计算机科学
COMPUTER SCIENCE

基于动态冲突预测的多智体寻路算法

张萌希, 韩建军, 肖彦

引用本文

张萌希, 韩建军, 肖彦. 基于动态冲突预测的多智体寻路算法[J]. 计算机科学, 2025, 52(4): 21-32.

ZHANG Mengxi, HAN Jianjun, XIAO Yan. Dynamic Conflict-Prediction Based Algorithm for Multi-agent Path Finding [J]. Computer Science, 2025, 52(4): 21-32.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241000101

基于动态冲突预测的多智体寻路算法

张萌希　韩建军　肖　彦
华中科技大学计算机科学与技术学院　武汉 430074
(m202273677@hust.edu.cn)

摘　要　多智体寻路(MAPF)是为多个智能体寻找无冲突路径的问题,灵活显式估计的基于冲突搜索算法是目前解决MAPF问题最有效的有界次优算法之一,但该算法仍存在调用底层算法次数多、迭代中冲突数量减少速度慢等问题。为此,提出基于动态冲突预测的多智体寻路算法(DCPB-MAPF)。该算法分为两层,在底层提出基于关键区间的动态避障方法与基于路径成本预测的迭代方法,用以提升底层算法的运算效率;以此为基础,在顶层提出基于冲突预测的搜索算法,通过快速预测冲突数量以优化冲突选择技术,进一步提出冲突数量优先的启发式函数以加速减少冲突数量。实验结果表明,相比现有算法,所提算法能显著提升多智体寻路问题的运算效率及成功率。

关键词:多智体路径寻找;有界次优算法;启发式搜索;路径成本预测;冲突预测
中图分类号　TP301

Dynamic Conflict-Prediction Based Algorithm for Multi-agent Path Finding

ZHANG Mengxi, HAN Jianjun and XIAO Yan
College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract　Multi-agent path finding(MAPF) is the problem of searching for collision-free paths for a group of agents. Currently, flexible explicit estimation conflict-based search algorithm(FEECBS) is deemed as one of the most effective bounded suboptimal algorithms to solve the MAPF problem, but it has several disadvantages concerning the frequent invocation of the low-level algorithm and slow reduction in the number of conflicts during iterations. To address such issues, This paper proposes a dynamic conflict-prediction based algorithm for multi-agent path finding(DCPB-MAPF). The DCPB-MAPF algorithm operates on a two-layer framework. On the low level, it investigates an optimized dynamic obstacle avoidance method based on critical intervals and a new iterative method based on path cost prediction for improving its efficiency. On the high level, it develops a conflict-prediction based search method together with the low-level algorithm. Firstly, the conflict selection technique is improved by quickly predicting the number of potential collisions. Further, a new method is proposed to accelerate reducing the number of conflicts by constructing a heuristic function. The extensively empirical experiment results demonstrate that DCPB-MAPF can effectively improve both efficiency and success rate on MAPF instances when compared to the existing algorithms.

Keywords　Multi-agent path finding, Bounded suboptimal algorithms, Heuristic search, Path cost prediction, Conflict prediction

1　引言

在当前的嵌入式多智体系统研究领域中,路径寻找问题是重要的研究方向之一。嵌入式系统由于具备集成度高、功耗低等优势,不仅可以满足系统的实时性要求,也在一定程度上使得系统的软件开发工作变得简单,因此在多智体寻路(Multi-Agent Path Finding, MAPF)领域有着广阔的应用前景。如今,基于嵌入式系统的多智体寻路算法已经被广泛应用于智能交通[1]、自动驾驶[2]、铁路运输[3]、无人机[4]等领域。

MAPF问题的关键约束是多个智能体同时沿着规划路径行进而不会发生冲突,其数学描述如下:

k个智能体的MAPF问题可被定义为一个元组(G,s,t),其中$G=(V,E)$是一个无向图,无向图中的节点$v\in V$是智能体可以占据的位置,边$(n,n')\in E$表示智能体的一次移动,每个智能体都有一个初始位置$s_i\in s$和一个目标位置$t_i\in t$。在MAPF问题中,时间被离散为时间步长。在每个时间步长中,每个智能体可以执行一个动作,一般有9种类型的动作:向上、向下、向左、向右、向左上、向右上、向左下、向右下和等待。一个单智能体的寻路是从起始位置到目标位置的一系列动作的集合$\pi=(a_1,a_2,\cdots,a_n)$,k个智能体的路径寻找就是k条路径的集合$\Pi=(\pi_1,\pi_2,\cdots,\pi_k)$,其中第$i$个智能体的路径对应$\pi_i$。称求得的所有智能体可行路径的集合为MAPF问题的解;当且仅当路径成本之和(即行程时间)最小时,称得到的MAPF问题的解为最优解。当路径成本之和在用户指定的有界次优区间之内时(即$C_1\leqslant(1+\omega)C$,其中ω为用户指定的有界次优参数),称得到的MAPF问题的解是有界次优解。

到稿日期:2024-10-21　返修日期:2025-02-24
通信作者:肖彦(yanx@hust.edu.cn)

目前 MAPF 问题的相关研究分为两类[5]：集中式规划算法及分布式规划算法。其中，集中式规划算法的运行速度与解的质量均优于分布式规则算法。集中式规划算法分为 4 种：基于 A* 搜索算法、基于冲突搜索算法（Conflict-Based Search，CBS）、代价增长树搜索算法和基于规约算法。其优缺点如表 1 所列。

表 1 集中式规划算法的优缺点

Table 1 Advantages and disadvantages of centralized planning algorithms

算法	优点	缺点
A* 搜索算法[6]	易于实现	空间代价高，求解速度慢
CBS 算法[7]	求解速度快，适用于高密度、大规模智能体环境	实现困难
代价增长树搜索算法[8]	实现简单，求解速度快	高密度智能体环境下易失效
规约算法[9]	求解速度快	规约证明困难

从表 1 中可以看出，CBS 算法是集中式规划算法中解决 MAPF 问题效率最高、适用范围最广的算法，但 CBS 及相关优化算法仍存在调用底层算法次数多以及迭代中冲突数量减少速度慢的问题。为解决上述问题，本文进行如下 3 方面的改进。

1）提出了一种优先考虑冲突数量的两层有界次优算法——动态冲突预测的多智体算法（Dynamic Conflict-Prediction Based Algorithm for Multi-Agent Path Finding，DCPB-MAPF），对 CBS 算法的顶层及底层算法进行双优化，以提升 MAPF 问题的运行效率。

2）提出动态有界次优 A* 算法（Dynamic Bounded Suboptimal A*，DBSA*）并将其应用于底层算法。该算法在 A* 算法基础上提出新的迭代方法及局部动态避障方法，进一步减少了顶层算法调用底层算法的频率。

3）在顶层算法提出新的冲突选择方法和新的启发式函数，提高冲突数量在智能体选择、主要冲突节点选择时的权重，加快迭代时冲突数量减少的速度，进而提升算法的运行效率。

2 相关工作

CBS 算法[7]是目前解决 MAPF 问题的最佳算法之一。该算法分为两层，底层算法为每个智能体独立规划一条路径，并在顶层解决智能体间的路径冲突。一旦在顶层发现路径冲突，则生成两个分支，每个分支为其中一个智能体添加冲突约束，以找到一条新的无冲突路径。目前针对 CBS 算法的优化方法非常多[10-19]，但由于该算法属于 NP 难问题[20]，因此在有限的计算资源下，MAPF 问题很难求出最优解。这一事实也推动了对该问题有界次优解的探索，以大幅提高算法的运行效率。

增强型基于冲突搜索算法（Enhanced Conflict- Based Search，ECBS）是 CBS 算法的有界次优变体[21]。该算法将底层与顶层的 A* 搜索算法替换为焦点搜索（Focal Search，FS）[22]来保证其有界次优性，进而减少算法运行时间。该算法及之后的有界次优算法均通过找到路径成本之和至多为 ω 倍路径下限的解来求得最终有界次优解。

显式估计的基于冲突的搜索算法（Explicit Estimation Conflict-Based Search，EECBS）[23]则针对 ECBS 启发式函数呈负相关时所导致的性能损耗，用显式估计搜索（Explicit Estimation Search，EES）[24]代替顶层的焦点搜索算法。EES 算法在最佳解决方案的路径成本总和上维护一个下限 LB，该下限由底层焦点搜索算法求出的最小路径成本下限得出。同时，该算法将 CBS 中的改良方法绕过冲突策略[10]、WDG 启发式函数[16]以及矩形推理[18-19]优化为适应有界次优算法的方法并加入到 EECBS 算法中，进一步减少了算法的运行时间。灵活显式估计的基于冲突的搜索算法（Flexible Explicit Estimation Conflict-Based Search，FEECBS）[25]在 EECBS 算法的基础上进一步放宽解集的要求以及底层算法的有界次优性要求，仅要求解集为有界次优，从而更灵活地分配各智能体的时间，同时改进绕过冲突策略并提出重启 EECBS 策略，以提升运行成功率。

