doi:10.19306/j.cnki.2095-8110.2022.05.006

# 基于冲突分类与消解的多智能体路径规划算法设计

王　东[1,2],于连波[1,2],曹品钊[1,2],连　捷[1,2]

(1. 大连理工大学工业装备智能控制与优化教育部重点实验室,大连 116024;
2. 大连理工大学控制科学与工程学院,大连 116024)

**摘　要**:多智能体路径规划应用广泛但求解困难。为更好地处理多智能体路径规划中的路径冲突问题,提高求解效率,将冲突进一步分类为相向顶点冲突和交叉顶点冲突,并提出了对应的消解方式。相向顶点冲突的消解方法采用提前添加约束的方式,避免在消解其冲突的过程中产生另一个可预见的冲突;交叉顶点冲突的消解方法采用寻找最佳等待时间的方式,在消解其冲突的同时消解其他存在的冲突。两种冲突消解方法均可减小约束树的规模,在一定程度上减少算法的计算量。并提出了基于冲突搜索算法的高层节点冲突搜索算法。实验结果表明,所提出的冲突分类及消解方式有效地减小了算法高层中约束树的规模,降低了算法计算量,并在智能体密集的环境下表现出更大的优势。

**关键词**:多智能体路径规划;基于冲突搜索;路径冲突;冲突分类与消解

**中图分类号**:TP242;V249　　**文献标志码**:A　　**文章编号**:2095-8110(2022)05-0056-11

# Design of a Multi-Agent Path Planning Algorithm Based on Conflict Classification and Resolution

WANG Dong[1,2], YU Lian-bo[1,2], CAO Pin-zhao[1,2], LIAN Jie[1,2]

(1. Key Laboratory of Intelligent Control and Optimization for Industrial Equipment of Ministry of Education, Dalian University of Technology, Dalian 116024, China;
2. School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China)

**Abstract**: Multi-agent path planning is widely used, however, it is difficult to plan the paths of agents because of the existence of conflicts for these agents. In order to deal with the conflicts and improve the efficiency of the planning, the conflicts are further classified into the opposite vertex conflict and the intersect vertex conflict, and the corresponding resolution methods are proposed. The method of resolving the opposite vertex conflict adopts the method of adding constraints in advance to avoid another foreseeable conflict in the process of resolving its conflict. The method of resolving the intersect vertex conflict finds the best waiting time to resolve its conflict while resolving other existing conflicts. Both conflict resolution methods can reduce the size of the constraint tree and the computational complexity of the algorithm to a certain extent. An algorithm based on a conflict-based search algorithm to search for high-level node conflict is proposed. The experimental results show that the proposed conflict classification and resolution method can effectively reduce the size of the constraint tree in the high-level algorithm and the computational complexity of

**收稿日期**: 2022-04-01;**修订日期**:2022-06-14

**基金项目**:国家重点研发计划项目(2019YFE0197700); 国家自然科学基金(61973050,62173061); 辽宁省兴辽英才计划项目(XLYC2007010);中央高校基本科研业务费(DUT20GJ209, DUT20JC14)

**作者简介**:王东(1980-),男,教授,博士,主要从事多智能体协同控制方面的研究。

the algorithm, and shows advantages in the dense environment of agents.

**Key words**: Multi-agent path planning; Conflict-based search; Path conflict; Conflict classification and resolution

## 0　引言

多智能体路径规划问题是在已知的环境地图中,为一组智能体规划从各自的起始点到相应目标点的无冲突路径,并实现路径代价总和最小或者最大完工时间最小等优化目标[1]。多智能体路径规划已经广泛应用于各个领域,包括自主仓库管理[2]、多智能体运动规划[3]和无人机集群避障[4-5]等。多智能体路径规划问题的求解困难程度远大于多个单智能体路径规划问题的叠加,两者的本质区别是多智能体路径规划问题是在规划单个智能体路径的同时消解智能体间的路径冲突。消解冲突是多智能体路径规划问题的关键[6]。

目前主流的一类多智能体路径规划方法为:先分别规划单个智能体的最优路径,然后再消解多智能体之间的路径冲突。冲突消解方案主要有优先级规划法[7-8]、基于行为的避碰方法[9]、分层地图法[10]以及交通规则法[11]等。这些冲突消解方案简单易行,能够有效消解多智能体间的路径冲突。但上述方案多会牺牲路径长度或智能体运行时间,难以保证多智能体路径规划方案的最优性。

基于冲突搜索(Conflict-Based Search,CBS)的算法[12]是一种经典的多智能体路径规划方法,因其具有最优性和完备性两大优点近年来受到广泛研究。CBS 是一种两层搜索算法,算法高层用于找到多个智能体之间的冲突,算法低层将多智能体路径规划问题视为多个单智能体在满足约束下的路径规划。CBS 在大地图和智能体耦合度不高的情况下具有较高的求解效率。文献[13]提高了 CBS 算法的速度,但求得的解只能保证次优性。E. Boyarski 等[14]提出了改进的基于冲突搜索算法(Improved Conflict-Based Search,ICBS),将路径冲突分为关键冲突、半关键冲突和非关键冲突,并对冲突的消解顺序进行优化。ICBS 算法保留了标准 CBS 算法的最优性和完备性,该算法中使用多值决策图对冲突类型进行判断,计算量较大。A. Felner 等在关键冲突的基础上,设计了带有启发值的代价函数,减少算法中节点的扩展数量,从而降低算法的计算量[15]。J. Li 等规定了一种新的冲突类型,即在网格地图内的某些冲突将必然发生在一个矩形,并提出了相应的检测和消解方法[16]。上述算法多是在 CBS 的高层进行改进,文献[17]在标准 CBS 的低层使用了增量式路径规划算法,从智能体的路径重规划方面减少了算法的计算量。

本文设计了一种基于冲突分类与消解的多智能体路径规划算法,以解决多智能体路径规划及路径冲突问题。本文在 ICBS 算法的基础上,做出以下改进:首先,将多智能体路径中出现的关键冲突类别进一步分类出方向冲突,方向冲突包括相向冲突和交叉冲突;然后,提出方向冲突中相向冲突和交叉冲突的消解方式;最后,提出 ICBS 高层节点中的冲突搜索算法。本文算法保留了 CBS 算法的最优性和完备性,并通过新提出的冲突分类及消解方式,减少了 ICBS 算法高层中约束树的规模,从而减少算法在高层中搜索的时间,一定程度上降低了算法的计算量。

