软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(8):3655−3676 [doi: 10.13328/j.cnki.jos.007234] [CSTR: 32375.14.jos.007234]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

局部搜索算法求解最小弱连通支配集问题*

李睿智[1,3], 何锦涛[1], 欧阳丹彤[2]

[1](吉林财经大学 管理科学与信息工程学院, 吉林 长春 130117)

[2](吉林大学 计算机科学与技术学院, 吉林 长春 130012)

[3](吉林省商务大数据研究中心, 吉林 长春 130117)

通信作者: 欧阳丹彤, E-mail: ouyd@jlu.edu.cn

摘　要: 最小弱连通支配集问题是一个经典的 NP 难问题, 在许多领域都有广泛的应用. 提出一种高效的局部搜索算法求解该问题. 在该算法中, 首先采用一个基于锁定顶点和频率反馈信息的初始解构造方法. 该方法可以确保将一定处于最优解中的顶点和大概率存在于最优解中的顶点添加到初始解中, 从而可以得到高质量的初始解. 其次, 提出基于双层格局检测策略, 年龄属性和禁忌策略的方法来避免循环问题. 第三, 提出扰动策略, 使得算法能够有效跳出局部最优. 第四, 将两个评分函数 *Dscore* 和 *Nscore* 与避免循环问题的策略相结合, 提出有效的顶点选择方法, 帮助算法选择适合添加到候选解中或从当前候选解中删除的顶点. 最后, 与现有的最优启发式算法和 CPELX 求解器, 在 4 组基准测试实例上对提出的局部搜索算法进行了对比. 实验结果表明, 该算法在 4 组经典基准测试实例上表现出更好的性能.

关键词: 最小弱连通支配集问题; 组合优化; 局部搜索; 反馈机制; 扰动策略; 年龄属性

中图法分类号: TP301

中文引用格式: 李睿智, 何锦涛, 欧阳丹彤. 局部搜索算法求解最小弱连通支配集问题. 软件学报, 2025, 36(8): 3655–3676. http://www.jos.org.cn/1000-9825/7234.htm

英文引用格式: Li RZ, He JT, Ouyang DT. Local Search Algorithm for Minimum Weakly Connected Dominating Set Problem. Ruan Jian Xue Bao/Journal of Software, 2025, 36(8): 3655–3676 (in Chinese). http://www.jos.org.cn/1000-9825/7234.htm

Local Search Algorithm for Minimum Weakly Connected Dominating Set Problem

LI Rui-Zhi[1,3], HE Jin-Tao[1], OUYANG Dan-Tong[2]

[1](School of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China)

[2](College of Computer Science and Technology, Jilin University, Changchun 130012, China)

[3](Jilin Province Business Big Data Research Center, Changchun 130117, China)

Abstract: The minimum weakly connected dominating set problem is a classic NP-hard problem that has wide applications in various fields. This study proposes an efficient local search algorithm to solve this problem. The algorithm employs a method to construct an initial solution based on locked vertices and frequency feedback. This method ensures the inclusion of vertices that are certain or highly likely to be in the optimal solution, resulting in a high-quality initial solution. Furthermore, the study introduces a method to avoid cycling based on two-hop configuration checking, age properties, and tabu strategies. A perturbation strategy is also proposed to enable the algorithm to effectively escape from the local optimum. Additionally, effective vertex selection methods are presented to assist the algorithm in choosing vertices suitable for addition to or removal from the candidate solution by combining two scoring functions, *Dscore* and *Nscore*, with strategies for avoiding cycling. Finally, the proposed local search algorithm is evaluated on four benchmark test instances and compared with four state-of-the-art algorithms and the CPELX solver. Experimental results demonstrate that the proposed algorithm achieves better performance.

