。SRSIPP 利用这一特性，实现复用先前规划上下文的目标。对于某些单智能体求解器，规划会沿着边将智能体从当前状态引导至目标。这些搜索方法称为正向搜索。规划也可以沿着反向边，从目标搜索至智能体的当前状态。这些搜索方法称为反向搜索。SRSIPP 是一种反向搜索算法。

在多智能体路径寻找问题中，大多数单智能体搜索方法都是在时空（TS）状态上进行正向搜索。图 3(a) 给出了一个示例。然而，由于整棵搜索树以起始顶点为根节点，而起始顶点会随每次搜索而改变，因此正向搜索无法复用先前的规划信息。注意到同一智能体的目标顶点不变，我们可以将目标顶点设为搜索树的根节点，以便在后续搜索中复用该树。然而，在搜索开始之前，我们无法预测到达时间。为保证算法的最优性，必须先对所有较早到达目标的状态进行完整搜索，再搜索较晚到达的状态，这会导致大量额外计算。图 3(b) 给出了一个示例。

为加快计算，我们提出在时间区间空间（TIS）状态上进行搜索。图 3(c) 给出了一个示例。设 $([t_l, t_r],\ v)$ 为 TIS 状态，其中 $[t_l, t_r]$ 是一个时间区间，v 是顶点；设 $(t,\ v)$ 为 TS 状态，其中 t 是一个时间点。TIS 状态 $([t_l, t_r],\ v)$ 包含一组 TS 状态 $\{(t,\ v)|t \in [t_l, t_r]\}$。设 $g_{TS}(s)$ 和 $g_{TIS}(s)$ 分别为从 TS 状态和 TIS 状态到 v^g 的代价。同一 TIS 状态中的所有 TS 状态都可以使用相同的顶点序列作为通往目标的最短路径。形式化地，我们有

$$\forall t \in [t_l, t_r], g_{TIS}(([t_l, t_r], v)) = g_{TS}((t, v)). \quad (1)$$

我们将 $g_{TS}(s)$ 和 $g_{TIS}(s)$ 的函数值分别称为 TS 状态和 TIS状态的g 值。当且仅当 TIS状态不覆盖任何受约束的

算法 2：SCBS

输入：智能体 A，当前时间点 t^c，规划上下文 pc

```
OPEN ← ∅
R ←新节点
R.cons ← ∅
对每个智能体 a_i 执行
    ipc ← GetIPC(pc,a_i,R.cons[a_i])
    R.path[a_i], ipc ← SRSIPP(a_i, NULL, t^c, ipc) // 算法3
    通过 ipc更新 pc。
结束循环
R.cost ← 计算 P.paths的 SOC
OPEN ← OPEN ∪ {R}
当 OPEN ≠ ∅ 时
    N ← OPEN 中代价最低的节点。
    OPEN ← OPEN\{N}
    L ← N 中最早发生的碰撞
    如果 L 是 None，则
        返回 N.paths, pc
    结束条件判断
    C ←从 L 获取约束
    对于 约束 c，在 C 执行
        P ←新节点
        P.cons ← N.cons
        P.paths ← N.paths
        a ← c.agent
        将 c 插入 P.cons[a]。
        ipc ← GetIPC(pc, a, P.cons[a])
        P.paths[a], ipc ← SRSIPP(a, P.cons[a], t^c, ipc) // 算法3
        通过 ipc更新 pc 。
        如果 P.path[a] 不是 NULL，则
            P.cost ← 计算 P.paths的 SOC
            OPEN ← OPEN ∪ {P}
        结束 if
    结束 for
结束 while
```

TS状态，或覆盖时间点小于 t^s 的 TS状态时，该 TIS状态才有效。最大 TIS状态是有效的 TIS状态，其时间区间不是同一顶点上其他有效 TIS状态的子集。在搜索顶点之前，将创建该顶点上的所有最大 TIS状态。它们的 g 值被设为无穷大，但 v^g 上的 TIS状态的 g 值被设为 0。