## 1　问题描述和算法简介

### 1.1　多智能体路径规划问题模型

多智能体路径规划问题[12]包括一个给定的无向无权图 $\boldsymbol{G}=(V,E)$ 和多个智能体 $A=\{a_1,a_2,\cdots,a_n\}$。无向无权图 $\boldsymbol{G}=(V,E)$ 中,$V$ 是图中顶点的集合,$E$ 是图中顶点间连接边的集合。任意一个智能体 $a_i\in A$,拥有唯一的起始顶点 $s_i\in V$ 和目标顶点 $g_i\in V$。在多智能体路径规划问题中,时间被离散化为时刻序列 $T=\{0,1,2\cdots\}$,在任意一个时刻智能体在当前顶点等待或者移动到相邻顶点,移动和等待两种动作都需要单位代价。智能体 $a_i$ 的路径 $p_i$ 被描述为一个从时间序列 $T$ 到顶点集合 $V$ 映射的序列 $p_i=\{v_i(0),v_i(1),\cdots,v_i(t),\cdots,v_i(c_i)\}$,其中 $v_i(t)$ 表示智能体 $a_i$ 在 $t$ 时刻占用的顶点,$c_i$ 表示路径 $p_i$ 的长度,路径重规划后,$c_i$ 的值会随着路径 $p_i$ 的变化而变化。本文假设智能体保持匀速运行,使用 $c_i$ 表示智能体 $a_i$ 的路径代价。智能体路径间不能发生冲突是多智能体路径规划问题的重要约束条件。冲突约束具体描述为:1) 任意两个智能体 $a_i$ 和 $a_j$ 不能在同一时刻占用同一个顶点,即 $v_i(t)\neq v_j(t),\forall a_i,a_j\in A\wedge i\neq j$;2) 任意两个智能体 $a_i$

和$a_j$不能在$t$时刻和$t+1$时刻间交换彼此占用的顶点，即$v_i(t)\neq v_j(t+1)\vee v_i(t+1)\neq v_j(t)$，$\forall a_i$，$a_j\in A\wedge i\neq j$。多智能体路径规划任务是为智能体$A=\{a_1,a_2,\cdots,a_n\}$规划一组从起始顶点到目标顶点的无冲突路径$P=\{p_1,p_2,\cdots,p_n\}$，同时实现路径代价总和最小的优化目标，其中优化目标形式化可表示为

$$\min_{p_i}\sum_{i=1}^{n}c_i(p_i) \tag{1}$$

### 1.2 CBS 算法

CBS 算法是解决多智能体路径规划问题的重要算法之一。CBS 算法拥有两层结构，其中低层用于规划多个单智能体的最优路径，高层用于解决多个智能体之间存在的路径冲突。CBS 算法引入了约束和冲突的概念，约束用一个元组$(a_i,s,t)$表示，表示禁止智能体$a_i$在$t$时刻占用元素$s$，$s$可以是一个顶点或者是一个边，如果表示一个顶点$v$，则该约束表示不允许$a_i$在$t$时刻占用该顶点，进一步可以表示为$(a_i,v,t)$；如果是一条边$(u,v)$，则表示禁止$a_i$在$t$时刻到$t+1$时刻采取的动作占用该边，进一步可以表示为$(a_i,u,v,t)$。冲突用一个元组$(a_i,a_j,s,t)$表示，表示智能体$a_i$和$a_j$在$t$时刻同时占用元素$s$；相应地，冲突可以分为顶点冲突和边冲突，分别表示为$(a_i,a_j,v,t)$和$(a_i,a_j,u,v,t)$。

在 CBS 算法的高层构建一个二叉结构的约束树，约束树中的每个节点$N$包括约束、解和代价三部分。约束树中根节点的约束为空集，每个节点从父节点中继承约束。高层搜索每次从优先级队列 OPEN 中选取代价值最低的节点进行扩展，若当前节点的解是一组无冲突路径，则将当前节点作为目标节点，并将该节点的解作为问题的解；否则，针对当前节点中的路径冲突，分别添加约束扩展出左右子节点，并添加到 OPEN 中，重复扩展和添加 OPEN 中的节点，直到得到具有无冲突解的目标节点。CBS 算法低层规划单个智能体满足约束的最优路径，即为高层节点提供解。低层的单智能体路径规划使用任何路径规划算法理论上都可满足 CBS 的需求，本文使用 A* 算法。CBS 算法对于优化路径而言代价是最优的，证明可见文献[12]。

### 1.3 ICBS 算法

CBS 算法高层随机选择冲突进行节点扩展，但不恰当的选择会增加约束树的规模，从而增加算法的运行时间。ICBS 通过两项改进改善了约束树节点规模较大的问题。不同于 CBS 算法的随机选择冲突进行节点扩展，ICBS 算法将冲突进行优先级划分，分为关键冲突、半关键冲突和非关键冲突。CBS 算法对关键冲突拆分出的两个子节点的代价值均大于当前节点的代价值，对半关键冲突拆分出的两个子节点中有且仅有一个子节点的代价值大于当前节点的代价值，对非关键冲突拆分出的两个子节点的代价值均不大于当前节点的代价值。通过优先解决关键冲突和半关键冲突，最后是非关键冲突，在一定程度上加快了算法求解速度。在遇到关键冲突时，ICBS 和 CBS 一样以拆分出该节点的左右子节点的方式来解决，在消解半关键冲突或非关键冲突时不直接将该节点进行拆分，而是试图通过寻找有效的绕路使冲突得到解决[14]。因此，在 ICBS 算法中，高层节点中除了 CBS 算法中包含的约束、解和代价三部分以外，还包括该节点中存在冲突个数。

**注解 1**：CBS 算法[12]及其改进算法[14]侧重消解多智能体间的路径冲突，通过智能体间中心互联式的通信拓扑结构由算法高层直接进行冲突检测。该类算法注重规划满足约束的无冲突路径的最优性，即路径长度最短。文献[18-19]中的适航性、使用 Bezier 曲线处理单个智能体路径的平滑性等，本文算法暂不进行研究。

## 2 冲突分类与消解算法

CBS 算法和 ICBS 算法能够为多智能体路径规划问题求解出最优解，但算法单独地处理当前冲突，没有考虑当前冲突的处理与后续路径或其他路径冲突的关系。本章基于当前冲突处理对后续冲突的影响以及优先处理关键冲突对其他路径冲突的影响，对路径冲突进行更加详细的分类并提出相应冲突消解方案。

### 2.1 冲突分类

在四连通二维栅格地图中，现有的 CBS 算法消解关键顶点冲突的最佳方案是某一个智能体在冲突发生时刻前进行等待。该方案在消解当前冲突的同时，对于某些特殊类型的关键顶点冲突可能会引起一个新的可预见的冲突，而该新发生的冲突是完全可以避免的。关键冲突具有优先消解权限，智能体在冲突发生前的任意一个时刻等待，均能够消解关键顶点冲突。选择合适的等待时刻，能够在消解当前关键顶点冲突的同时消解其他的半关键冲

突或非关键冲突。根据上述分析，本文在 ICBS 算法的基础上，根据发生冲突的两个智能体的运行方向，将关键顶点冲突进一步分为相向顶点冲突和交叉顶点冲突。本文将冲突定义为 $C=\{cft,type\}$，其中 $cft=(a_i,a_j,s,t)$ 表示传统 CBS 算法中定义的冲突，$type=\{P_t,D_t\}$ 用于描述冲突类型，$P_t$ 表示优先级冲突类型，$D_t$ 表示方向冲突类型。

2.1.1　相向顶点冲突

顶点冲突 $(a_i,a_j,v,t)$ 中，若智能体 $a_i$ 和 $a_j$ 的运行方向完全相反，即智能体 $a_i$ 和 $a_j$ 通过冲突顶点 $v$ 后交换占用的顶点，则在本文中该类冲突被定义为相向顶点冲突。通过等待的方式消解该类冲突后，必然会引发一个新的边冲突。下面以图 1 所示的案例及图 2 中对应的约束树直观展示相向顶点冲突。