* 基金项目: 吉林省科技厅自然科学基金 (YDZJ202201ZYTS413); 吉林省教育厅重点基金 (JJKH20240201KJ)
收稿时间: 2024-03-21; 修改时间: 2024-05-02; 采用时间: 2024-06-05; jos 在线出版时间: 2024-09-30
CNKI 网络首发时间: 2024-10-08

Key words: minimum weakly connected dominating set problem; combinatorial optimization; local search; feedback mechanism; perturbation strategy; age property

1 引 言

给定一个无向连通图 $G(V, E)$, 其中 V 表示图 G 的顶点集, E 表示图 G 的边集. 若存在一个顶点子集 $D \subseteq V$, 使得 V 中的每一个顶点都在 D 中或者至少与 D 中的一个顶点相邻, 那么称集合 D 为支配集 (dominating set, DS). 若集合 D 是一个支配集, 并且该集合的诱导子图是连通的, 那么称集合 D 为连通支配集 (connected dominating set, CDS), 其中 D 的诱导子图定义为 $IS_G(D) = (D, E \cap (D \times D))$. 若集合 D 是一个支配集, 且 D 的弱诱导子图是连通的, 那么称集合 D 为弱连通支配集 (weakly connected dominating set, WCDS), 其中 D 的弱诱导子图定义为 $WIS_G(D) = (N[D], E \cap (D \times N[D]))$, 其中 $N[D]$ 包含集合 D 及 D 中顶点的邻居. 最小弱连通支配集问题 (minimum weakly connected dominating set problem, MWCDSP) 是在图 G 上找到一个具有最小顶点规模的弱连通支配集.

最小弱连通支配集问题是一个经典的 NP 难问题, 最早是由 Dunbar 等人[1]提出, 在分布式传感器网络[2], 离散时间线性 2 次调节[3], 计算最优网络安全投资问题[4], 电力系统安全[5]和自然语言处理[6]等多个领域得到广泛应用. 在一个无向网络中, 节点之间可以通过全向天线进行通信. 然而, 由于每个节点的通信半径有限, 因此需要中继节点来实现信息传播, 从而引入了最小连通支配集问题和最小弱连通支配集问题的研究. 然而, 弱连通支配集在集群形成[7]中具有显著的潜力, 可用于减少网络中的集群头数量, 有效解决了分布式传感器网络[2]中关键的安全集群问题. 同时, 寻找一个最小弱连通支配集意味着确定一组最少数量的节点作为数据转发的中继点, 这样所有节点都能够通过这些中继节点实现信息的全网覆盖, 即使并非所有节点都直接相连. 这样的解决方案有助于减少所需的通信资源, 降低能耗, 同时保证网络的整体连通性和稳定性. 此外, 除了一般图的 NP 难度计算挑战之外, 许多研究还专注于特定类型图[8−10]的最小弱连通支配集问题计算复杂性. 这些研究进一步表明了研究最小弱连通支配集问题的理论意义, 且该问题更为灵活且具有挑战性, 因此本文对该问题进行了深入的研究.

目前, 已有许多学者对最小弱连通支配集问题进行了研究并设计了多种求解算法, 这些算法主要分为两类, 一类是近似算法, 一类是启发式算法. 对于近似算法, Chen 等人[11]提出了一种分布式逼近算法, 用于在无线自组网中进行聚类. 该算法通过寻找接近最优解的最小弱连通支配集问题的解来实现. Devdatt 等人[12]提出了一种在分布式网络中具有良好的“低拉伸”特性的快速分布式算法. Ding 等人[13]提出了一种针对最小弱连通支配集的线性时间自稳定算法 (MWCDS), 该算法可以快速构造最小弱连通支配集. Torkestani 等人[14]首先给出了随机图中弱连通支配集问题的定义, 接着提出了几种基于自动学习的算法, 用于解决顶点相关权重的概率分布函数未知的随机最小弱连通支配集问题. 但对于通常包含许多局部最优解的复杂图, 仍难以应用于求解最小弱连通支配集问题. 然而, 近似算法并不能保证解的质量, 而且通常得到的解与最优解有很大的差异. 因此, 一些学者致力于启发式搜索算法的研究, 该类算法能够在合理的时间内给出更高质量的解.