贪婪的基于优先级的搜索算法（Greedy Priority-Based Search，GPBS）[26]则是另一种求解 MAPF 问题的有界次优变体。GPBS 算法通过为冲突代理分配优先级并在搜索过程中重新规划它们的路径来解决冲突；同时，针对高密度智能体与障碍物的 MAPF 问题实例，引入贪婪策略，通过最小化代理之间的碰撞次数来提升算法运行效率。而基于大邻域的快速修复多智体寻路算法（Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search，MAPF-LNS2）[27]则从一组包含冲突的路径开始，反复选择冲突智能体的子集并重新规划它们的路径以减少冲突的数量，直到找到无冲突的可行路径。与 GPBS 算法类似，MAPF-LNS2 算法也无法保证求得解的质量，但最大限度地提升了算法的运行效率及成功率。

除针对有界次优解的探索外，也有对智能体路径冲突规划与动态避障能力的进一步研究。Jin 等基于 D* Lite 算法[28]提出了 CBS-D*[29]算法，以解决智能体面对环境动态变化的路径重规划问题。该算法以曼哈顿距离为启发式函数，将增量搜索范围从 1 邻域扩展到 3 邻域，同时提出了避免碰撞的预判断机制以进一步提升动态避障能力。此外，该算法提出了一种机器人行走过程中的等待和迂回策略，大幅提升了寻路成功率，减少了避碰次数及时间步长。

当前的领先有界次优算法存在以下两个问题：

1）底层算法均需寻找从起点到终点的可行路径，考虑到顶层算法生成新节点时均需调用底层算法，导致遍历节点数较多，本文第 4 章提出动态有界次优 A* 算法。

2）顶层算法中缺少对冲突数量下降速度的加速方法，导致出现顶层约束树层数多、算法运行效率低的问题。本文第 5 章针对此问题，提出新的冲突选择方法和新的启发式函数。

3 符号说明及 DCPB-MAPF 算法框架

本章对本文出现的符号进行简要说明并给出 DCPB-MAPF 算法的整体框架。

3.1 符号说明

表 2 列出了本文使用的所有符号及其描述。

表 2 符号及其描述

Table 2 Symbols and its description

符号	描述
ω	有界次优参数
$g(v)$	底层中智能体a_i从起点到节点v的时间步
$c(v,v')$	底层节点v与节点v'间的路径成本
va_i	底层算法中，约束增加且在当前智能体路径上的关键节点
vr_i	底层算法中，约束减少且可能导致当前智能体路径改变的关键节点
v_s	当前智能体的路径起点
v_{goal}	当前智能体的路径终点
n_w	底层节点1邻域内的障碍物数量
$d(v)$	底层节点v与路径起点的距离
$\vec{d}_1(v)$	底层节点v与焦点搜索算法得出路径的距离
$\vec{d}_2(v)$	底层节点v与当前路径的距离
$\vec{d}_3(v)$	$\vec{d}_3(v_r)=\vec{d}_2(v)-\vec{d}_1(v)$
S_b	新增关键约束区间前的减少约束关键节点所组成的集合
S_a	新增关键约束区间后的减少约束关键节点所组成的集合
v_p'	新增约束关键节点v'的前继节点
v_s'	新增约束关键节点v'的后继节点
C_i	智能体a_i路径成本
S_i	智能体a_i预测路径成本
R_i	智能体a_i综合预测路径成本
$f_{\min}$	智能体a_i最低路径成本
$NumNode_1$	第一次得到可行路径需遍历节点数
$NumNode_2$	第一次迭代遍历节点数
$h_c(N)$	顶层节点N的冲突数量
$pri(N)$	顶层节点N的优先级
C_{con}	顶层节点代价
$C_{con,1}$	主要冲突节点代价
$C_{con,2}$	次要冲突节点代价
$C_{con,3}$	不重要冲突节点代价
$\Delta h_{c,j}(n)$	顶层不重要节点j冲突数量的增长值
$g(N)$	$\sum_{i=1}^{k} f_{\min,i}(N)$
$h(N)$	顶层所有节点的最小路径代价与$g(N)$的差

3.2 算法整体框架

为进一步优化 MAPF 问题的相关算法，在 EECBS 算法的基础上对其顶层及底层算法进行双优化并提出 DCPB-MAPF 算法（如图 1 所示）。

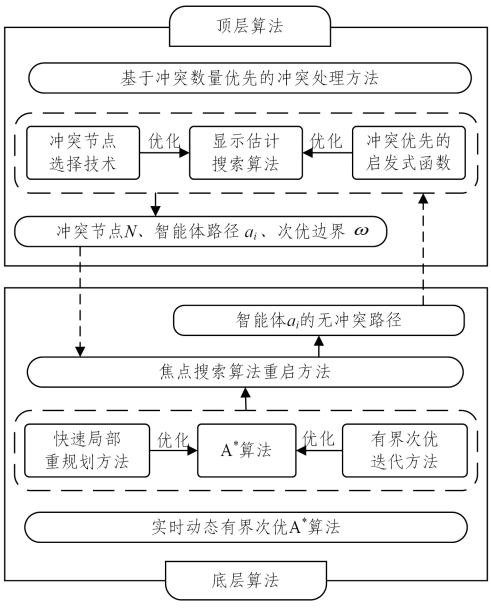

图 1 DCPB-MAPF 算法示意图

Fig. 1 Schematic diagram of DCPB-MAPF

优化思路如下：

1）在底层算法中，为避免重复计算起点到终点的路径而只计算受影响的局部路径，提出基于动态约束关键区间的快速局部重规划方法（见 4.1 节）；又考虑到此方法会影响算法的有界次优性，提出基于路径成本预测的有界次优迭代搜索方法（见 4.2 节）以确保算法的有界次优性；最后，加入焦点搜索算法的重启判断方法（见 4.3 节），以解决多次重规划导致的寻路失败问题。

2）在顶层算法中，提出优化冲突选择技术（见 5.1 节）以优化节点扩展方式，使预测性能好的节点被优先选择；同时，提出新的冲突数量优先的启发式函数（见 5.2 节），筛选冲突数量下降快的节点优先扩展，以提升算法的运行速度。

4 底层算法——动态有界次优A* 算法

本章具体介绍图 1 中底层算法的优化方法。底层算法基于一条可行路径，针对智能体约束的动态变化及有界次优参数的改变进行以下 3 步优化。

步骤 1 基于动态约束关键区间实现快速局部重规划方法。由于本文算法与 D* Lite[28] 以及 CBS-D*[29] 等算法所处理的情况不同，现有算法主要用于解决动态约束实时变化的问题，而本文面临的动态约束变化为已知情况，因此本步骤将优先识别由关键节点所组成的关键区间，并基于关键区间进行整体的局部调整，以提升算法运行速度。

步骤 2 基于有界次优参数迭代获得更优路径。本步骤将优化现有的迭代方法，在兼顾递减性的基础上进一步提升算法迭代的收敛速度。

步骤 3 重启焦点搜索方法。本步骤将详细阐述优化步骤 1 及步骤 2 可能导致的问题，并针对性地提出重启焦点搜索方法以减少可能带来的额外时间损耗。

4.1 基于动态约束关键区间的快速局部重规划方法

在 MAPF 问题中，约束的改变分为两种情况：第一种是新增约束，当顶层算法为其他智能体重规划路径时，会导致当前智能体无法通过部分节点，称此情况为新增约束；第二种是减少约束，当顶层算法为其他智能体重规划路径，且经过若干时间步后新增约束消失，当前智能体可以重新通过该部分节点，称为减少约束。因此，基于两种约束定义关键节点及关键区间类型。

定义 1（新增约束关键节点） 满足以下两个条件的为新增约束关键节点va_i。

条件 1 该节点为顶层智能体路径重规划导致的约束增加。

条件 2 该节点在当前智能体的可行路径上。

由这两个条件可知，新增约束关键节点均为一定会导致当前可行路径发生改变的节点。接下来基于新增约束关键节点定义新增约束关键区间。

定义 2（新增约束关键区间） 所有新增约束关键节点组成的集合$[va_1,\cdots,va_n]$称为新增约束关键区间。

与定义 1 对应，定义 3 对减少约束关键节点进行定义。

定义 3（减少约束关键节点） 满足以下 4 个条件的为减少约束关键节点vr_i。

条件 1 该节点为顶层智能体路径重规划导致的约束减少。

条件 2 该节点 1 邻域内的障碍物数量n_w满足 $2\leqslant n_w\leqslant 6$。

条件 3 该节点需满足条件 $d(vr_i)\notin[d(va_1),d(va_k)]$，其中 $d(vr_i)$为该节点与起点v_s的距离，$d(va_i)$为新增约束关键节点va_i与起点的距离，$[d(va_1),d(va_k)]$为新增约束关键区间与起点的距离范围。

条件 4 计算该节点与初始智能体路径的距离$\vec{d}_1(v_r)$、该节点与当前智能体路径的距离$\vec{d}_2(v_r)$及两者距离差$\vec{d}_3(v_r)$，减少约束关键节点需满足$|\vec{d}_3(v_r)|>|\vec{d}_2(v_r)|$。