反向搜索中扩展一个状态的过程如下所述。搜索的操作包括沿反向边移动到相邻顶点，以及停留在同一顶点。我们用 $([t_l, t_r], v)$ 表示需要扩展的 TIS 状态。若 s'中的所有 TS 状态都能沿前向方向通过一个动作转移到 s 中的某个 TS 状态，则定义 TIS 状态 s'为另一个 TIS s 的虚拟子节点。虚拟子节点的代价比原状态多一，即 $g_{TIS}(s') = g_{TIS}(s) + 1$。令 $N(v) = \{v' | ((v', v) \in E) \vee (v' = v)\}$ 表示 v的邻域，并令 $ds(([t_l, t_r], v)) = \{([t'_l, t'_r], v') | v' \in N(v)\}$表示 $([t_l, t_r], v)$ 的虚拟子节点集合。对于虚拟子节点状态 $([t'_l, t'_r], v') \in ds(([t_l, t_r], v))$ 。我们设定

$$\begin{aligned} t'_l &= \max(t^s, t_l - 1) \\ t'_r &= t_r - 1 \end{aligned} \tag{2}$$

虚拟子节点用于更新同一顶点上的所有当前 TIS 状态。如果整个 TIS 状态都可以改进，则可以直接修改该 TIS 状态的代价。假设由于时间区间覆盖的限制，只能改进 TIS 状态的一部分。在这种情况下，TIS 状态将根据时间区间覆盖拆分为多个 TIS 状态，且只有被虚拟子节点的时间区间完全覆盖的状态才会更新。图4展示了一个示例。

我们使用 A* 进行反向搜索。令 $h_{TIS}(s)$ 为从 TIS 状态 s到 s^c的代价估计启发式函数，令 $h_v(v)$ 为从顶点 v 到 v^c 的路径长度估计启发式函数。我们将 $h_{TIS}(s)$ 定义为

$$h_{TIS}(([t_l, t_r], v)) = \max(\max(t_l - t^c, 0), h_v(v)) \tag{3}$$

其中 $t_l \geq t^s$。对于满足 $t_r < t^s$ 的状态，将不进行搜索，且这些状态的启发式函数未定义。在四邻域网格中，$h_v(v)$ 通常定义为到目标点的曼哈顿距离，即

$$h_v(v) = |v_x - v_x^g| + |v_y - v_y^g| \tag{4}$$

其中，(v_x, v_y) 是 v的协调，(v_x^g, v_y^g) 是 v^g的协调。TIS状态的评价函数定义如下。

$$f_{TIS}(s) = g_{TIS}(s) + h_{TIS}(s) \tag{5}$$

令 $h_{TIS}(s)$ 和 $f_{TIS}(s)$ 的函数值分别为 TIS状态 s的 h值和 f 值。

在 SRSIPP 中，个体规划上下文包括开放列表和关闭列表。我们用 $OPEN$ 和$CLOSED$ 表示它们。在新一轮搜索开始时，我们会调整个体规划上下文以适应新的规划。具体而言，根据新的当前状态和公式 (3, 5)，重新计算 $OPEN$中所有状态的 h 值和 f值。$CLOSED$ 可以直接使用，无需修改。调整后，新一轮搜索可以复用上一轮搜索的 $OPEN$和 $CLOSED$。

伪代码如算法3所示。在代码中，$a.v^c$ 和 $a.v^g$ 分别是智能体 a的当前顶点和目标顶点。此外，$s.t_l$ 和 $s.t_r$ 是 s中时间区间的端点。状态s 的 g 值、h 值和 f 值分别保存在 $s.g$、$s.h$ 和 $s.f$中。如果状态是搜索的最优最终状态，则该状态为终止状态。我们使用 ts 保存终止状态，并使用 cts 保存所有候选终止状态。具体而言，cts 保存顶点 $a.v^c$中所有已关闭的状态；该顶点对智能体可达，即 $t^c \leq s.t_r$。第1行提取个体规划上下文。如果当前状态无效，则直接返回（第2–4行）。在第5行，我们更新$OPEN$中所有状态的 h 值和 f 值。在第6行，如果尚未创建 $a.v^g$ 上的TIS状态，则创建所有最大TIS状态并将其放入 $OPEN$。在第7–12行，使用函数 $StopCheck$ 检查搜索是否可以停止。如果可以，则构建路径