图 1 中，智能体 $a_i$ 从起始顶点 $A_2$ 出发去往目标顶点 $A_5$，智能体 $a_j$ 从起始顶点 $A_4$ 出发去往目标顶点 $A_1$。使用 CBS 算法及 ICBS 算法为智能体 $a_i$ 和智能体 $a_j$ 规划路径，分别为智能体 $a_i$ 初始规划最优路径 $p_i=\{A_2,A_3,A_4,A_5\}$，为智能体 $a_j$ 初始规划最优路径 $p_j=\{A_4,A_3,A_2,A_1\}$。智能体 $a_i$ 和智能体 $a_j$ 在 1 时刻共同占用顶点 $A_3$，发生冲突 $C_1=(a_i,a_j,A_3,1)$；在 0 时刻智能体 $a_i$ 占用顶点 $A_2$，智能体 $a_j$ 占用顶点 $A_4$；在 2 时刻智能体 $a_i$ 占用顶点 $A_4$，智能体 $a_j$ 占用顶点 $A_2$，即智能体 $a_i$ 和智能体 $a_j$ 在冲突发生的前后时刻交换彼此占用的顶点。

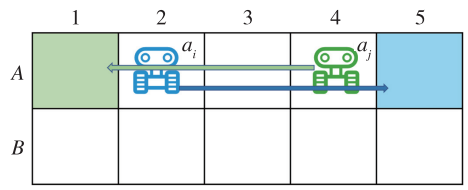


**图 1　相向顶点冲突案例**

**Fig. 1　A case of the opposite vertex conflict**

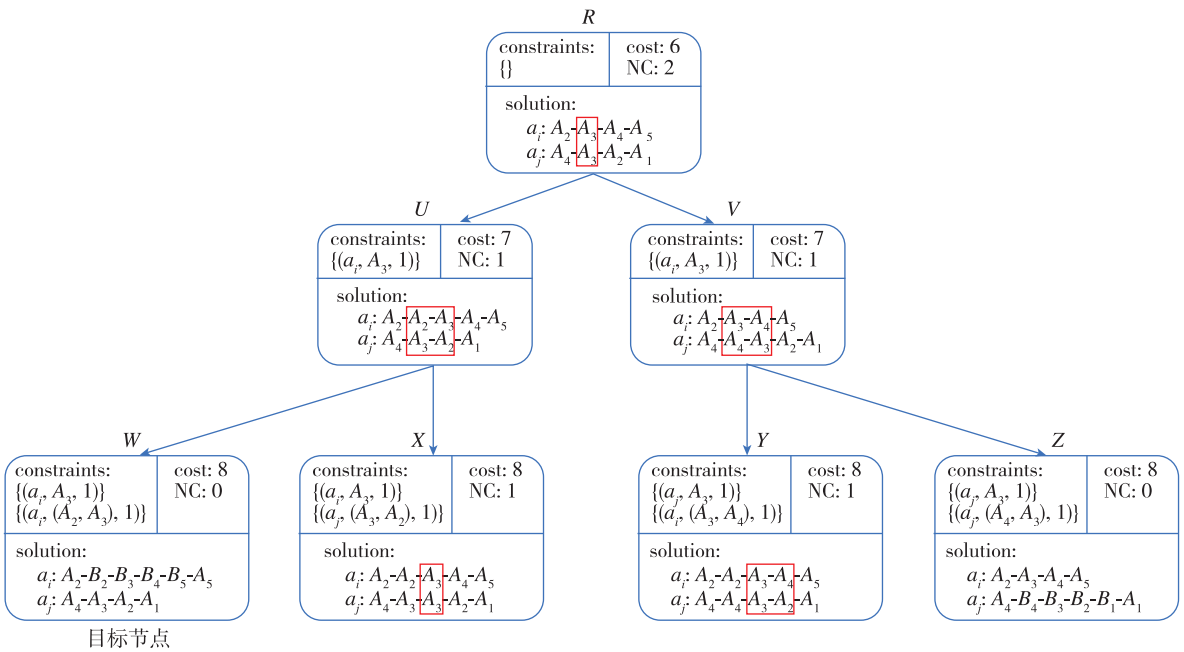


**图 2　相向顶点冲突案例的约束树**

**Fig. 2　A constraint tree of the opposite vertex conflict case**

从图 2 所示约束树的角度来看，依据初始化路径构建约束树的根节点 $R$ 后，检测到根节点 $R$ 中存在顶点冲突 $C_1=(a_i,a_j,A_3,1)$，如图 2 根节点 $R$ 中红色边框标注所示。为消解顶点冲突 $C_1$，添加约束 $(a_i,A_3,1)$ 和 $(a_j,A_3,1)$ 拆分根节点，扩展出左右子节点 $U$ 和 $V$。虽然子节点 $U$ 和 $V$ 不存在顶点冲突 $C_1=(a_i,a_j,A_3,1)$，但两个子节点中都产生了新的边冲突 $C_2=(a_i,a_j,A_2,A_3 1)$ 或 $C'_2=(a_i,a_j,A_4,A_3,1)$。为消解该冲突必须再一次进行添加约束和扩展节点的操作，如图 2 中由节点 $U$ 和 $V$ 扩展出节点 $W$、$X$、$Y$ 和 $Z$ 所示。

本文将图 1 和图 2 展示的在冲突时刻前后交换彼此占用的顶点，并且无法通过一次节点扩展而消解的特殊顶点冲突称为相向顶点冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$，表示智能体 $a_i$ 和 $a_j$ 在 $t$ 时刻共同占用顶点 $v$，在 $t-1$ 时刻和 $t+1$ 时刻交换占用顶点 $u$ 和 $w$，

下面给出相向顶点冲突的定义。如果一个冲突满足以下条件，那么称其为相向顶点冲突：

1）冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$ 中智能体 $a_i$ 和 $a_j$ 将在 $t$ 时刻同时占用顶点 $v$，即冲突 $C_{ov}$ 为顶点冲突；

2）冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$ 是关键冲突；

3）冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$ 中智能体 $a_i$ 和 $a_j$ 将交换彼此在 $t-1$ 时刻和 $t+1$ 时刻占用顶点 $u$ 和 $w$，即智能体 $a_i$ 将在 $t+1$ 时刻占用智能体 $a_j$ 在 $t-1$ 时刻占用的顶点 $w$，并且智能体 $a_j$ 将在 $t+1$ 时刻占用智能体 $a_i$ 在 $t-1$ 时刻占用的顶点 $u$。

2.1.2 交叉顶点冲突

交叉顶点冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 是相对于相向顶点冲突的概念，两者的区别是交叉顶点冲突中智能体 $a_i$ 和 $a_j$ 通过冲突顶点 $v$ 后不会交换彼此占用顶点，如图 3 中智能体 $a_i$ 和 $a_j$ 所示。智能体 $a_i$ 和 $a_j$ 的初始化路径为 $p_i=\{B_1,B_2,B_3,B_4\}$ 和 $p_j=\{A_2,B_2,C_2,C_3\}$。智能体 $a_i$ 和 $a_j$ 在 1 时刻共同占用顶点 $B_2$，即存在顶点冲突 $C_2=(a_i,a_j,B_2,1)$，但在 0 时刻和 2 时刻两者不会交换彼此占用的顶点。