Niu 等人[15]提出了一种针对最小弱连通支配集问题的自稳定模因算法 (MA). 该算法首先构造若干个初始解, 然后随机选择两个解, 将其作为一对染色体序列, 然后对该对染色体序列进行交叉和突变操作, 接着将得到的新的染色体序列通过简单的局部搜索来对其进一步优化. 随后, 针对该问题, Niu 等人[16]提出了一种贪婪的随机自适应搜索算法 (GRASP). 该算法采用平衡贪婪和随机策略求出初始解, 然后通过简单的局部搜索对该初始解进行优化. 此外, Niu 等人[16]对 GRASP 进行了改进, 提出了 GRASP_impLS 算法, 以获得一个更小的弱连通支配集. 在局部搜索阶段, 引入了一个新的贪婪函数和一个参数. 贪婪函数增强了顶点选择的多样性, 参数控制使用顶点去除方法的概率. 这些改进显著地提高了 GRASP_impLS 算法解决该问题的效率.

然而, 许多学者更倾向于针对最小支配集问题或其变体设计启发式算法[17,18]和精确算法[19,20], 而关于最小弱连通支配集问题的启发式算法研究仍处于起步阶段. 尽管对于小规模实例, 目前求解效果最好的启发式算法 (GRASP_impLS) 可以得到较好的解, 但对于大规模和复杂的实例来说, 仍有很大的改进空间. 其中一个可能的原因是, 在寻找最优解的过程中, 很难保证不会出现同一顶点被多次选择的情况, 导致当前候选解陷入局部最优. 因

此, 需要提出更加有效的启发式策略来求解最小弱连通支配集问题. 为此, 本文设计了一种更有效的启发式算法FPLS (local search algorithm based on the feedback mechanism and the perturbation strategy) 来寻找最小弱连通支配集问题的次优解或最优解. 在该算法框架中, 提出了基于锁定顶点和候选解中顶点反馈信息的初始解构造过程, 并提出了带有年龄属性、双层格局检测策略、禁忌策略和扰动策略的局部搜索过程.

首先, 本文提出了图中锁定顶点的定义. 锁定顶点可被证明是最优解的一部分, 通过在算法初期就明确锁定这些必选顶点, 可以直接缩小后续搜索的顶点集合, 使得算法能更集中资源在剩余顶点的优化选择上, 从而提高搜索效率. 因此, 将锁定顶点添加到候选解中可以有效地减少搜索空间, 并且在拥有锁定顶点的基础上, 其他策略如反馈机制、年龄属性、双层格局检测策略和禁忌策略等可以更专注于优化剩余顶点的选择, 无需再考虑这些已锁定顶点, 简化了策略的复杂度. 在贪心构造初始解过程中, 优先将锁定顶点的加入解中, 然后根据顶点的分数和局部搜索过程中所反馈的信息, 优先考虑那些历史上频繁出现在最优解中的顶点, 这直接提升了初始解的质量, 为后续的局部搜索打下坚实基础.

其次, 循环问题是局部搜索算法固有的问题. 为了提高算法效率, 本文提出了一种基于年龄属性、双层格局检测策略和禁忌策略的方法来避免循环问题. 同时, 本文提出了一个扰动策略, 当算法陷入局部最优时, 该策略可以帮助算法有效地跳出.

最后, 本文综合运用反馈机制、年龄属性、双层格局检测策略、禁忌策略、扰动策略以及两个评分函数 *Dscore* 和 *Nscore*, 提出了一种有效的顶点选择方法和局部搜索算法. 实验结果表明, 与著名的 CPLEX 求解器、MWCDS、MA、GRASP 和 GRASP_impLS 算法相比, 本文提出的局部搜索算法获得了更优的结果.

本文第 2 节介绍了相关概念. 第 3 节详细描述了算法的框架和相关策略的细节. 第 4 节对本文提出的算法与其他算法在 4 组实例上的实验结果进行了对比分析. 第 5 节对实验部分进行了讨论分析. 最后, 第 6 节是本文的结论.