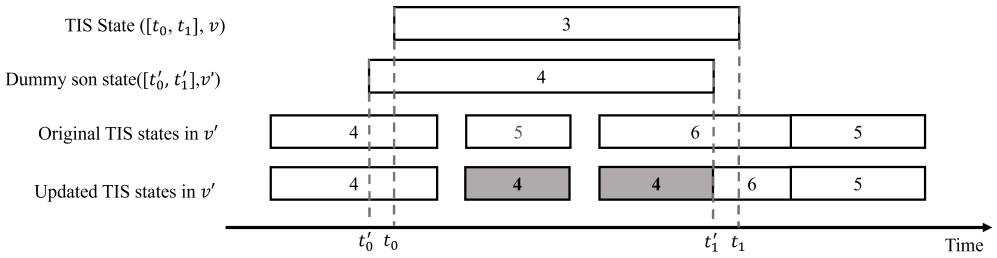


图4：展示更新TIS状态代价过程的示例。方块表示该状态的时间区间，方块中的数字表示到达 v^g的代价。第一行和第二行分别展示TIS状态（$[t_0, t_1]$， v）及其虚拟子状态（$[t'_0, t'_1]$， v）。第三行表示更新前 v 中的所有有效TIS状态。第四行展示更新后的TIS状态。第一个状态无法改进，而第二个状态可以整体改进。对于第三个状态， $[t'_0, t'_1]$只覆盖了部分时间区间。该状态被拆分为两个状态，且仅更新被覆盖状态的代价。第四个状态的时间区间未被覆盖，因此不受影响。灰色方块表示改进后的TIS状态。

使用函数 $BuildPath$ 构建路径并直接返回（稍后我们将讨论 $StopCheck$ 和 $BuildPath$ 的细节）。否则，搜索开始。每次迭代中，获取 $OPEN$ 中 f 值最小的状态（第14行）。如果该时间区间无法覆盖 t^c之后的任何时间点，则该状态对当前及后续搜索都没有用处。我们忽略该状态并进入while循环的下一次迭代（第15–17行）。在第18–27行，生成虚拟子节点以更新状态。在第29行，我们更新 $OPEN$、$CLOSED$ 和 cts。最后，我们检查搜索是否可以停止（第30–32行）。如果不能，则进入下一次迭代。

$StopCheck$ 算法会检查搜索是否可以停止，并找到最佳终止状态。当不可能从 cps 中找到更优的目标状态时，我们就停止搜索。停止搜索有两种不同的情形。如果智能体位于场景中，当 cps 中的一个状态覆盖 t^c时，搜索便停止。否则，智能体可以选择一个时间点进入场景。假设选定状态（$[t_l, t_r]$， v）作为终止状态，则最佳进入时间点为 $\max(t_l, t^c)$，从智能体当前的 TS 状态（t^c， v^c）到顶点 v^g 的总代价为 $\max(t_l - t_c, 0) + g_{TIS}([t_l, t_r], v)$。如果在 cps 中选择状态的最小总代价小于或等于当前扩展状态的 f值，则无法找到更优的解，搜索可以停止。

$StopCheck$ 算法的伪代码如算法4所示。在第1–3行中，如果 cts中没有元素，搜索就不能停止。在第4–10行中，如果智能体位于场景中，只有当 cts 中的一个状态覆盖 t^c时，搜索才能停止。在第12–18行中，如果智能体不在场景中，则查找总代价最小的状态。如果该代价不高于 $OPEN$中所有节点的最小 f值，搜索便可停止，并返回最佳终止状态。