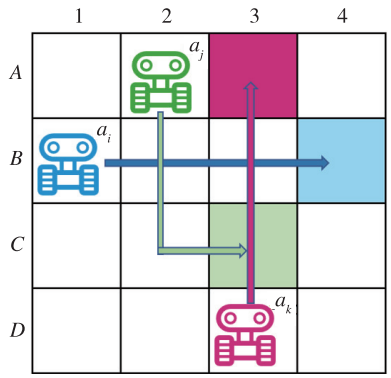


**图 3 交叉顶点冲突案例**

**Fig. 3 A case of the intersect vertex conflict**

下面给出交叉顶点冲突的定义，如果一个冲突满足以下条件，那么称其为交叉顶点冲突：

1）冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 中智能体 $a_i$ 和 $a_j$ 将在 $t$ 时刻同时占用顶点 $v$，即冲突 $C_{iv}$ 为顶点冲突；

2）冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 是关键冲突；

3）冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 中智能体 $a_i$ 和 $a_j$ 将不交换彼此在 $t-1$ 时刻和 $t+1$ 时刻占用的顶点 $u$ 和 $w$，即智能体 $a_i$ 在 $t+1$ 时刻不占用智能体 $a_j$ 在 $t-1$ 时刻占用的顶点 $w$，或者智能体 $a_j$ 在 $t+1$ 时刻不占用智能体 $a_i$ 在 $t-1$ 时刻占用的顶点 $u$。

## 2.2 冲突消解

2.2.1 相向顶点冲突消解方案

消解相向顶点冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$ 时，若采用 CBS 算法或者其改进算法，则首先针对其中的顶点冲突 $(a_i,a_j,v,t)$ 添加一个顶点约束 $(a_i,v,t)$（或 $(a_j,v,t)$）进行一次节点扩展，然后对随之产生的边冲突 $(a_i,a_j,u,v,t)$（或 $(a_i,a_j,v,w,t)$）添加约束 $(a_*,(u,v),t)$（或 $(a_*,(v,w),t)$）进行第二次的节点扩展。简言之，CBS 算法及其改进算法至少需要进行两次连续的节点扩展才能将相向顶点冲突完全消解，如图 2 所示。本文消解相向顶点冲突的主要思想是提前为冲突整体添加一个约束，仅进行一次节点扩展便可完全消解相向顶点冲突。具体步骤描述为：

1）检测冲突并判断其为相向顶点冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$。

2）针对相向顶点冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$，直接分别添加 4 个组合约束 $(a_i,u,v,t)$、$(a_j,v,w,t)$、$(a_i,v;a_j,u,v,t)$ 和 $(a_j,v;a_i,v,w,t)$ 拆分相向顶点冲突 $C_{ov}$ 所在的节点 $N$，扩展出 4 个子节点。其中，组合约束 $(a_i,u,v,t)$ 表示智能体 $a_i$ 在 $t$ 时刻不得占用顶点 $u$ 和 $v$；组合约束 $(a_j,v,w,t)$ 表示智能体 $a_j$ 在 $t$ 时刻不得占用顶点 $v$ 和 $w$；组合约束 $(a_i,v;a_j,u,v,t)$ 表示智能体 $a_i$ 在 $t$ 时刻不得占用顶点 $v$，并且智能体 $a_j$ 在 $t$ 时刻不得占用顶点 $u$ 和 $v$；组合约束 $(a_j,v;a_i,v,w,t)$ 表示智能体 $a_j$ 在 $t$ 时刻不得占用顶点 $v$，并且智能体 $a_i$ 在 $t$ 时刻不得占用顶点 $v$ 和 $w$。

3）新生成的子节点分别调用低层路径规划算法，得到消解相向顶点冲突 $C_{ov}$ 后的路径。

下面以图 1 中的案例具体展示相向顶点冲突的消解方案。约束树根节点 $R$ 的构建与 ICBS 算法相同。检测冲突，进一步判断出其为相向顶点冲突 $C_{ov1}=(a_i,a_j,A_2,A_3,A_4,1)$，如图 4 中 $R$ 节点的红色边框标注部分。对冲突 $C_{ov1}$ 现有的顶点冲突和下一步必然会发生的边冲突直接整体添加两顶点约束 $(a_i,A_2,A_3,1)$ 和 $(a_j,A_4,A_3,1)$ 以拆分根节点 $R$，扩展出左右子节点，分别如图 4 中的 $U'$ 和 $V'$ 所示。图 4 明确展现出经过上述一次节点扩展，即可将相向顶点冲突 $C_{ov1}$ 完全消解。

对比图 2 和图 4 可知，本文设计的冲突消解方

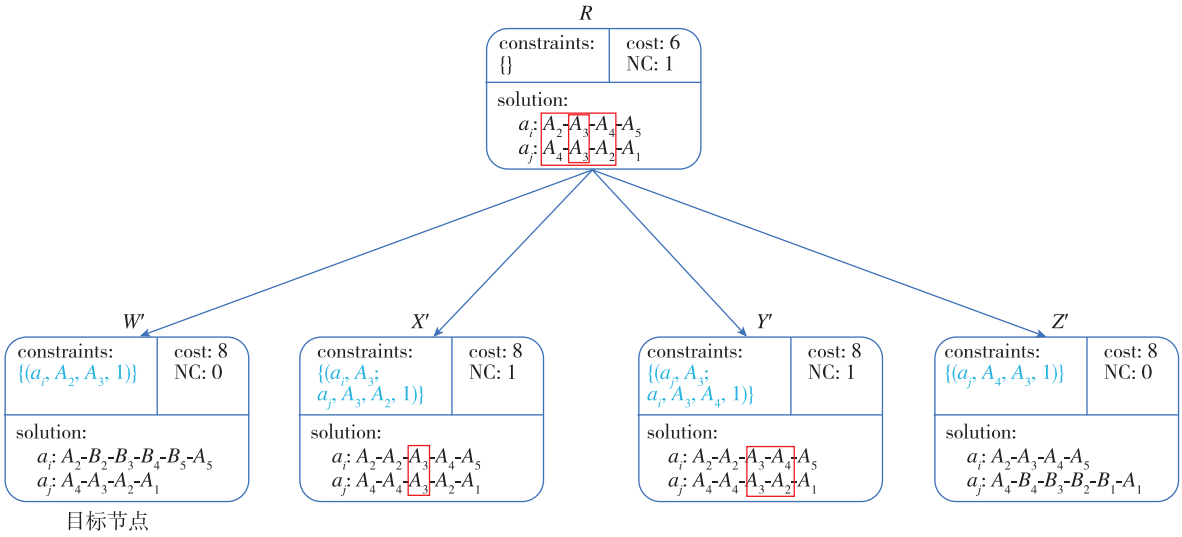


**图 4　相向顶点冲突案例的本文方案约束树**

**Fig. 4　A constraint tree of the designed scheme for the opppsite vertex conflict case**

案与 ICBS 算法所求目标节点的路径代价值相同，并且本文设计方案的约束树节点扩展数量小于 ICBS 算法。表 1 在算法的高层扩展节点数、高层生产节点数以及低层算法调用次数方面具体展示本文设计方案的优势。