接下来分别说明定义 3 中 4 个条件的必要性。定义 3 的条件 1 与定义 1 的条件 1 类似，是关键约束节点判断的基本条件。条件 2 则保证减少约束关键节点为“墙”的缺口，图 2(a)和图 2(c)均展示了减少约束关键节点。如图 2(b)所示，节点的 1 邻域内只出现 1 个障碍物，该情况下可能存在可优化空间，但由于其在 4.2 节的迭代搜索方法中将被首先优化，因此此处不考虑该类节点，避免部分节点被重复遍历。如图 2(d)所示，节点的 1 邻域内出现 7 个障碍物，意味着节点周围只有一个出口，因此最终求得的可行路径不可能通过该节点。条件 3 保证了减少约束关键节点不在新增约束关键区间中，避免节点被重复计算(如图 2(e)所示，减少约束节点在新增约束关键区间中，故不是减少约束关键节点)。此条件的制定受两种关键区间的处理策略影响，后文的策略 1 中将详细解释原因。条件 4 则要求减少约束关键节点需在可行路径附近且通过该节点的新路径成本可能减少，即要求新的可行路径有概率通过该节点(如图 2(f)所示即违反了该条件)。

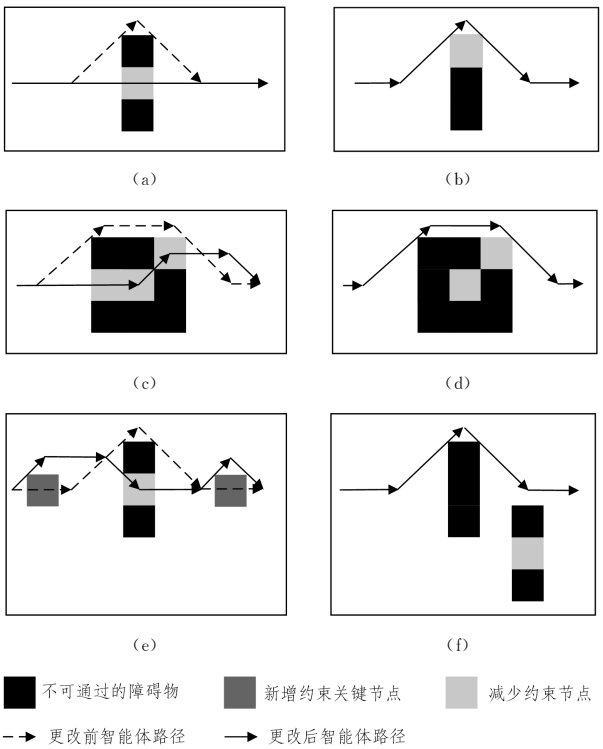

图 2 减少约束关键节点判断示意图

Fig. 2 Judge of constraint-reduction key nodes

综上，定义 1 和定义 3 的条件 1 均为各自的基本条件，定义 1 的条件 2 与定义 3 的条件 2 和条件 4 均用于筛选掉不会影响路径的节点，定义 3 的条件 3 则用于避免节点被重复计算。其中，定义 3 的条件 3 也导致减少约束关键区间与新增约束关键区间的定义有所不同，减少约束关键区间由新增约束关键区间前后两个小区间组成。称新增约束关键区间前的区间为S_b，其后的区间为S_a，有如下定义：

定义 4(减少约束关键区间) 减少约束关键节点组成的集合$[S_b,0,S_a]$称为减少约束关键区间，其中$S_b=[vr_1,\cdots,vr_i]$，$S_a=[vr_{i+1},\cdots,vr_m]$。$S_b$中节点均为$va_1$前继节点之前的节点，$S_a$中节点均为$va_n$后继节点之后的节点，$S_b$和$S_a$均可为空集。

在完成关键区间的定义后，引入 $RHS(v)$以及 $G(v)$两个列表存储单智能体路径搜索时节点 v 到终点v_{goal}的路径成本，方便后续阐述区间处理策略。补充说明两个列表特点，$RHS(v)$为实时计算得出的值，具有实时性；而 $G(v)$记录之前的节点数据，根据 $RHS(v)$的值进行更新。$RHS(v)$的计算式如式(1)所示，$G(v)$则由 $RHS(v)$在合适时机赋值得到。

$$RHS(v)=\begin{cases}0, & v \text{ is } v_{\text{goal}}\\ \min\limits_{v'\in s(v)}(g(v')+c(v,v')), & \text{else}\end{cases} \tag{1}$$

接下来具体阐述两种关键区间的处理策略。

策略 1 新增约束关键区间处理。在参照定义 1 寻找新增约束关键节点时，逐步更新所寻找节点的 RHS 值。当寻找到新增约束节点va_i时，更新其前继节点v_p'、后继节点v_s'与该节点va_i间的成本，令 $c(v',v_p')=c(v',v_s')=+\infty$。在确定新增约束关键区间$[va_1,\cdots,va_n]$后，以最靠近可行路径起点的新增约束关键节点$va_1$的前继节点$v_p'$为起点，以最靠近可行路径终点的新增约束关键节点$va_n$的后继节点$v_s'$为终点，运行双向 A* 算法，并更新已搜索节点$v_s'$前所有节点的 G 值，完成新增约束关键区间$[va_1,\cdots,va_n]$的路径搜索。此时，由于是在两节点间直接运行双向 A* 算法，区间内所有减少约束节点均已被考虑在内，因此在定义 3 中增加条件 3 以避免节点被重复遍历。

策略 2 减少约束关键区间处理。参照定义 4，减少约束关键区间分为S_b和S_a两个区间，对两个区间分别进行处理，寻找区间内可行路径，由于两区间处理策略一致，因此不重复说明。减少约束通常出现在某父节点增加约束后，经过时间步 t，智能体移动，该约束消失。因此，找到靠近起点的减少约束关键节点vr_1，迭代搜索vr_1，直至找到一个节点v_l在当前路径上。根据 $RHS(v_l)$更新节点vr_1前继节点的 RHS 值，使用双向 A* 算法依次寻找减少约束关键节点的路径 $sol(vr_1,vr_2)$，…，$sol(vr_{i-1},vr_i)$并更新对应节点的 RHS 值。接着，迭代搜索vr_i的后继节点，直至找到一个节点v_r在当前路径上，根据 $RHS(v_i)$更新节点vr_i后继节点的 RHS 值，直至更新完成 $RHS(v_r)$。此时，若满足式(2)，则更新路径，为减少约束区间内所有新路径节点的 G 赋值；若不满足，则维持原路径。

$$RHS(v_r)+t<G(v_r) \tag{2}$$

在具体阐述完 4 种定义及 2 种约束关键区间的处理策略后，补充策略 3 以描述其实现顺序及原因。

策略 3 优先处理新增约束关键区间(策略 1)，再处理减

少约束关键区间(策略 2)。

由于寻找新增约束关键区间仅需遍历可行路径节点(参照定义 1 的条件 2)，遍历节点数较少且在遍历过程中会直接得到区间内的最佳路径(参照策略 1)，而寻找减少约束关键区间需首先寻找新增约束关键节点(参照定义 3 的条件 3)，因此首先处理新增约束关键区间的相关问题，再根据定义 3 和定义 4 确定减少约束关键区间并根据策略 2 得到减少约束关键区间内的可行路径。

4.2 基于路径成本预测的有界次优迭代方法

底层算法除需处理约束的动态改变外，还需满足算法整体的有界次优性要求，因此根据当前的可行路径及有界次优参数进行路径迭代，以获得问题更优解并保证算法的有界次优性。以下给出已有算法中 3 种主要的迭代方法[30]。

迭代方法 1　依据有界次优参数，以$\omega_1>\cdots>\omega_k=1$为序列进行迭代，其中第 i 次迭代的有界次优参数为ω_i。

迭代方法 2　依据路径成本 C，以$C_1=\infty>\cdots>C_k$为序列进行迭代，其中第 i 次迭代的路径成本为C_i。

迭代方法 3　依据预测路径成本S_i，令成本$C_i=S_{i-1}-\varepsilon$，其中 ε 为一小正数，则$\omega_i=\dfrac{S_{i-1}-\varepsilon}{f_{\min}}$，以$C_1=\infty>\cdots>C_k$为序列进行迭代。

迭代方法 1 的算法收敛速度快；迭代方法 2 具备严格递减性，能更快求得最优解；迭代方法 3 基于迭代方法 2 提出，部分兼顾了迭代方法 1 和 2 的优点，但其成本依赖于上一次迭代，在部分情况下使得算法的收敛速度慢于迭代方法 1，因此结合迭代方法 1 和 3 提出新的迭代方法 4。

新迭代方法 4　提出并以新预测路径成本R_i作为依据进行迭代。令$R_i=\min(S_{i-1}-\varepsilon,\omega_i\cdot f_{\min})$，$\omega_i=\min\left(\omega_i,\dfrac{R_i}{f_{\min}}\right)$，以$R_1=+\infty>\cdots>R_k$为序列进行迭代。

图 3 用一种情况比较说明了 4 种迭代行为。图 3(a)为迭代方法 1 路径成本的下降示意图，在第 2 次迭代时其严格递减性无法保证；迭代方法 2(如图 3(b)所示)的收敛速度较慢，相较于迭代方法 1，第一次迭代收敛速度大幅降低；迭代方法 3(如图 3(c)所示)能保证迭代的递减性，相较于迭代方法 2，提升了算法收敛速度，但第一次迭代的收敛速度相较于迭代方法 1 仍有明显下降，因此有了新迭代方法 4(见图 3(d)所示)，其相较于迭代方法 3，进一步提升了收敛速度。