$BuildPath$ 函数在算法3中构建最终的 TS 状态路径。到达终止状态后，我们通过回溯得到一条 TIS路径。基于此，我们构建 TS 路径作为解。特别地，如果智能体不在场景中，且当前时间点早于终止状态时间区间中的任意时间点，智能体将等待，直到到达目标 TIS状态的最早时间点，然后进入场景。图3(d)展示了路径构建的一个示例，其中选择了 $([3, \infty], v_1)$ 作为终止状态。

4 理论分析

定理1。 如果 h_v（v） 是可采纳的，并满足一致性假设，那么当 *StopCheck* 函数的第一个返回值为 *true* 时，cts之外不可能存在更优的终止状态。

证明。 证明见附录。 □

定理2。 如果 $h_v(v)$是可采纳的，并且满足一致性假设，那么*SRISPP*算法是完备且最优的。

证明。 如果 $h_v(v)$是可采纳的，并且满足一致性假设，那么 $h_{TIS}(s)$也可采纳，并满足一致性假设。证明见附录。

如果这是对 a 和$cons$配置的首次规划，那么 $OPEN$ 和 $CLOSED$ 最初为空。这是标准的A*，满足搜索的完备性和最优性。

如果搜索开始时 $OPEN$ 和 $CLOSED$ 非空，那么 $CLOSED$ 中的状态不会重新打开，而且该状态的 g 值已经是到目标顶点的最小代价。对于 $OPEN$中的状态，我们会更新其 h 值和 f 值，以适应新的情况。这种情形可以看作从标准A*中的一个快照开始，只是某些状态已预先关闭。然而，提前关闭的节点不会影响算法的完备性和最优性，因为它们所有尚未关闭的邻居都在 $OPEN$中。

结合定理1和上述讨论，定理得证。

□

推论1. 如果 $h_v(v)$ 是可采纳的并满足一致性假设，则 *SCBS* 是完备且最优的。

推论2. 如果 $h_v(v)$ 是可采纳的并满足一致性假设，则 *SR* 是完备且快照最优的。

这两个推论可根据 CBS 和 RA算法的性质证明（ Sharon 等人，2015；Svancara 等人（2019））。由于定理2已得到证明，且 SCBS 和 SR 中对规划上下文的操作不会影响完备性和最优性，因此这两个推论得证。

算法3：SRSIPP 输入：智能体 a，约束 $cons$，当前时间点 t^c，个体规划上下文 ipc1： $OPEN$，$CLOSED \leftarrow ipc$2：如果 a 位于场景中且 $(t^c, a.v^c) \in cons$，则 3：返回 $false, ipc$4：结束 if 5：更新 $OPEN$ 中状态的 h 值和 f 值。6：如果尚未创建，则通过 $cons$ 在 v' 中创建尽可能多的 TIS状态，并将其放入 $OPEN$。7： $OPEN$ 中最小的 f 值 8： $cps \leftarrow \{([t_l, t_r], v) \in CLOSED | a.v^c = v \wedge a.t^s \leq t_r)\}$ 9： $stop, ts \leftarrow StopCheck(cps, f_{min}, a)$ // 算法4 10：如果 $stop$，则 11：返回 $BuildPath(ts, a)$ $\{OPEN, CLOSED\}$12：结束 if 13：当 $OPEN$ 不为空时执行 14： $OPEN$ 中具有最小 f 值的 TIS状态 15：如果 $s.t_r < t^c$，则 16：继续 17：结束 if 18：对于 $N(s.v)$ 中的每个顶点 v' 19： v' 中 s 的虚拟子节点 20：如果尚未创建，则通过 $cons$ 在 v' 中创建尽可能多的 TIS状态，并将其放入 $OPEN$。21：对于 v' 中的每个 TIS状态 s'，执行 22：如果 s' 可通过 ds 改进，则 23：改进后得到的新状态 s' 24： $OPEN \leftarrow OPEN \backslash \{s'\} \cup S'_{new}$25：结束 if 26：结束 for 27：结束 for 28： $stop, ts \leftarrow StopCheck(cps, s.f, a)$ // 算法 4 29：通过 s 更新 $OPEN, CLOSED$ 和 cps。30：如果 $stop$，则 31：返回 $BuildPath(ts, a)$ $\{OPEN, CLOSED\}$32：结束 if 33：结束 while