**表 1　相向顶点冲突案例算法指标对比**

**Tab. 1　Comparisons of algorithm indexes for the opposite vertex conflict case**

| 算法 | 高层扩展节点数 | 高层生成节点数 | 低层算法调用次数 |
|---|---|---|---|
| ICBS | 4 | 7 | 10 |
| 本文算法 | 2 | 5 | 8 |

**注解 2**：对于相向顶点冲突 $C_{ov}=(a_i,a_j,u,v,w,t)$，CBS 算法及其改进算法至少需要两次连续的节点扩展才能完全消解冲突。本文设计方案仅需一次节点扩展便能完全消解冲突。本文设计方案在路径代价方面与 ICBS 算法具有相同的最优性，并且对计算量要求较低，能够减小算法搜索空间规模和算法搜索时间。

2.2.2　交叉顶点冲突消解方案

交叉顶点冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 的最佳消解方案是智能体 $a_i$ 或者 $a_j$ 在冲突发生时刻 $t$ 前等待一个时刻，在 $\{0,\cdots,t-1\}$ 范围内的任何一个时刻等待都满足消解冲突的约束条件。CBS 算法及其改进算法依据低层算法的路径规划原理，默认选择在冲突发生时刻 $t$ 的前一个时刻等待。在 $t$ 时刻前存在其他半关键冲突或者非关键冲突时，此种默认选择等待时刻 $t_w=t-1$ 的方式仅能消解当前交叉顶点冲突 $C_{iv}$，无法同时消解其他冲突。由图 3 案例以及图 5 中使用 ICBS 算法为例，规划多智能体路径的约束树，进一步展示默认选择等待时刻的不足之处。

图 3 中，智能体 $a_i$、$a_j$ 和 $a_k$ 存在的起始顶点分别为 $B_1$、$A_2$ 和 $D_3$，目标顶点分别为 $B_4$、$C_3$ 和 $A_3$，初始化最优路径为 $p_i=\{B_1,B_2,B_3,B_4\}$、$p_j=\{A_2,B_2,C_2,C_3\}$ 和 $p_k=\{D_3,C_3,B_3,A_3\}$。由冲突检测得到智能体 $a_i$ 和 $a_k$ 发生交叉顶点冲突 $C_{iv1}=(a_i,a_k,\bar{B}_2,B_3,\bar{C}_3,2)$，智能体 $a_i$ 和 $a_j$ 之间发生半关键冲突 $C_2=(a_i,a_j,B_2,1)$，如图 5 中 $R$ 节点所示。对交叉顶点冲突 $C_{iv1}=(a_i,a_k,\bar{B}_2,B_3,\bar{C}_3,2)$ 添加约束 $(a_i,B_3,2)$ 后，智能体 $a_i$ 在 1 时刻对应顶点 $B_2$ 处等待，消解交叉顶点冲突 $C_{iv1}=(a_i,a_k,\bar{B}_2,B_3,\bar{C}_3,2)$，但是智能体 $a_i$ 和 $a_j$ 之间发生半关键冲突 $C_2=(a_i,a_j,B_2,1)$ 仍然存在，如图 5 中 $U$ 节点所示。消解冲突 $C_2=(a_i,a_j,B_2,1)$ 需要再次进行添加约束和节点扩展，如图 5 中 $U$ 节点扩展出 $W$ 节点和 $Z$ 节点所示。$V$、$X$、$Z$ 节点情况与此类似。

本文设计了针对交叉顶点冲突的消解方案，选择合适的智能体，让其在恰当等待时刻 $t_w$ 进行等待，保证在消解当前交叉顶点冲突 $C_{iv}$ 的同时消解其他半关键冲突或者非关键冲突，进而减小算法搜索空间以及算法搜索时间。

交叉顶点冲突的消解方案描述如下：

1）交叉顶点冲突 $C_{iv}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 使用

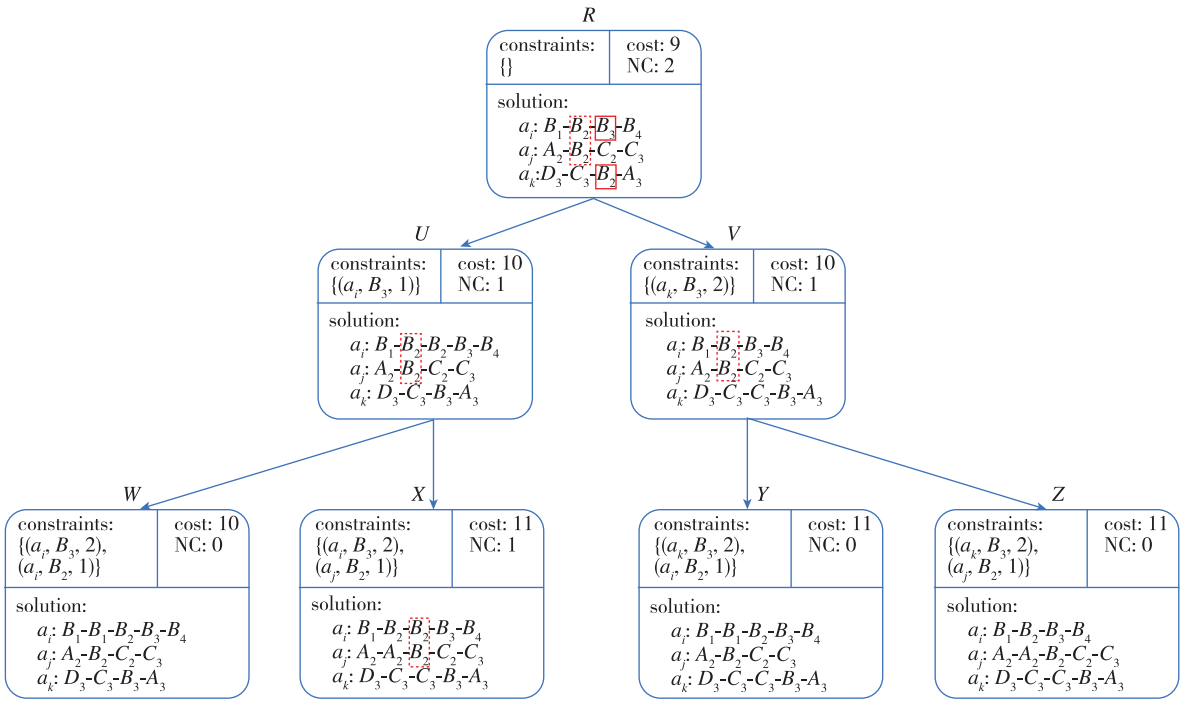


**图 5 交叉顶点冲突案例的约束树**

**Fig. 5 A constraint tree for the intersect vertex conflict case**

让智能体 $a_i$ 或 $a_j$ 在原路径的 $t_w$ 时刻对应顶点处等待的方式进行消解。

2)选择智能体 $a_i$:交叉顶点冲突 $C_{\text{iv}}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 中智能体 $a_i$ 在 $t$ 时刻前共存在 $N_i^t$ 个半关键冲突和非关键冲突,智能体 $a_j$ 在 $t$ 时刻前共存在 $N_j^t$ 个半关键冲突和非关键冲突,则优先选择智能体 $a_i$ 添加约束 $(a_i,v_{t_w},t_w)$ 扩展节点,其中 $N_i^t \geqslant N_j^t$,$v_{t_w}$ 表示 $a_i$ 在 $t_w$ 时刻所处的顶点位置。