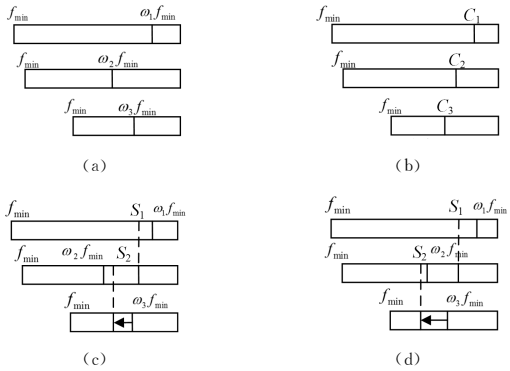

图 3　4 种迭代方法示意图

Fig. 3　Schematic diagram of 4 iterative methods

4.3 焦点搜索重启方法

上述方法在实际测试时存在两个问题：

问题 1　MAPF 问题中的约束改变较为频繁，而动态约束的多次改变有一定概率导致寻找到路径成本较大的解。

问题 2　次优边界的迭代优化在迭代次数少时表现较好，但当迭代次数较多时，时间效率会大幅降低。

问题 1 为优化步骤 1 的相关问题，问题 2 为优化步骤 2 迭代过程中产生的问题。

问题 1 的出现主要是由于 4.1 节中基于关键约束区间的动态优化方法会提前筛选关键节点，这样虽然避免了节点被重复遍历，使算法在运行效率上有较大提升，但提前筛去部分关键节点会导致路径成本增加，进而使解的最优性下降。图 4 展示了其中一种路径成本增加的情况，图中的减少约束节点不符合 4.1 节中定义 3 减少约束关键节点判断的条件 4($|\vec{d}_3(v_r)|=\sqrt{5}<|\vec{d}_2(v_r)|=\sqrt{10}$)，因而不是减少约束关键节点。但该情况下，通过该节点的理论最优路径成本为 $6+4\sqrt{2}$；而若按照本文算法，则路径成本不变，为 $4+6\sqrt{2}$，高于理论最优路径成本。因此，本节点在算法中被提前筛掉，就会导致智能体路径成本下降速度变慢的问题，在后续节点的计算中将导致节点被迫需要满足更严苛的有界次优区间，增加算法运行时间，故提出重启焦点搜索算法的条件 2，在迭代中避免严苛的有界次优区间。

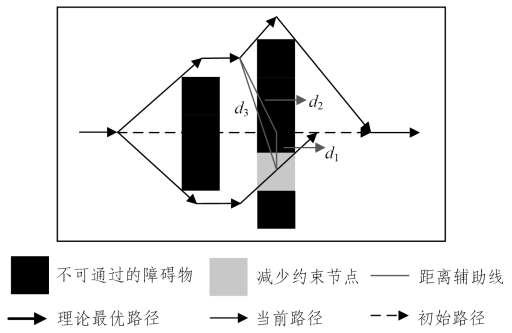

图 4　路径成本增加示意图

Fig. 4　Schematic diagram of increase of path costs

问题 2 的出现是因为底层算法迭代次数过多时，部分节点被重复遍历，进而造成额外的时间开销。表 3 列出了在一张 50 * 30 的空白地图上运行 FS 算法及本文算法的遍历节点数。

表 3　有界次优参数变动时 FS 算法与 DSBA* 算法的遍历节点数

Table 3　Traversed nodes' number of FS and DBSA* when bounded suboptimal parameter changes

有界次优参数	焦点搜索算法遍历节点数	DBSA* 算法遍历节点数
2.4	323	446
2.1	366	93
1.8	432	127
1.5	478	87
1.2	494	109

在该地图中分别运行焦点搜索及动态有界次优 A* 算法。当需直接得到一条有界次优参数分别为 2.4，2.1，1.8，1.5，1.2 的可行路径时，本文底层算法需遍历的节点数分别

为 446,539,666,753,862 个,均大于对应有界次优参数下焦点搜索算法所需遍历的节点数。因此提出重启焦点搜索算法的条件 1,采用焦点搜索算法得到初始的可行路径。在迭代时,当有界次优参数由 2.4 调整至 2.1 时,若采用之前的次优边界调整方法则需运行两次焦点搜索算法,共遍历 689 个节点,而本文动态有界次优 A* 算法仅需遍历 539 个节点,此时本文底层算法优于焦点搜索算法。但当有界次优参数由 2.4 变为 1.2 时,使用焦点搜索方法仅需遍历 817 个节点,而本文算法需遍历 862 个节点,此时焦点搜索算法优于本文算法。

该情况也表明该算法更适用于有界次优参数变动较小、迭代次数较少的场景,不适用于直接寻找有界次优参数较小的可行路径或迭代时有界次优参数变动较大的场景。

基于上述两点问题,通过加入重启焦点搜索方法(Focal Search Restart Method,FSR),修正底层算法。当顶层算法选中当前智能体时,将此时的有界次优参数ω_1及目标有界次优参数ω_k传至底层。此时底层算法首先进行一次迭代,得到有界次优参数ω_2及此次迭代额外遍历的节点数$NumNode_2$,当满足如下条件时,重启焦点搜索算法。

重启条件 1 运行该算法的冲突节点 N 中没有记录智能体a_i的可行路径,则启用焦点搜索算法并记录当前智能体的探索节点数$NumNode_1$。

重启条件 2 进行一次迭代,若此时有界次优区间满足$\left\lceil\frac{\omega_1-\omega_2}{\omega_1-\omega_k}\right\rceil \cdot NumNode_2 > NumNode_1$,则重启焦点搜索。

重启条件 1 主要针对顶层算法中的根节点,由于根节点的智能体没有可行路径,因此采用焦点搜索算法代替原算法以解决问题 2 中直接寻找路径遍历节点数多的问题。尤其在 MAPF 问题中,根节点的有界次优参数一般设定较小,因此使用焦点搜索算法也可以有效避免多次迭代,进而减少算法的运行时间;条件 2 则用于解决问题 1 提前筛选关键节点所导致的有界次优参数要求严格及问题 2 有界次优参数变化大所导致的迭代次数多的问题。先进行一次迭代,基于初始焦点搜索算法遍历节点数量以及节点 N 处的单次迭代节点数量估计本文底层算法的时间效率,若预估时间效率不好,则证明可能出现问题 1 与问题 2,导致需求的有界次优参数过小,进而导致迭代遍历时间过长,因此转为使用传统焦点搜索算法。

底层动态有界次优 A* 算法的伪代码如算法 1 所示。

算法 1 底层动态有界次优 A* 算法

输入:冲突节点 N,智能体a_i的起点与终点,对该路径要求的有界次优界限 ω

输出:智能体a_i的可行路径

```
1. if Path(a_i) is not in N then/* 重启焦点搜索条件 1 */
2.     FS(N,ω);
3. end if
/* 新增约束关键区间处理 */
4. if FindAddConstraint(N) then/* 新增约束关键节点 */
5.     UpdateC(v);/* 更新节点间路径成本 */
6.     BiAStar(v_p1', v_sk');/* 双向 A* 算法 */
7.     UpadateG(v);
8. end if
/* 减少约束关键区间处理 */
9. if FindReduceConstraint(N) then
10.    for v in ALSEARCH and GetIndex(v) > GetIndex(vr_k) do
       /* 找到 vr_k与可行路径间的路径 */
11.        UpdateRHS(v);
12.        CalD(vr_k);
13.    end for
14.    for v_i in [vr_k,…,vr_j+1] do
15.        BiAstar(v_i, v_i+1);
16.    end for
17.    Findpath(vr_j+1);/* 重复上述减少约束关键区间处理方法得到另一段关键区间的路径 */
18.    if RHS(v_r)+t<G(v_r)
19.        UpdateRoad();/* 若新路径成本更低,更新路径 */
20.    end if
21. end if
/* 重启焦点搜索方法条件 2 */
22. if ⌈(ω1−ω2)/(ω1−ωk)⌉ · NumNode2 > NumNode1 then
23.    FS(N,ω);
24. else
25.    NewARA(R);/* 依据迭代方法 4 多次迭代 */
26. end if
27. return Path(a_i)
```

算法 1 分为 4 步,第一步(第 1—3 行)依据重启条件 1 判断是否重启焦点搜索(见 4.3 节),第二步(第 4—8 行)处理新增约束关键区间(见 4.1 节策略 1),第三步(第 9—21 行)处理减少约束关键区间(见 4.1 节策略 2),第四步(第 22—27 行)依据重启条件 2 判断是否重启焦点搜索(见 4.3 节),若不符合则依照 4.2 节新迭代方法 4 更新路径。

4.4 底层算法复杂度分析

本节将分析底层算法的复杂度,并从理论角度分析本文优化方法相较于现有算法的优势。底层算法包括重启焦点搜索算法及 4.1 节和 4.2 节中所述的优化方法。

焦点搜索算法为 A* 算法的变体,也是当前领先的有界次优算法的底层算法,其算法复杂度为 $O(b^l)$,其中 $b(1\leqslant b\leqslant 8)$为每个节点的分支数,$l$ 为探索路径长度,该值大于可行路径长度l_1。在最差情况下,该算法需要遍历整个搜索空间,复杂度为指数级,但在实际计算中,该算法通常能在较短时间内搜索到可行路径。相较于 A* 算法,焦点搜索算法允许搜索到次优解,进而提升扩展子节点的成功率(A* 算法中部分节点的扩展子节点能找到可行路径,但不是最优路径也会被认为是不成功的,需要回溯),进而使分支数 b 降低,运行速度提升。