5 实验

我们的实验旨在评估所提出方法的计算效率。我们使用两个四邻接网格地图数据集，其设置与 Svancara 等人（2019）类似，分别为小型网格地图数据集和大型网格地图数据集。我们在这些网格地图上执行在线多智能体路径查找。具体而言，我们将每个智能体从一个单元格移动到另一个单元格。在此期间，随时都可能有新的智能体开始移动。

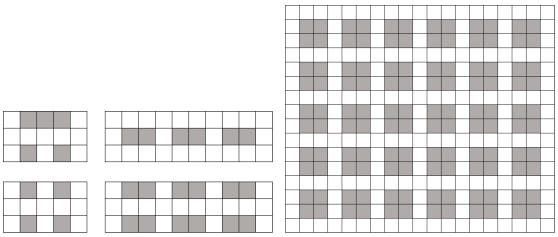
我们使用成功率和平均运行时间作为指标。运行时间限制为30秒。如果算法运行超过时限，则该实验实例判定为失败，并将时限作为其运行时间。我们按智能体数量进行统计。对于每个智能体数量，两个数据集中都各有100个实例。

实验假设场景初始时没有智能体。一方面，各算法的离线部分

算法4：StopCheck

输入：候选终止状态集合 cts，最小估计值函数在开放列表中的取值 f_{min}，智能体 a

1: 如果 cts 为空**则**
2: **返回** $false, NULL$
3: **结束 if**
4: **如果** a 在场景中**则**
5: s ← 覆盖 t^c的状态
6: **如果** s 是 $NULL$ **则**
7: **返回** $false, NULL$
8: **else**
9: **返回** $true, s$
10: **结束条件判断**
11: **否则**
12: $s_{min} \leftarrow argmin_{s \in cts}(\max(s.t_l - t^c, 0) + s.g)$
13: $c_{min} \leftarrow min_{s \in cts}(\max(s.t_l - t^c, 0) + s.g)$
14: **如果** $c_{min} \leq f_{min}$ **则**
15: **返回** $true, s_{min}$
16: **else**
17: **返回** $false, NULL$
18: **结束如果**
19: **结束如果**



（a）小型网格地图。 （b）大型网格地图。

˅ 图 5：实验所用的网格（引自 Svancara 等人（2019））。

相同。如果某些实例在离线部分失败，则不会执行在线算法。这些测试用例不适合用于比较。如果场景初始时没有智能体，所有随机生成的实例都始终可解。另一方面，所有参与比较的方法都使用相同的离线多智能体路径查找求解器。忽略这一点不会影响比较。

我们在配备 4.40 GHz AMD R7-5800X CPU 和 16GB 内存的计算机上运行这些算法。我们选取了四种基线方法进行比较：

- **RA+CBS+A*(A1)**：该算法使用 Replan All算法求解在线多智能体路径查找问题。新智能体开始行动时，它会使用 CBS 算法计算多智能体路径规划，其低层求解器为 A*。
- **RA+CBS+RSIPP(A2)**：该算法移除了我们提出的方法中的所有可持续操作，即所有求解器都不维护或使用规划上下文。
- **SR+SCBS+RSIPP(A3)**：对于该算法，低层求解器无法重用规划上下文。在中层求解器中，如果智能体严格遵循冲突树节点中的先前路径，则不会调用低层

k	A1	A2	A3	A4
10	**96%**	**96%**	**96%**	**96%**
12	**87%**	84%	86%	**87%**
15	78%	74%	79%	**80%**
17	**65%**	63%	63%	64%
20	**49%**	47%	47%	**49%**
22	36%	35%	37%	**37%**
25	27%	26%	28%	**29%**