3)确定等待时刻 $t_w$:假设智能体 $a_i$ 在 $t$ 时刻前发生的 $N_i^t$ 个半关键冲突和非关键冲突的时刻分别为 $t_1,t_2,\cdots,t_{N_i^t}$,则 $t_w \in \{0,\cdots,t_{\min}\}$,其中 $t_{\min}=\min\{t_1,t_2,\cdots,t_{N_i^t}\}$。

使用本文设计消解方案为图 3 案例规划多智能体路径的约束树如图 6 所示。图 6 中 $R$ 节点扩展出 $U'$ 节点时添加的约束与图 5 中约束树完全相同均为 $(a_i,B_3,2)$,不同之处是图 6 约束树按照本文冲突消解方案选择智能体 $a_i$ 在 0 时刻等待,对应的节点为 $B_1$。经过一次节点扩展 $U'$ 节点中智能体 $a_i$ 和 $a_k$ 间的交叉顶点冲突 $C_{\text{iv1}}=(a_i,a_k,\bar{B}_2,B_3,\bar{C}_3,2)$ 以及智能体 $a_i$ 和 $a_j$ 之间的半关键冲突 $C_2=(a_i,a_j,B_2,1)$ 均被消解掉。

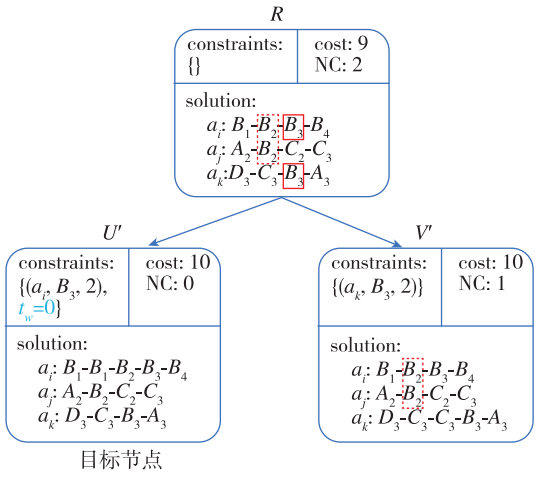


**图 6 交叉顶点冲突案例的本文方案约束树**

**Fig. 6 A constraint tree of the designed scheme for the intersect vertex conflict case**

对比图 5 和图 6 可知,本文设计的交叉顶点冲突消解方案与 ICBS 算法所求目标节点的路径代价值相同,并且本文设计方案的约束树节点扩展数量小于 ICBS 算法。表 2 在算法的高层扩展节点数、高层生成节点数以及低层算法调用次数方面具体展示本文设计方案的优势。

**表 2　交叉顶点冲突案例算法指标对比**

**Tab. 2　Comparisons of algorithm indexes for the intersect vertex conflict case**

| 算法 | 高层扩展结点数 | 高层生成节点数 | 低层算法调用次数 |
|---|---|---|---|
| ICBS | 4 | 7 | 9 |
| 本文算法 | 2 | 3 | 5 |

**注解 3**：消解相向顶点冲突 $C_{\mathrm{iv}}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 的方案为某个整体在冲突时刻 $t$ 前等待。CBS 算法及其改进算法默认选择等待时刻 $t_w=t-1$ 的方式仅能消解当前交叉顶点冲突 $C_{\mathrm{iv}}$，无法同时消解其他冲突。本文设计的相向顶点冲突 $C_{\mathrm{iv}}=(a_i,a_j,\bar{u},v,\bar{w},t)$ 消解方案选择冲突数量多的智能体在恰当的等待时刻 $t_w$ 对应的顶点处等待，消解当前交叉顶点冲突 $C_{\mathrm{iv}}$ 的同时消解其他半关键冲突或非关键冲突。对比 ICBS 算法，本文方案能够求得路径代价相同的目标节点，并有效减小算法搜索空间规模和算法搜索时间。

### 2.3　基于冲突分类与消解的多智能体路径规划算法

综合本文提出的相向顶点冲突和交叉顶点冲突分类与消解方案，根据 CBS 算法及其改进算法，现给出基于冲突分类与消解的多智能体路径规划算法，如算法 1 所示。输入一个多智能体路径规划实例后，首先用低层 A* 算法分别为每个智能体规划最优路径并计算路径代价，进而构建出根节点 $R$，将根节点 $R$ 插入到 OPEN 列表，如算法第 1～2 行所示。第 3～14 行是算法检测冲突和添加约束消解冲突的主体部分。第 5 行调用搜索高层节点中的冲突算法检测冲突，并根据本文给出的相向顶点冲突和交叉顶点冲突分类判断冲突类型，具体见算法 2。第 9～13 行根据本文所提的方向冲突添加不同的约束进行冲突消解，其中第 9、10 行是相向顶点冲突的检测与约束添加，第 11、12 行是交叉顶点冲突的检测与约束添加，第 13 行生成新的子节点，具体见算法 3。算法 1 最终为输入的多智能体路径规划实例输出一组无冲突最优路径。

算法 2 展示了高层节点冲突检测与类型判断方案，检测本文所提的方向冲突并将冲突返回给算法 1。算法 2 中增加了半关键和非关键冲突列表以及智能体冲突表。半关键和非关键冲突列表分别保存各智能体发生的半关键和非关键冲突，智能体冲突表保存各智能体的等待时刻 $t_w$，为消解交叉顶点冲突提供相应信息。

```
算法 1：基于改进冲突搜索的冲突分类与消解算法
    输入：一个多智能体路径规划实例
    输出：该实例的一个最优解
1   调用低层 A* 算法为每个智能体规划初始最优路径
2   生成根节点 R，并将其放入 OPEN 列表
3   while OPEN 列表非空时 then
4     从 OPEN 中取出代价值最低的节点，作为当前节点 N
5     调用搜索高层节点中的冲突算法检测冲突
6     if 节点 N 中没有冲突且不是关键冲突 then
7       调用低层 A* 算法重规划绕行路径
8     else 节点 N 中存在冲突且是关键冲突 then
9       if 相向顶点冲突 then
10        添加组合约束
11      else 交叉顶点冲突 then
12        添加约束并确定智能体等待时刻
13      生成子节点 N′并将其放入 OPEN 列表
14  节点 N 中没有冲突
15  返回路径规划的解
```

```
算法 2：搜索高层节点中的冲突算法
    输入：一个约束树中的节点 N 的解
    输出：节点 N 中存在的冲突 C
1   for 智能体中的任意一个组合对 (a_i,a_j) then
2     if 智能体 a_i 和 a_j 间存在冲突 C then
3       if 冲突 C 是关键冲突 then
4         判断相向顶点冲突或交叉顶点冲突
5         返回 C
6       else if 冲突 C 是半关键冲突 then
7         将 C 放入半关键冲突列表
8       else then
9         将 C 放入非关键冲突列表
10      更新智能体冲突表
11      返回第一个半关键或非关键冲突
```

算法 3 展示了用约束集合生成新的子节点的算法，和 ICBS 算法不同之处在于添加的约束是本文设计的针对冲突类型消解方案的组合约束。

```
算法 3：生成新的子节点
    输入：一个约束树中的节点 N 和对智能体添加的约束
    输出：生成的新的子节点
1   将对智能体添加的约束加入到节点 N 原有的约束
2   调用低层路径规划算法更新智能体路径
3   生成新的子节点 N′
```