对于 4.1 节中所述优化方法,关键区间的确定复杂度为 $O(l)$,关键区间内的寻路问题复杂度为 $O(b^{l'})$,其中 l'为关键区间宽度,其值远小于探索路径长度 l。对于大部分约束变化较小的情况或是仅有一个约束发生变化时,l'为 1,该部分时间复杂度近似为常数。对于 4.2 节中所述优化方法,其

复杂度近似为 $O(k*b^{l_2})$。其中 k 为迭代次数，在有界次优界限小于 3 时，该值较小，可默认为常数；l_2 为迭代时沿着可行路径遍历的节点数，在满足预测路径成本下降的预期值后，就可中断迭代，因此 $l_2<l_1$。综上所述，4.1 节和 4.2 节中所述优化方法的综时间复杂度为 $O(b^{l'}+k*b^{l_2})$，而 l' 与 l_2 均小于 l，因此相较于焦点搜索算法（时间复杂度为 $O(b^l)$），本文底层优化算法的运算时间将大幅降低。

4.5 底层算法对比实验

本节通过对比实验，验证底层算法改进方法的有效性。

1）比较算法：顶层算法均采用 EECBS 算法的顶层算法，底层算法分别采用经典的焦点搜索算法、本文动态有界次优 A* 算法（无焦点搜索重启方法）与动态有界次优 A* 算法（有焦点搜索重启方法）。通过对比 3 种算法来验证焦点搜索重启方法的有效性。

2）参数设置：地图参数及对比实验的智能体数量如表 4 所列，每张地图中的智能体起点与终点都为均匀分布，算法测试时有界次优设置为 1.2。

表 4 地图参数的选取

Table 4 Selection of maps parameters

地图	大小	可走空格数量	最大智能体数量
paris_1_256	256 * 256	47 240	1 100
maze-32-32-4	32 * 32	790	210
empty-16-16	16 * 16	256	128

3）性能指标：算法运行速度与算法运行成功率。

实验结果如表 5、表 6 所列。相较于经典的焦点搜索算法，本文算法在 3 张地图上均能显著提升算法运行速度，略微提升算法成功率，尤其在地图 maze-128-128-2 上，本文算法在运行时间上能减少 54.7%，成功率提升 4.7%。

表 5 3 种算法的运行时间

Table 5 Runtime of three algorithms

(s)

算法	paris		maze		empty	
	600 智能体	1000 智能体	60 智能体	100 智能体	90 智能体	100 智能体
FS	4.7	13.7	0.3	4.2	0.60	22.4
DBSA*	**4.4**	14.1	0.4	60.0	**0.14**	60.0
DBSA* (FSR)	4.6	**12.3**	**0.3**	**3.5**	0.20	**21.3**

表 6 3 种算法的运行成功率

Table 6 Success rate of three algorithms

(%)

算法	paris		maze		empty	
	600 智能体	1 000 智能体	60 智能体	100 智能体	90 智能体	100 智能体
FS	100.0	83.7	100	95.3	100	46.6
DBSA*	91.4	60.3	100	74.8	100	14.7
DBSA* (FSR)	100.0	**94.3**	100.0	**100**	100	**49.2**

从表 5 与表 6 还可以得出，焦点搜索重启方法能更好地适应智能体密度高的复杂环境。在 3 张地图的高密度智能体环境下，FSR 算法的运行成功率显著提升，提升幅度最高可达 34.5%。

5 顶层算法优化——基于冲突数量优先的显式估计搜索算法

图 5 为 DCPB-MAPF 顶层算法的流程图，其中淡灰色区域为图 1 中顶层算法的优化。一方面，提出基于冲突数量预测的节点选择技术，为冲突节点制定新分类方法以提升节点选择的合理性；另一方面，提出冲突数量优先的启发式函数以加大冲突数量在冲突节点选择时的比重，进而加速冲突数量下降。

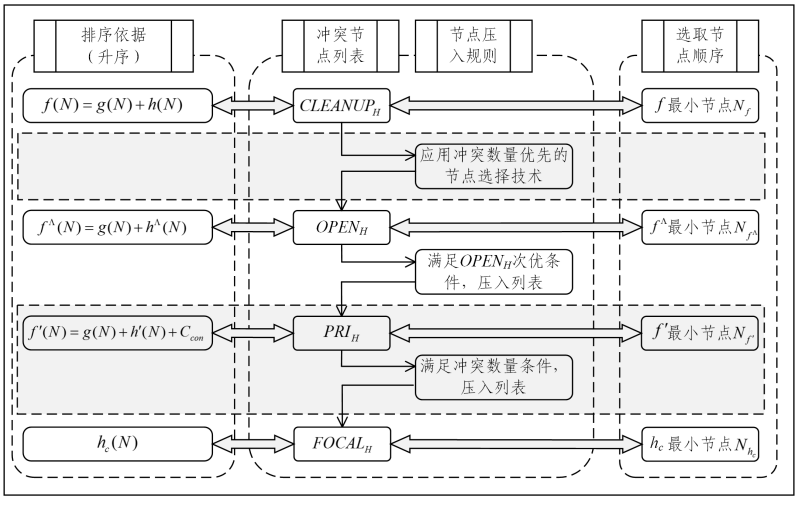

图 5 顶层算法流程

Fig. 5 Flow of the high-level algorithm

5.1 冲突数量优先的节点选择技术

现有的优先冲突选择技术（Prioritizing Conflicts Technique，PC）将冲突分为主要冲突、半主要冲突以及其他冲突，其分类标准主要为有界次优条件下冲突节点的成本下限。当 $\sum_{i=1}^{m} f_{\min,i}(N)<\sum_{i=1}^{m} f_{\min,i}(N')$ 时，认为子节点 N' 的成本下限减少。因此节点分类如下：当 N 的子节点 N' 与 N'' 均满足成本下限减少时，称 N 为主要冲突节点；当子节点 N' 与 N'' 之一满足成本下限减少时，称 N 为半主要冲突节点。节点扩展优先级依次为主要冲突节点、半主要冲突节点、其他冲突节点。

而本文提出的算法则进一步提出了基于冲突数量优先的节点分类方法，将底层算法重启带来的额外时间开销及可能的冲突数量增长考虑在内，具体如下：

节点类型1(主要冲突节点) 被选中的节点 N 生成子节点 N' 以及 N''，N' 若满足 $\sum_{i=1}^{m} f_{\min,i}(N) < \sum_{i=1}^{m} f_{\min,i}(N')$，同时满足 $\left\lceil \frac{\omega_1-\omega_2}{\omega_1-\omega_k} \right\rceil \cdot NumNode_2 \leqslant NumNode_1$，则认为 N' 节点为主要冲突节点。对于子节点 N''，同理。

节点类型2(次要冲突节点) 子节点 N' 若满足 $\sum_{i=1}^{m} f_{\min,i}(N) < \sum_{i=1}^{m} f_{\min,i}(N')$，同时满足 $h_c(N') < h_c(N)$，则认为 N' 为次要冲突节点。

节点类型3(不重要冲突节点) 两种情况外的冲突节点。

基于以上冲突节点类型，为节点的扩展制定优先级。首先，依据父节点的优先级依次顺序遍历。父节点的优先级为 $pri(N)=pri(N')+pri(N'')$。其中子节点优先级参照冲突节点分类制定，主要冲突节点优先级为2，次要冲突节点为1，其余为0。然后，根据子节点的冲突类型，按如下规则扩展。

规则1 当子节点 N' 以及 N'' 均为主要冲突节点或均为次要冲突节点时，对 N' 以及 N'' 均进行扩展操作。

规则2 当子节点 N' 以及 N'' 均为不重要节点时，随机选择其中一个进行扩展操作。

规则3 当其中一个子节点优先级高于另一个子节点时，仅对优先级高的子节点进行扩展操作。

需要注意的是，未扩展节点均为回溯时再扩展。

相较于过去的节点扩展方式，本文算法的扩展规则将冲突数量提前考虑在内，在节点选取前就做出判断，以加快冲突数量减少速度；同时将节点分类细化到子节点，避免部分冲突数量下降慢、成本下限下降慢的节点被优先选中。

图6(a)展示了现有算法顶层节点扩展方式。相较于本文的节点扩展方式(如图6(b)所示)，现有算法会导致部分预测性能不好的节点被优先选中，例如节点D的预测性能不好，但在旧扩展方式中会先于节点G被扩展，这可能会导致算法的运行效率降低。