2 相关概念

对于一个无向连通图 $G(V, E)$, 其中 $V = \{v_1, v_2, v_3, \ldots, v_n\}$表示图 G 的顶点集, $E = \{e_1, e_2, e_3, \ldots, e_m\}$表示图 G 的边集. $N(v)$ 表示顶点 v 的邻居集合但不包括顶点 v, 记为 $N(v)= \{u \in V \mid \{u, v\} \in E\}$. $d(v) = |N(v)|$表示 v 的度. $N[v]$ 表示顶点 v 的邻居集合并且包含顶点 v, 记为 $N[v] = N(v) \cup \{v\}$. $N[D]$ 表示 D 的邻居集合并且包含集合 D, 记为 $N[D] = \bigcup_{v \in D} N[v]$. 假设 u 和 v 是图中的两个顶点, 他们之间的最短路径记为 $hop(u, v)$. 顶点 v 的 i 跳邻居记为 $N_i(v) = \{u \mid 1 \leqslant hop(u, v) \leqslant i\}$, 并且 $N_i[v] = N_i(v) \cup \{v\}$. 当 $i = 1$ 时, $N_i(v) = N(v)$, $N_i[v] = N[v]$. 下面, 给出本文中将会提到的一些基本定义.

定义 1 (**支配集**). 给定一个无向图 $G(V, E)$, 若顶点子集 $D \subseteq V$ 使得 $V \setminus D$ 中的每一个顶点都至少与 D 中的一个顶点相邻, 则 D 是一个支配集.

定义 2 (**连通支配集**). 给定一个无向连通图 $G(V, E)$, 若顶点子集 $D \subseteq V$ 是一个支配集, 并且 D 中的任意两个顶点之间存在一条路径, 且该路径上的顶点都在 D 中, 则集合 D 是一个连通支配集.

定义 3 (**弱诱导子图**). 给定一个无向连通图 $G(V, E)$ 和一个顶点子集 $D \subseteq V$, 子图 $WIS_G(D) = (N[D], E \cap (D \times N[D]))$ 被称为基于 D 的弱诱导子图, 其中 $N[D]$ 包含集合 D 及 D 中顶点的邻居顶点, 即 $N[D] = \bigcup_{v \in D} N[v]$.

定义 4 (**弱连通支配集**). 给定一个无向连通图 $G(V, E)$, 若顶点子集 $D \subseteq V$ 是一个支配集, 并且弱诱导子图 $WIS_G(D)$ 是连通的, 则集合 D 是一个弱连通支配集.

定义 5 (**最小弱连通支配集问题**). 给定一个无向连通图 $G(V, E)$, 图 G 的最小弱连通支配集问题是找出一个具有最小基数的弱连通支配集.

定义 6 (**锁定顶点**, **locked vertex**). 给定一个无向连通图 $G(V, E)$, 如果图 G 中存在一个顶点 v, 且该顶点的度为 1, 即 $d(v) = 1$, 那么顶点 v 的邻居顶点 u 被定义为锁定顶点.

本文采用了两个评分函数来引导局部搜索, 这两个评分函数分别是 *Dscore* 和 *Nscore*. 对于顶点 v, *Dscore*(v) 表示当顶点 v 加入或移出当前候选解时, 该顶点的邻居从非支配变成支配或者从支配变成非支配的顶点数量, 如公式 (1) 所示:

$$Dscore(v)=\begin{cases}|N[D\backslash\{v\}]|-|N[D]|, \text{如果} v\in D\\ |N[D\cup\{v\}]|-|N[D]|, \text{否则}\end{cases} \tag{1}$$

其中, $N[D]$ 包括了当前候选解 D 中的顶点和它的邻居顶点, 因此 $N[D]$ 也可以表示当前候选解 D 所支配的顶点集. 同样, $|N[D\backslash\{v\}]|$表示集合 $D\backslash\{v\}$所支配的顶点个数, $|N[D\cup\{v\}]|$表示集合 $D\cup\{v\}$所支配的顶点个数. 根据公式 (1) 可知, 当顶点 v 被加入候选解中后, $Dscore(v)$ 为非正数, 当顶点 v 从候选解中移出后, $Dscore(v)$ 为非负数.