## 3 实验与仿真

本章进行对比实验，以验证本文设计算法求解多智能体路径规划问题的优越性。首先，本文设计的基于冲突分类与消解的多智能体路径规划算法(ICBS-DC)与采用优先级方法消解路径冲突、使用蚁群算法规划智能体路径的多智能体路径规划方法(Ant-PP)相比，展现出求解结果质量的优越性，即求解结果路径代价方面的优越性。然后，将本文算法与 CBS 算法以及 ICBS 算法进行对比，体现本文算法在计算量方面的优势。本章所有实验的代码均采用 Python 编写，均在参数为 2.3GHz Intel Core i5-6300HQ 8GB RAM 的笔记本电脑上进行。使用大小为 8×8 的四连通二位栅格地图，其中障碍物占比为地图大小的 20%，在每次实验中障碍物均随机生成。对比实验中智能体数量从 1 到 15 逐个增加，并且在每个智能体数量下随机生成 100 个多智能体路径规划案例，取各项实验数据的平均值作为最终数据。

图 7 和图 8 分别展示了本文 ICBS-DC 算法和使用优先级的 Ant-PP 算法的成功率与路径规划方案的路径代价。其中，成功率采用文献[12]中描述的在 5min 时间限制内，算法能够求解出结果的案例数量与案例总数量的比值。由图 7 和图 8 可知，本文的 ICBS-DC 算法的成功率与路径规划方案的路径代价均优于 Ant-PP 算法，并且随着智能体数量的增加，ICBS-DC 算法的优势越加明显。

为了验证本文算法在计算量方面的优势，将本文 ICBS-DC 算法与 CBS 及 ICBS 进行对比实验，分别比较三种算法的成功率、运行时间、高层约束树节点扩展数量以及低层路径规划算法平均调用的次数，结果如图 9～图 12 所示。

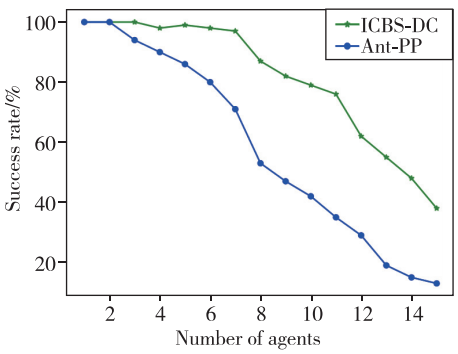


图 7 ICBS-DC 和 Ant-PP 成功率对比

Fig. 7 Success rates of ICBS-DC and Ant-PP algorithms

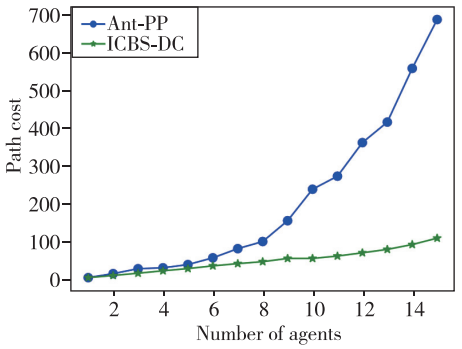


图 8 ICBS-DC 和 Ant-PP 路径代价对比

Fig. 8 Path costs of ICBS-DC and Ant-PP algorithms

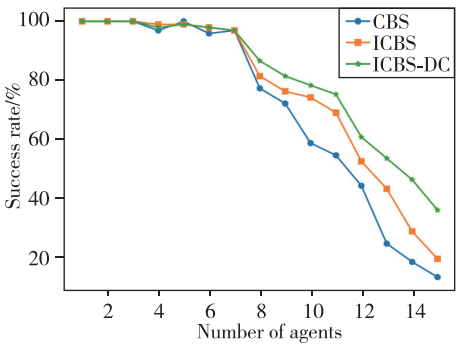


图 9 三种算法成功率对比

Fig. 9 Success rates of three algorithms

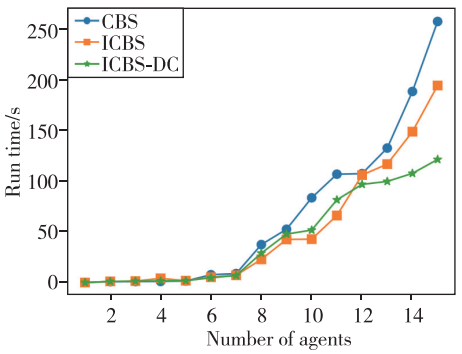


图 10 三种算法运行时间对比

Fig. 10 Run times of three algorithms

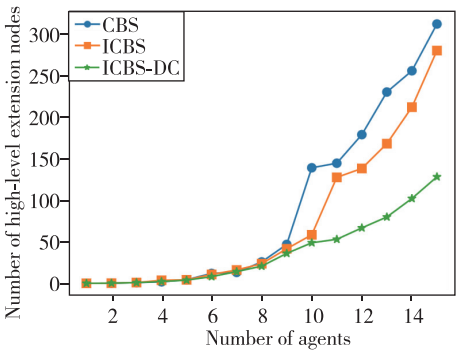


图 11 三种算法节点扩展数量对比

Fig. 11 Number of extension nodes of three algorithms

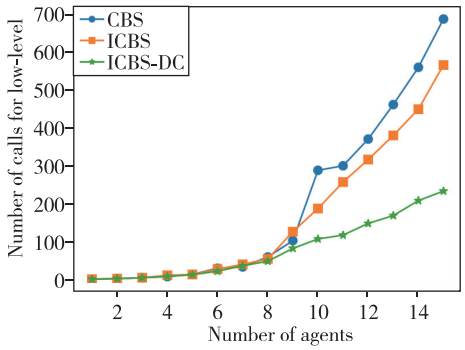


**图 12　三种算法调用低层路径规划次数对比**

**Fig. 12　Number of calls for low-level of three algorithms**

当智能体数量较少时，各个智能体的路径之间发生冲突的可能性较小，由图 9～图 12 所示结果可以看出，三种算法在成功率、平均运行时间、高层约束树节点扩展数量和低层路径规划算法调用的次数等方面几乎具有相当的性能。当智能体数量较多时，各智能体的路径在环境中发生更多的冲突，由图 9 报告的结果可知，CBS 算法在智能体数量较多时，其解决多智能体路径规划问题的成功率下降速度最快，其次是 ICBS。ICBS-DC 相比 CBS 和 ICBS 而言，在智能体密度较大的情况下具有更高的成功率。而就图 10 报告的平均运行时间而言，ICBS-DC 运行速度虽然比 CBS 快，但相较于 ICBS 在某些情况下速度稍慢，然而随着智能体数量的不断增加，智能体的路径之间冲突的可能性增加，ICBS-DC 算法的平均运行时间的增长速度与 CBS 和 ICBS 相比较为平缓，说明 ICBS-DC 在智能体比较密集的环境下展现出更好的运行时间优势。