表1：小型网格中的成功率表。 k 为智能体数量。

k	A1	A2	A3	A4
10	1.68(-)	2.2(0.77)	1.72(0.98)	**1.58(1.06)**
12	5.65(-)	5.96(0.95)	5.35(1.06)	**4.77(1.18)**
15	8.05(-)	8.67(0.93)	7.96(1.01)	**7.27(1.11)**
17	12.26(-)	12.61(0.97)	11.96(1.03)	**11.54(1.06)**
20	17.01(-)	17.18(0.99)	16.6(1.02)	**16.23(1.05)**
22	21.08(-)	22.02(0.96)	20.83(1.01)	**20.15(1.05)**
25	22.53(-)	22.39(1.01)	22.15(1.02)	**21.9(1.03)**

表 2：小型网格中的运行时间和加速比表格。单元格中的第一个数字表示运行时间（秒），括号中的数字表示相对于 A1 的加速比。 k 为智能体数量。

求解器，而是将先前规划结果中的后缀用作本次规划的结果。

- **SR+SCBS+SRSIPP(A4)**：所提出的方法。

5.1 小型网格地图

在第一个数据集中，使用了 4 个小型网格地图，如图5(a)所示。起点和目标点随机采样自网格两侧的两个单元格，起始时间点则从 $[1, 30]$ 均匀采样。设 k 为智能体数量，其范围为$\{10, 12, 15, 17, 20, 22, 25\}$。对于每个网格地图，我们针对每种智能体数量设置生成 25 个实验实例。对于每种智能体数量，都有 $25*4 = 100$ 个实例。

表1展示了成功率的结果。除智能体数量为 17 时外，A4 的成功率均大于或等于 A1。表2展示了所有基线方法的运行时间，以及其他基线方法相对于 A1 的加速比。结果显示，A2 未取得显著改进，而 A3 在大多数情况下能够加速。A4 的表现最佳。在所有智能体数量设置下，其平均运行时间均优于 A1，并且在所有基线方法中实现了最高加速比。

5.2 大型网格地图

在第二个数据集中，我们使用图5(b)所示的大型网格地图。对于范围为 $[90, 100]$ 的智能体数量 k 的每种设置，我们生成 100 个实验实例。起始时间点从 $[1, 100]$ 中随机采样。智能体的起点和目标点分别从两个不同的边采样。

表3和表4展示了成功率、平均运行时间，以及相对于各个基线方法的 A1 加速比结果。A2 和 A3 的表现优于 A1

k	A1	A2	A3	A4
90	84%	85%	88%	**91%**
92	82%	84%	86%	**89%**
94	90%	89%	91%	**92%**
96	75%	77%	81%	**84%**
98	72%	76%	82%	**83%**
100	54%	69%	72%	**77%**

表3：大型网格中的成功率表。 k 是智能体数量。

k	A1	A2	A3	A4
90	10.09(-)	8.98(1.12)	7.48(1.35)	**6.06(1.67)**
92	9.69(-)	9.61(1.01)	8.13(1.19)	**6.8(1.43)**
94	8.04(-)	8.27(0.97)	6.59(1.22)	**5.52(1.46)**
96	13.05(-)	13.35(0.98)	11.25(1.16)	**9.84(1.33)**
98	13.37(-)	12.4(1.08)	10.5(1.27)	**9.21(1.45)**
100	18.06(-)	14.78(1.22)	13.02(1.39)	**11.58(1.56)**

表4：大型网格中的运行时间和加速比。单元格中的第一个数字表示运行时间（秒），括号中的数字表示相对于 A1 的加速比。 k 表示智能体数量。

在大多数情况下，而 A4 在所有方法中运行速度最快。在所有实例中，A4 相对于 A1 的平均加速比达到 1。48。

92 个和 94 个智能体的运行时间比 90 个智能体更短。这是因为，当智能体数量较多时，运行时间不会受到智能体数量小幅增加的显著影响，而是由一些难以求解的情况所主导。