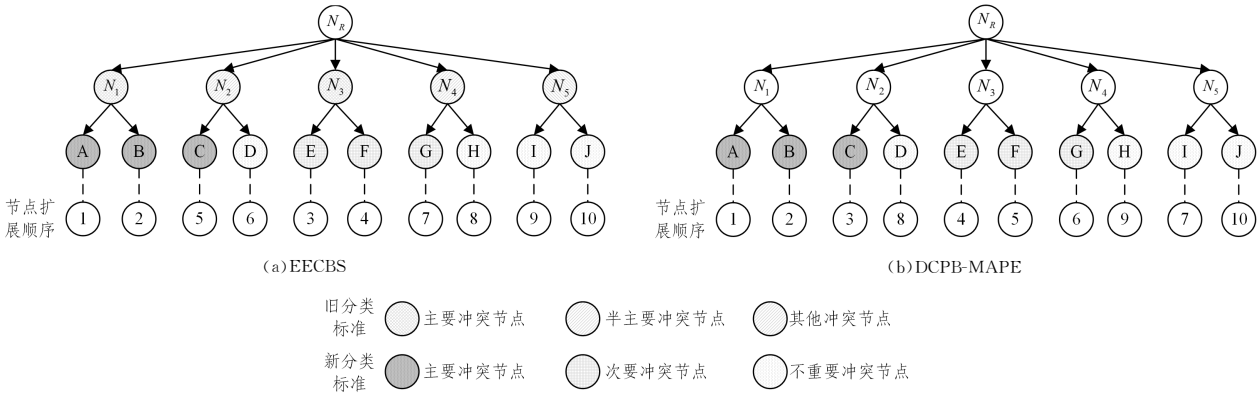

图6 EECBS和DCPB-MAPF节点扩展方式比较

Fig. 6 Comparison of node expansion method between EECBS and DCPB-MAPF

5.2 冲突数量优先的启发式函数优化

传统的搜索算法以 $f(n)=g(n)+h(n)$ 作为搜索依据。在EECBS中，引入在线学习方法，新增代价函数 $h^{\wedge}(n)$ 作为启发式函数，但其仍然主要基于最小下界的增长来衡量子节点的优劣。因此本文算法在 $OPEN_H$ 与 $FOCAL_H$ 之间多维护 $PRIC_H$ 列表，筛选冲突节点数量较少的智能体。

基于上述冲突节点分类标准，设计新的启发式函数 $f'(N)$，如式(3)－式(5)所示：

$$f'(N)=g(N)+h(N)+C_{con} \tag{3}$$

$$C_{con}=\frac{1}{h_c(N)}\sum_{i=1}^{3}C_{con,i} \tag{4}$$

$$C_{con,i}=\begin{cases} h_{c1}(N), & i=1 \\ h_{c2}(N), & i=2 \\ \sum_{j=1}^{h_{c3}(N)}\Delta h_{c,j}(n), & i=3 \end{cases} \tag{5}$$

式(3)为顶层算法的启发式函数表达式。其中，$g(N)$ 为已处理的冲突节点数，$h(N)$ 为仍需处理的节点数量(为预估值，对于性能好的节点，该值会快速下降)；C_{con} 为本文新增的冲突数量因子，用于表示每个扩展节点的预测性能(预测性能越好的节点，其冲突数量下降速度越快，算法运行时间越短，该因子也就越小)。参数的具体含义为：$C_{con,1}$ 代表主要冲突节点代价 $C_{con,2}$ 代表次要冲突节点代价，$C_{con,3}$ 代表不重要冲突节点代价；$h_{c1}(N)$，$h_{c2}(N)$，$h_{c3}(N)$ 定义与之类似，分别为对应冲突类型的节点数量；$\Delta h_{c,j}(n)$ 则代表对应不重要节点 j 冲突数量的增长值。

$PRIC_H$ 列表根据 $f'(N)$ 升序排序，同时会将满足条件 $f'(N)\leqslant(1+\omega)f'_{\min}$ 的所有节点压入 $FOCAL_H$ 列表。

相较于EES[24]算法中的启发式函数，该函数进一步增加了冲突数量在顶层算法中的权重，提前筛选出部分冲突数量下降迅速的节点，进一步减少需扩展的节点数量。

5.3 顶层算法时间复杂度分析

本节将分析顶层算法的复杂度，并从理论角度分析本文优化方法相较于现有算法的优势。

顶层算法的时间复杂度为 $O(k^{\max\left(\frac{t}{v_t},\frac{LB}{v_{lb}}\right)})$，其中 k 为约束树每一层扩展的子节点，t 为冲突数量，v_t 为冲突数量的下降速度，LB 为预期路径成本下限(与预设的有界次优参数相关)，v_{lb} 为路径成本下限的上升速度。k 值与冲突节点的选择、扩展策略有关。5.1节的优化冲突节点选择技术基于预测冲突数量下降速度、路径成本下限提升速度以及底层是否重启焦点搜索算法3个条件重新制定节点扩展策略，以减少每层节点扩展数量 k，同时加速冲突数量的下降速度 v_t 与路径

成本下限的增长速度v_{lb}，减少时间复杂度中的指数项大小。5.2节的优化方法则以冲突数量的下降速度为标准提出新的启发式函数，进一步加速冲突数量的下降速度，进而减少时间复杂度中的指数项，提升算法的运行效率。

5.4 顶层算法对比实验

本节通过对比实验验证顶层算法改进方法的有效性。

1）比较算法：底层算法为本文DBSA*算法，顶层算法分别采用EES算法、本文提出的基于冲突数量优先的显式估计搜索算法（Prioritizing Conflict Based EES，PCBEES）。通过对两种算法进行比较，验证本文算法对顶层冲突数量下降的加速作用。

2）参数设置：同4.5节。

3）性能指标：顶层节点扩展的节点数（扩展节点数越少，冲突数量下降速度越快，调用底层算法次数越少）。

由表7可知，PCBEES算法需扩展的节点数均小于现有的EES算法，尤其在高智能体密度环境下，其能大幅提升顶层算法冲突数量的下降速度。

表7 两种算法的扩展节点数

Table 7 Number of expanded nodes of two algorithms

算法	paris		maze		empty	
	600智能体	1000智能体	60智能体	100智能体	90智能体	100智能体
EES	1	2	17	99	88	2164
PCBEES	1	**1**	**14**	**57**	**61**	**1260**

6 实验评估

本章通过基准测试样例上的大量实验，验证本文算法在解的质量、算法运行时间、算法成功率等方面的优越性。

6.1 实验设置

6.1.1 实验平台

算法用C++实现，实验在Ubuntu 22.04.3 LTS Linux操作系统、内存为8GB的11th Gen Intel® Core™ i5-11400H CPU上运行。

6.1.2 基准算法

将目前领先的3种有界次优算法EECBS，FEECBS，GPBS与本文DCPB-MAPF算法进行比较。

1）EECBS[23]：采用EES算法优化ECBS顶层算法，并引入多种CBS优化策略提升算法效率。

2）FEECBS[25]：灵活分配各智能体的有界次优区间。

3）GPBS[26]：基于PBS算法引入贪婪策略以最小化碰撞次数，并引入CBS优化策略提升算法效率。

其中，EECBS，FEECBS，DCPB-MAPF算法为同类型算法，可以设置有界次优参数以控制解的质量；而GPBS算法则无法提前设定有界次优参数，因此难以控制最终解的路径成本。

6.1.3 基准测试样例

本文在MAPF基准测试样例[31]中选取6个不同大小和结构的地图（前2张为大地图，后4张为小地图），每个地图和场景共50个测试实例（25个智能体的起点与终点位置为随机分布的测试样例，25个智能体的起点与终点位置为平均分布的测试样例），地图参数如表8所列。

表8 选取地图的参数

Table 8 Parameters of selected maps

地图	大小	可走空格数量	最大智能体数量
warehouse-20-40-10-2-1	321×123	22599	1000
berlin_1_256	256×256	47540	1000
empty-32-32	32×32	1024	512
random-32-32-20	32×32	819	409
room-32-32-4	32×32	682	341
maze-32-32-2	32×32	666	333

6.2 解的质量

本节采用路径成本之和作为解的质量评判指标。由于GPBS算法在架构设计上缺乏有界次优参数的调控机制，因此在本节实验中，EECBS，FEECBS与DCPB-MAPF算法的有界次优参数的选择将依据算法运行时间的限制决定，即在满足10s内解决MAPF问题的条件下，选择最小的有界次优参数值。

接下来分析有界次优参数对解的质量及算法运行时间的影响，并介绍有界次优参数的选择方法。此实验以地图“empty-32-32”为例进行测试，实验中的智能体数量为280个，实验结果如表9所列。

表9 有界次优参数对运行时间及解的影响

Table 9 Effects of bounded suboptimal parameters on runtime and solution optimality

算法	指标	1.05	1.1	1.15	1.2	1.25
EECBS	运行时间	—	—	0.48	0.33	0.31
	解的质量	—	—	7508	7701	7726
FEECBS	运行时间	—	3.53	0.24	0.23	0.23
	解的质量	—	7104	7345	7444	7423
DCPB-MAPF	运行时间	—	1.82	0.26	0.21	0.19
	解的质量	—	6990	7323	7336	7325

注：—表示算法无法在10s内得到结果。

由表9可得，当有界次优参数从1.05逐步提升至1.25时，3类算法展现出显著的性能变化规律：算法的运行时间逐渐减少，解的质量逐渐下降（路径成本之和逐渐增加）。依据表9选取运行时间小于10s的最小有界次优参数值，当前情况下，EECBS算法的有界次优参数值设定为1.15，FEECBS和DCPB-MAPF算法的设定为1.1。