由图 11 和图 12 展现的高层和低层的指标来看，ICBS-DC 在高层约束树节点扩展数量及低层路径规划次数方面均少于 CBS 和 ICBS，但这正是 ICBS-DC 在一些情况下耗时增加和平均运行时间增长速度稍慢的原因。ICBS-DC 为了能使高层约束树节点扩展的数量减少，在算法 2 中对产生的关键冲突再判断其是否属于本文提出的方向冲突，并在算法 1 中对冲突进行捕捉。当环境中产生的关键冲突较多时，判断方向冲突又在一定程度上增加了运行时间。图 11 和图 12 报告的结果符合 ICBS-DC 的主要目的：一方面减少了高层约束树节点扩展数量，使得约束树的规模减小；另一方面使得低层路径规划调用次数减少。因为算法高层每生成一个节点就会给相应智能体添加约束，调用低层路径规划算法对该智能体进行重新规划，所以图 11 和图 12 具有相似的曲线走势。当高层约束树中节点数量不是很多时，ICBS 算法的运行速度与 ICBS-DC 相当，甚至在一些情况下要快于 ICBS-DC；但当智能体数量较多时，使用本文所提的冲突消解方案可以有效对约束树的规模进行限制，从而减小高层算法的搜索时间，这与图 10 中报告的 ICBS-DC 的平均运行时间在智能体数量大于 12 时增长速度相比 ICBS 较为缓慢是一致的，说明 ICBS-DC 在智能体密集的环境下能展现出更好的求解速度优势。

## 4　结论

本文设计了一种基于冲突分类与消解的多智能体路径规划算法，以解决多智能体路径规划中的路径冲突问题。本文将多智能体路径冲突中的关键冲突进一步分类出相向顶点冲突和交叉顶点冲突，并针对这两类冲突类型提出相应的消解方案：1）相向顶点冲突的消解方案以添加两点约束的方式，避免了在消解该类冲突的过程中引发的新冲突，减少算法高层约束树节点扩展数量，减小约束树规模，降低算法运行时间；2）交叉顶点冲突的消解方案通过选择恰当的智能体等待时刻，在消解该类冲突的同时消解其他冲突，提高算法的搜索效率。最后通过仿真实验验证了本文提出的冲突分类与消解方案能够为多智能体路径规划问题求解出路径代价最优的解，并且能够有效减少算法高层约束树节点的数量以及低层路径规划算法的调用次数，进而加快算法的求解速度。

未来的研究工作包括以下两点：1）本文算法是基于四连通二维栅格地图设计的，未来考虑将本文算法扩展到三维空间的其他类型地图；2）在二维及三维环境中进行实物实验，验证本文算法的有效性。

## 参考文献

[1] Stern R, Sturtevant N, Felner A, et al. Multi-agent pathfinding: definitions, variants, and benchmarks [C]// Proceedings of 12th International Symposium on Combinatorial Search (SoCS), 2019: 151-158.

[2] Hönig W, Kiesel S, Tinka A, et al. Persistent and robust execution of MAPF schedules in warehouses [J]. IEEE Robotics and Automation Letters, 2019, 4 (2): 1125-1131.

[3] Dayan D, Solovey K, Pavone M, et al. Near-optimal multi-robot motion planning with finite sampling[C]// Proceedings of 2021 IEEE International Conference on

Robotics and Automation (ICRA). IEEE, 2021: 9190-9196.

[4] Choudhury S, Solovey K, Kochenderfer M J, et al. Efficient large-scale multi-drone delivery using transit networks[J]. Journal of Artificial Intelligence Research, 2021, 70: 757-788.

[5] 黄进，黄宗文，凌子燕．多智能体寻路系统在计算机游戏上的应用[J]．电脑知识与技术，2012，8(13)：3159-3164.
Huang Jin, Huang Zongwen, Ling Ziyan. Application of multi-agent pathfinding system in computer game[J]. Computer Knowledge and Technology, 2012, 8(13): 3159-3164(in Chinese).

[6] 刘庆周，吴锋．多智能体路径规划研究进展[J]．计算机工程，2020，46(4)：1-10.
Liu Qingzhou, Wu Feng. Research progress of multi-agent path planning[J]. Computer Engineering, 2020, 46(4): 1-10(in Chinese).

[7] Čáp M, Novák P, Kleiner A, et al. Prioritized planning algorithms for trajectory coordination of multiple mobile robots[J]. IEEE Transactions on Automation Science and Engineering, 2015, 12(3): 835-849.

[8] 夏绪辉，端木艳霞，郭钰瑶，等．立体仓库多机器人协同路径规划与冲突消解[J]．现代制造工程，2021，494(11)：49-57+16.
Xia Xuhui, Duanmu Yanxia, Guo Yuyao, et al. Multi-robot collaborative path planning and conflict resolution in three-dimensional warehouse[J]. Modern Manufacturing Engineering, 2021, 494(11): 49-57+16(in Chinese).

[9] Sabattini L, Digani V, Secchi C, et al. Optimized simultaneous conflict-free task assignment and path planning for multi-AGV systems[C]// Proceedings of 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017: 1083-1088.

[10] Digani V, Sabattini L, Secchi C. A probabilistic Eulerian traffic model for the coordination of multiple AGVs in automatic warehouses[J]. IEEE Robotics and Automation Letters, 2015, 1(1): 26-32.

[11] 张丹露，孙小勇，傅顺，等．智能仓库中的多机器人协同路径规划方法[J]．计算机集成制造系统，2018，24(2)：410-418.
Zhang Danlu, Sun Xiaoyong, Fu Shun, et al. Cooperative path planning in multi-robots for intelligent warehouse[J]. Computer Integrated Manufacturing Systems, 2018, 24(2): 410-418(in Chinese).

[12] Sharon G, Stern R, Felner A, et al. Conflict-based search for optimal multi-agent pathfinding[J]. Artificial Intelligence, 2015, 219: 40-66.

[13] Cohen L, Koenig S. Bounded suboptimal multi-agent path finding using highways[C]// Proceedings of 25th International Joint Conference on Artificial Intelligence (IJCAI-16), 2016: 3978-3979.

[14] Boyarski E, Felner A, Stern R, et al. ICBS: improved conflict-based search algorithm for multi-agent pathfinding[C]// Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI), 2015: 740-746.

[15] Felner A, Li J, Boyarski E, et al. Adding heuristics to conflict-based search for multi-agent path finding[C]// Proceedings of International Conference on Automated Planning and Scheduling, 2018, 28: 83-87.

[16] Li J, Harabor D, Stuckey P J, et al. Symmetry-breaking constraints for grid-based multi-agent path finding[C]// Proceedings of AAAI Conference on Artificial Intelligence, 2019, 33(1): 6087-6095.

[17] Semiz F, Polat F. Incremental multi-agent path finding[J]. Future Generation Computer Systems, 2021, 116: 220-233.

[18] 曹如月，张振乾，李世超，等．基于改进 A* 算法和 Bezier 曲线的多机协同全局路径规划[J]．农业机械学报，2021，52(S1)：548-554.
Cao Ruyue, Zhang Zhenqian, Li Shichao, et al. Multi-machine cooperation global path planning based on A-star algorithm and Bezier curve[J]. Transactions of the Chinese Society for Agricultural Machinery, 2021, 52(S1): 548-554(in Chinese).

[19] Song B, Wang Z, Zou L. An improved PSO algorithm for smooth path planning of mobile robots using continuous high-degree Bezier curve[J]. Applied Soft Computing, 2021, 100: 106960.

（编辑：李瑾）