5.3 讨论

在大型网格中，性能提升比在小型网格中更加明显。我们认为这是路径长度造成的。在小型网格地图中，智能体的路径较短，可持续信息只能使用几次。而在大型地图中，路径要长得多，可持续信息的使用频率更高，因此 A4 获得了更高的加速比。

6 结论

我们提出了一种用于解决在线多智能体路径查找问题的三级算法。三个层级分别负责模拟多智能体在线环境、解决多智能体路径规划问题，以及利用历史规划信息辅助解决单智能体路径规划问题。我们证明了该方法的完备性和快照最优性。实验表明，我们提出的方法比 SOTA 算法运行得更快。在实验过程中，我们还发现，该方法在大型网格中的表现远优于在小型网格中的表现。这是因为在较大的网格中，智能体的路径更长，因此规划上下文可以重复利用更多次。这表明，路径越长，我们的方法带来的加速效果越好。

未来还将探索使用可持续信息的更多方面，例如构建冲突树，以进一步提高算法的效率。

参考文献

Choudhury, S.；Solovey, K.；Kochenderfer, M.；以及 Pavone, M. 2022。面向道路网络中无人机和卡车的协调多智能体路径查找。载于第*21*届国际自主智能体与多智能体系统会议论文集，272–280。Hart, P. E.；Nilsson, N. J.；以及 Raphael, B. 1968。启发式确定最小代价路径的形式基础。*IEEE*系统科学与控制论学报，4(2)：100–107。Ma, H.；Koenig, S.；Ayanian, N.；Cohen, L.；H¨onig, W.；Kumar, T.；Uras, T.；Xu, H.；Tovey, C.；以及 Sharon, G. 2017a。概述：多智能体路径查找向现实场景的推广。*arXiv*预印本 *arXiv:1702.05515*。Ma, H.；Tovey, C.；Sharon, G.；Kumar, T.；以及 Koenig, S. 2016。具有有效载荷转移的多智能体路径查找与包裹交换机器人路径规划问题。载于*AAAI*人工智能会议论文集，第30卷。Ma, H.；Yang, J.；Cohen, L.；Kumar, T. S.；以及 Koenig, S. 2017b。可行性研究：在拥塞的电子游戏环境中移动异质团队。载于第十三届人工智能与互动数字娱乐会议。Morris, R.；Pasareanu, C. S.；Luckow, K.；Malik, W.；Ma, H.；Kumar, T. S.；以及 Koenig, S. 2016。机场地面运行的规划、调度与监控。载于第三十届*AAAI*人工智能会议研讨会。Phillips, M.；以及 Likhachev, M. 2011。SIPP：动态环境中的安全区间路径规划。载于*2011*年*IEEE*机器人与自动化国际会议，5628–5635。IEEE。Salzman, O.；以及 Stern, R. 2020。多智能体路径查找及多智能体取送问题中的研究挑战与机遇。载于第*19*届国际自主智能体与多智能体系统会议论文集，1711–1715。Sharon, G.；Stern, R.；Felner, A.；以及 Sturtevant, N. R. 2015。面向最优多智能体路径查找的基于冲突的搜索。人工智能，219：40–66。Stern, R. 2019。多智能体路径查找——概述。人工智能，96–115。ˇSvancara, J.；Vlk, M.；Stern, R.；Atzmon, D.；以及 Bart´ak, R. 2019。在线多智能体路径查找。载于*AAAI*人工智能会议论文集，第33卷，7732–7739。Wurman, P. R.；D’Andrea, R.；以及 Mountz, M. 2008。在仓库中协调数百辆协作型自主车辆。《*AI*杂志》，29(1)：9–9。Yu, J.；以及 LaValle, S. M. 2013。图上的最优多机器人路径规划的结构与不可解性。载于第二十七届*AAAI*人工智能会议。