明确有界次优参数的选取方式之后，比较4种算法解的质量，实验结果如图7所示。本文算法DCPB-MAPF与FEECBS算法在6张地图上均表现出良好性能，略优于EECBS算法，大幅优于GPBS算法，尤其在地图“empty-32-32”与“random-32-32-20”中，DCPB-MAPF算法相比于GPBS算法，求得的路径总成本减少10%左右。而在地图“maze-32-32-2”中，DCPB-MAPF算法与GPBS算法求得的解的质量相近。结合6张地图的结构特点，DCPB-MAPF算法在相对空旷的地图中求得的路径成本较低，而在Maze地图中，最终解的路线相对接近，更多是需要处理局部的冲突，此时DCPB-MAPF算法的性能表现与FEECBS和GPBS算法接近。

综上，DCPB-MAPF算法在解的质量上显著优于GPBS算法，且由于GPBS算法无法灵活设置有界次优值，因此难以在相同的有界次优区间内衡量其与另外3种算法的运算效率，故后续实验将比较3种同类型算法DCPB-MAPF，FEECBS，EECBS的运算效率。

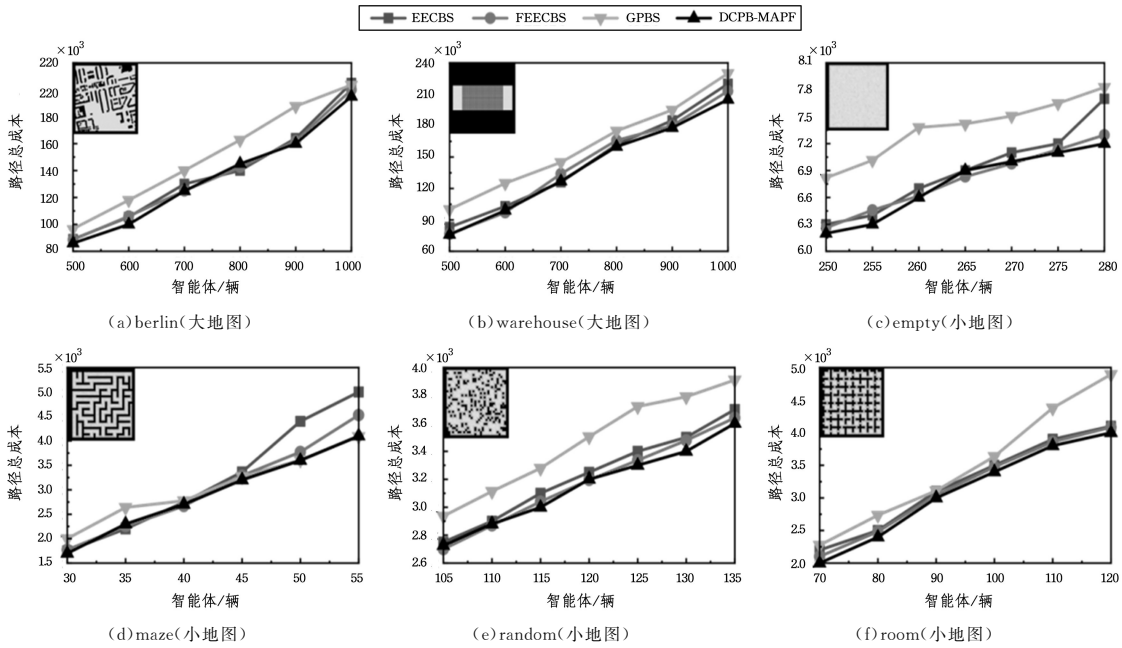

图 7 在 6 张地图上 4 种算法解的质量

Fig. 7 Solution optimality of four algorithms in six maps

6.3 算法运行时间

图 8 展示了 6 张地图上 3 种算法的运行时间，3 种算法的有界次优参数值为 1.2。在大地图“berlin_1_256”和“warehouse-20-40-10-2-1”上，本文算法与 FEECBS 算法的运行速度相近；而在小地图上，本文算法相较于 FEECBS 和 EECBS 算法，运行速度有明显提高。在有限的时间内，DCPB-MAPF 算法能解决更多智能体的寻路问题，其在地图“room-32-32-4”中效果最好，可解决的智能体规模扩大了 21%。

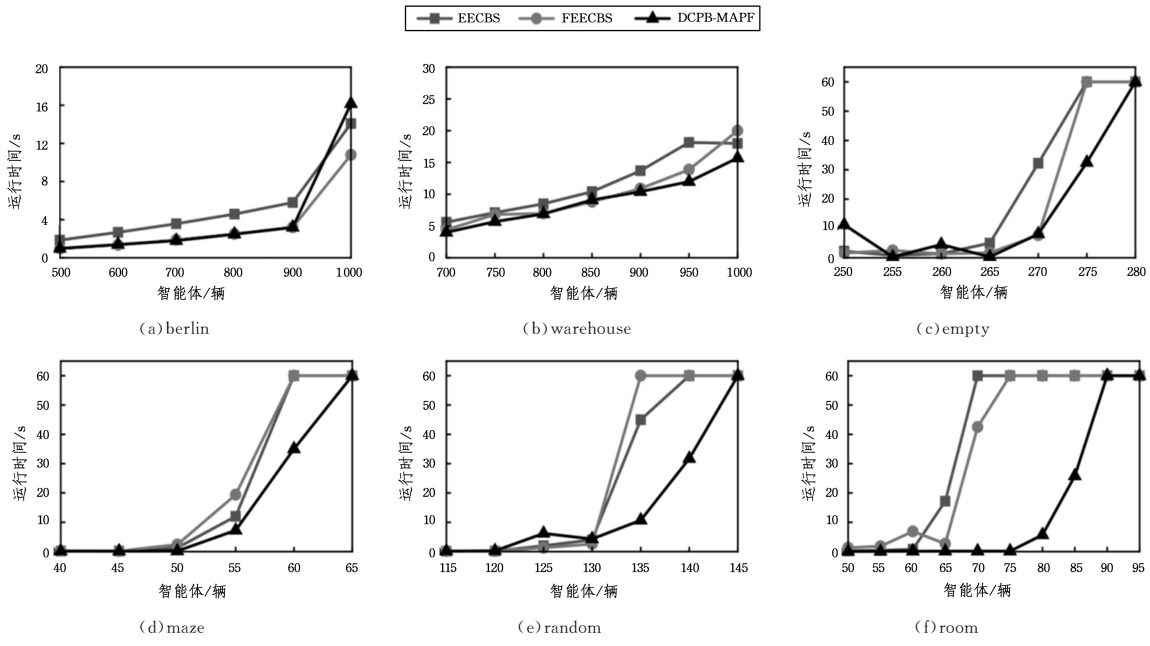

图 8 DCPB-MAPF，FEECBS，EECBS 算法在 6 张地图上的运行时间

Fig. 8 Runtime of DCPB-MAPF, FEECBS and EECBS in six maps

从表 8 中可知，大地图相对于小地图更为空旷，冲突数量相对较少，因此 DCPB-MAPF 算法基于冲突数量优先的优化方法在大地图上效果不明显，在小地图上则有显著提升。而小地图“room-32-32-4”中 DCPB-MAPF 算法的优化效果最明显，主要原因在于其特殊的结构，其由多个小房间组成且每个小房间均有多个“门”通往其余房间，这也导致处于同一房间的智能体在“门”处会发生多次冲突，且冲突节点在重规划路线时，路线不会出现大规模变动，因此 DCPB-MAPF 算法的底层优化方法可以显著减少算法运行时间。

6.4 算法运行成功率

本节测试算法运行成功率，实验有界次优参数值设定为 1.2。如图 9 所示，在大地图上，FEECBS 算法与本文 DCPB-MAPF 算法的运行成功率相近；而在小地图上，本文算法相较于 FEECBS 及 EECBS 算法在运行成功率上有明显提高。

表 10 列出了 4 张小地图上 3 种不同成功率下 DCPB-MAPF 算法相较于 FEECBS 算法优化的幅度。在“room-32-32-4”与“maze-32-32-2”中，DCPB-MAPF 算法优化幅度最大。而在“empty-32-32”中，DCPB-MAPF 则与 FEECBS 算法能解决的智能体规模基本相同。结合表 8 所列的地图参数以及地图的具体结构，“room”与“maze”可走空格数量最小（智能体密度相对较大）且均有大量狭长通道，这会导致智能体在寻路过程中出现大量冲突，且冲突智能体将普遍选择等待策略而无法通过绕路的策略获得更优解，从而使 DCPB-MAPF 算法的顶层、底层优化均能取得良好效果。