对于顶点 v, $Nscore(v)$ 则表示顶点 v 的邻居顶点集合中有多少个顶点是在候选解中, 如公式 (2) 所示:

$$Nscore(v)=|N(v)\cap D| \tag{2}$$

$Nscore(v)$ 可以反映出当顶点 v 被移除时, 对当前候选解可行性的影响, 即当从解中移除顶点 v 时, 其邻居顶点集合中的在解中的顶点数量将减少 1. 因此, 当某个顶点的 $Nscore$ 值降低至 1, 并且该顶点被移除时, 当前候选解将变得不可行.

3 最小弱连通支配集问题的求解

在本节中, 提出了一种高效的局部搜索算法求解最小弱连通支配集问题. 该算法框架中提出了基于年龄属性、双层格局检测策略和禁忌策略相结合的避免循环搜索方法、扰动策略和反馈机制, 以优化局部搜索过程. 关于循环问题的详细描述请参见第 3.1 节. 在第 3.2 节中介绍从当前候选解的弱连通元素中选择要添加的顶点的方法, 该方法能够将多个弱诱导子图转变为连通状态. 扰动策略在第 3.3 节中进行介绍, 反馈机制在第 3.4 节中进行阐述. 顶点选择方法在第 3.5 节中进行说明, 算法框架在第 3.6 节中进行介绍.

3.1 循环问题

循环问题是局部搜索算法的固有问题, 即算法往往会重复访问最近已经访问过的顶点, 这会导致计算资源的浪费并且影响了算法的性能. 目前尚无完全解决这一问题的方法. 为了有效避免循环问题, 本文提出了年龄属性, 双层格局检测策略和禁忌策略相结合的方法, 从而改善算法性能. 年龄属性用于判断是否将顶点加入候选解或从当前候选解中移出; 禁忌策略用于防止刚刚加入解中的顶点被立即删除; 双层格局检测策略用于选择要添加到候选解中的顶点. 下面, 将进一步介绍这 3 种方法.

3.1.1 年龄属性

在局部搜索过程中, 为了得到一个更优质的候选解, 经常需要添加或删除一些顶点以改变当前候选解的结构. 因此, 选择哪些顶点进行添加或删除变得非常重要. 然而, 在局部搜索的过程中, 往往会无意地重复删除或加入同一个顶点, 从而导致出现循环问题. 为了有效解决这个问题, 本文提出了一种称为年龄属性的方法. 年龄属性通过为每个顶点赋予一个“年龄值”, 年龄值表示了一个顶点在解中或者不在解中的状态下持续的时间, 这直接指导了顶点的选择优先级. 在选择顶点时, 会优先考虑那些年龄值较大的顶点. 这是因为, 如果一个顶点在解中, 并且该点的年龄值越大, 那么这意味着它在解中存在的时间更长; 反之, 如果一个顶点不在解中, 并且该点的年龄值越大, 就说明它在不在解中的状态下持续的时间更长. 因此, 基于年龄值的优先级排序可以避免顶点的状态长时间不被改变, 这减少了重复访问相同顶点导致的循环问题, 增加了搜索过程中的新鲜度, 鼓励算法探索未充分检验的解空间区域. 一旦年龄大的顶点的状态被改变, 很可能带来结构上的新信息, 推动算法朝向全局或更优解迈进. 这种策略鼓励算法在搜索中不断探索新的搜索空间, 加快了算法对新解的发现. 年龄属性的实现非常简单且计算复杂度较低. 它通过使用 age 值来表示每个顶点的年龄. 具体而言, 其值的初始化和更新规则如下.

年龄更新规则 1: 在初始化阶段, 对于每个顶点 $v\in V$, $age[v]$ 被设置为 0.

年龄更新规则 2: 在整个算法框架中, 当一个顶点 v 添加到解中时, 对于该顶点的 $age[v]\leftarrow 1$, 而对于顶点 $u\in V\backslash\{v\}$, $age[u]$ 加 1.