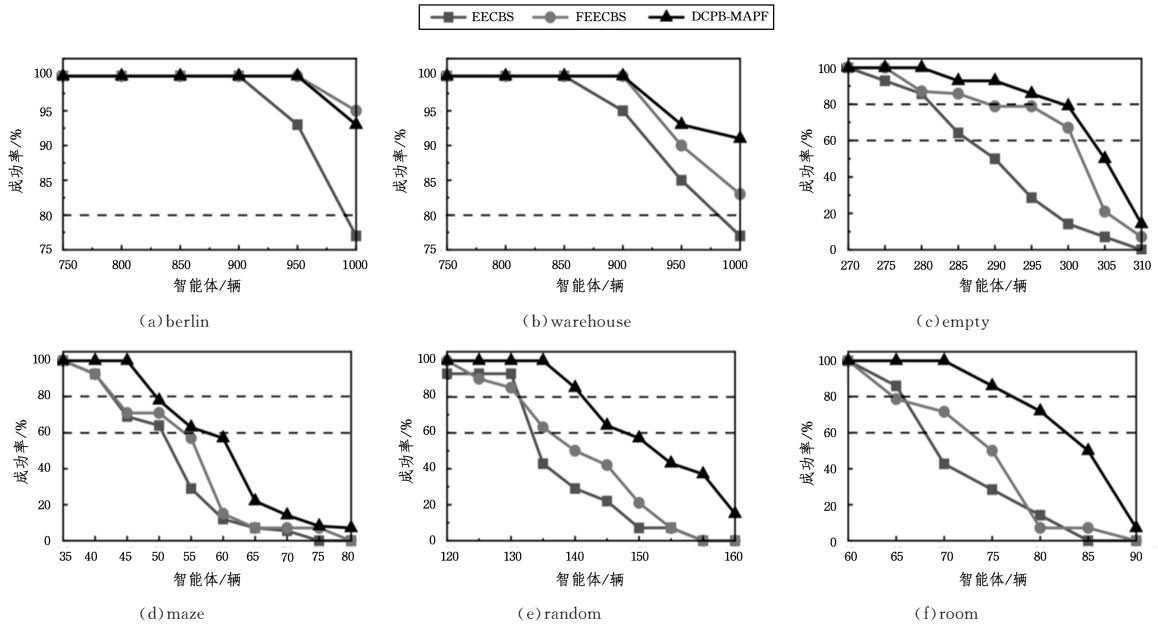

图 9 DCPB-MAPF，FEECBS，EECBS 算法在 6 张地图上的算法运行成功率

Fig. 9 Success rate of DCPB-MAPF，FEECBS and EECBS in six maps

表 10 DCPB-MAPF 和 FEECBS 可处理智能体数量的增长幅度的比较

Table 10 Compare of growth rate of agents handled by DCPB-MAPF and FEECBS

（%）

地图	成功率 100%智能体增长幅度	成功率 80%智能体增长幅度	成功率 60%智能体增长幅度
Room	14.3	18.1	13.7
Empty	1.8	2.5	1.1
Maze	28.6	13.9	5.5
Random	12.5	7.5	8.8

综合上述实验结果，在智能体密度高、有大量狭长通道的地图上，DCPB-MAPF 算法运行速度快，成功率高。

结束语 本文提出了一种新的有界次优 MAPF 算法 DCPB-MAPF，在算法底层提出了动态有界次优 A* 算法，根据环境的动态变化情况以及有界次优参数迭代计算单个智能体的路径；在算法顶层优化了冲突优先选择技术，提出了基于冲突数量优先的启发式函数以加速冲突数量的减少。通过这些改进，DCPB-MAPF 算法在智能体密度高、存在大量狭长通道的地图上的运算性能相较于目前领先的有界次优算法 FEECBS 有了显著提高。

未来的工作包括以下 3 方面：

1）为底层算法制定更细致的重启方法；

2）将更多 CBS 算法相关的研究方法优化为冲突优先的相关方法，并应用到 DCPB-MAPF 算法中，以进一步加快冲突数量减少的速度；

3）将 DCPB-MAPF 算法与实际的应用场景相结合，如路口交通路线规划等。

参 考 文 献

[1] BAO X Y. Application practice of general cooperative vehicle infrastructure in the field of transportation[J]. Information and Communications Technology and Policy，2024，50(3)：53-59.

[3] LI J，LIN E，VU H L，et al. Intersection coordination with priority-based search for autonomous vehicles[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park，CA：AAAI，2023：11578-11585.

[4] LI J，CHEN Z，ZHENG Y，et al. Scalable rail planning and replanning：Winning the 2020 flatland challenge[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park，CA：AAAI，2021：477-485.

[5] HU D，GAN V J L，WANG T，et al. Multi-agent robotic system (MARS) for UAV-UGV path planning and automatic sensory data collection in cluttered environments[J]. Building and Environment，2022，221：109349.

[6] LIU Z F，CAO L，LAI J，et al. Overview of multi-agent path finding[J]. Computer Engineering and Applications，2022，58(20)：43-62.

[7] FERGUSON D，LIKHACHEV M，STENTZ A. A guide to heuristic-based path planning[C]// Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems，International Conference on Automated Planning and Scheduling. Menlo Park，CA：AAAI，2005：9-18.

[8] SHARONG，STERN R，FELNER A，et al. Conflict-based search for optimal multi-agent pathfinding[J]. Artificial Intelligence，

2015,219:40-66.
[9] WALKER T T,STURTEVANT N R,FELNER A,et al. Conflict-based increasing cost search[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2021:385-395.
[10] BARTAK R,ZHOU N F,STERN R,et al. Modeling and solving the multi-agent pathfinding problem in picat[C]// 2017 IEEE 29th International Conference on Tools with Artificial Intelligence. Piscataway,NJ:IEEE,2017:959-966.
[11] BOYARSKIE,FELNER A,SHARON G,et al. Don't split,try to work it out:Bypassing conflicts in multi-agent pathfinding [C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2015:47-51.
[12] BOYARSKIE,FELNER A,STERN R,et al. Icbs:The improved conflict-based search algorithm for multi-agent pathfinding [C]// Proceedings of the International Symposium on Combinatorial Search. New York:ACM,2015:223-225.
[13] BOYARSKIE,HARABOR D,STUCKEY P,et al. F-cardinal conflicts in conflict-based search[C]// Proceedings of the International Symposium on Combinatorial Search. New York:ACM,2020:123-124.
[14] BOYARSKIE,FELNER A,HARABOR D,et al. Iterative-deepening conflict-based search[C]// Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. San Francisco,CA:Morgan Kaufmann,2021:4084-4090.
[15] FELNERA,LI J,BOYARSKI E,et al. Adding heuristics to conflict-based search for multi-agent path finding[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2018:83-87.
[16] GANGEG,HARABOR D,STUCKEY P J. Lazy CBS:implicit conflict-based search using lazy clause generation[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2019:155-162.
[17] LI J,FELNER A,BOYARSKI E,et al. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search[C]// Proceedings of the International Conference on International Joint Conferences on Artificial Intelligence. San Francisco,CA:Morgan Kaufmann,2019:442-449.
[18] LI J,HARABOR D,STUCKEY P J,et al. Disjoint splitting for multi-agent path finding with conflict-based search[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2019:279-283.
[19] LI J,HARABOR D,STUCKEY P J,et al. Symmetry-breaking constraints for grid-based multi-agent path finding[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2019:6087-6095.
[20] LI J,GANGE G,HARABOR D,et al. New techniques for pairwise symmetry breaking in multi-agent path finding[C]// Proceedings of the International Conference on Automated Planning and Scheduling. Menlo Park,CA:AAAI,2020:193-201.
[21] YU J,LAVALLE S. Structure and intractability of optimal multi-robot path planning on graphs[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2013:1443-1449.
[23] BARER M,SHARON G,STERN R,et al. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem[C]// Proceedings of the International Symposium on Combinatorial Search. New York:ACM,2014:19-27.
[23] PEARL J,KIM J H. Studies in semi-admissible heuristics[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,1982(4):392-399.
[24] LI J,RUML W,KOENIG S. EECBS:A bounded-suboptimal search for multi-agent path finding[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2021:12353-12362.
[25] THAYER J T,RUML W. Bounded suboptimal search:A direct approach using inadmissible estimates[C]// Proceedings of the International Conference on International Joint Conferences on Artificial Intelligence. San Francisco,CA:Morgan Kaufmann,2011:674-679.
[26] CHAN S H,LI J,GANGE G,et al. Flex Distribution for Bounded Suboptimal Multi-Agent Path Finding[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2022:9313-9322.
[27] CHAN S H,STERN R,FELNER A,et al. Greedy priority-based search for suboptimal multi-agent path finding[C]// Proceedings of the International Symposium on Combinatorial Search. New York:ACM,2023:11-19.
[28] LI J,CHEN Z,HARABOR D,et al. MAPF-LNS2:Fast repairing for multi-agent path finding via large neighborhood search[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2022:10256-10265.
[29] KOENIG S,LIKHACHEV M. D* lite [C]// Eighteenth National Conference on Artificial Intelligence. Menlo Park,CA:AAAI,2002:476-483.
[30] JIN J Z,ZHANG Y,ZHOU Z P,et al. Conflict-based search with D* lite algorithm for robot path planning in unknown dynamic environments[J]. Computers and Electrical Engineering,2023,105:108473.
[31] LIKHACHEV M,GORDON G J,THRUN S. ARA*:Anytime A* with provable bounds on sub-optimality[C]// Advances in Neural Information Processing Systems. San Francisco,CA:Morgan Kaufmann,2003.
[32] STERN R,STURTEVANT N,FELNER A,et al. Multi- agent pathfinding:Definitions,variants,and benchmarks[C]// Proceedings of the International Symposium on Combinatorial Search. New York:ACM,2019:151-158.

ZHANG Mengxi, born in 2000, postgraduate, is a member of CCF(No. V1352G). His main research interests include AI algorithm and multi-agent path finding.

XIAO Yan, born in 1995, Ph.D. Her main research interests include AI algorithm, privacy preserving, and information security technology.

(责任编辑:何杨)