年龄更新规则 3: 在局部搜索阶段, 当一个顶点 v 从当前解中移出时, 对于该顶点的 $age[v]\leftarrow 1$.

3.1.2 禁忌策略

禁忌策略, 又被称为 tabu 策略, 最初由 Glover[21]提出, 并被广泛应用于各种局部搜索算法中, 旨在解决循环问

题. 该策略的主要作用是禁止对刚刚改变状态的顶点进行再次翻转. 禁忌策略有一个重要参数, 即禁忌长度, 用于限制在一定步数内, 刚刚改变状态的顶点不会再次被翻转. 在本文提出的算法中, 禁忌长度被设置为 1. 在每轮循环中, 将添加到解中的顶点加入禁忌列表, 然后在下一轮的循环中, 在选择移除候选解中的顶点时, 禁止移除禁忌列表中的顶点.

3.1.3 双层格局检测策略

顶点的格局具有重要的物理意义, 它反映了一个图中特定顶点与其邻居顶点之间关系的当前状态. 在求解最小顶点覆盖 (MVC) 问题时, 为了避免局部搜索算法陷入循环, 即重复访问相同的候选解, Cai 等人[22]提出了格局检测 (configuration checking, CC) 策略. 该策略在向候选解中添加顶点时考虑顶点的格局, 即当一个顶点 v 的格局自上次从候选解中删除以来没有改变时, 禁止将该顶点添加到当前候选解中. 该策略是一种思想, 易于实现, 并可以根据问题和算法的不同提出不同的格局检测策略, 格局检测策略已经被广泛应用于组合优化问题的局部搜索算法中, 例如最小加权顶点覆盖问题[23]和最小可满足性问题[24]等.

在本文提出的算法中采用双层格局检测策略 (two hop configuration checking, CC^2). 与原始的格局检测策略不同的是, CC^2 策略考虑了顶点的 2 层邻居的格局. 也就是说, 传统的格局检测只考虑顶点及其直接邻居的格局状态, 而双层格局检测扩展到考虑顶点的 2 层邻居, 即顶点的邻居的邻居. 这种扩展不仅考虑了更宽泛的影响范围, 还提升了对网络结构敏感性的把握, 帮助算法识别那些在更大范围内能引起解结构变化的顶点变动, 从而避免了仅在局部小范围内无效的尝试. 当顶点 v 加入候选解或从候选解中移出时, 该顶点的 1 层邻居和 2 层邻居的格局都会被改变为允许被加入候选解的状态. 初步测试表明, 通过考虑两层邻居的动态, CC^2 策略在添加或移除顶点时, 能更准确判断对整体结构的影响, 减少不必要的扰动. 同时, 它促进了搜索过程中解的多样性, 因为对更远距离的考虑自然引入了更多的解空间探索路径, 避免了算法过早陷入局部最优解从而可以得到令人满意的结果.

与原始的格局检测策略相同, 如果一个顶点的格局自上次从解中删除以来没有改变, 则禁止该顶点加入候选解. 在实现中, 本文使用 *ConfChange* 值来表示每个顶点的格局值. 当 $ConfChange[v] = 1$ 时, 表明顶点 v 的格局已经改变, 可以将该顶点添加到当前候选解中. 当顶点 v 的 $ConfChange[v] = 0$ 时, 表示顶点 v 的格局自上次从候选解中删除以来没有发生变化, 因此不能将其添加到当前候选解中. 具体而言, *ConfChange* 值将根据以下规则进行初始化和更新.

格局更新规则 1: 在算法开始时, 对于每个顶点 $v \in V$, $ConfChange[v]$ 设置为 1.

格局更新规则 2: 当顶点 v 添加到候选解中时, 对于顶点 $u \in N_2(v)$, $ConfChange[u]$ 设置为 1.

格局更新规则 3: 当顶点 v 被移除时, $ConfChange[v]$ 设置为 0; 对于顶点 $u \in N_2(v)$, $ConfChange[u]$ 设置为 1.

3.2 候选解的弱连通元素

在局部搜索过程中, 首先会移除一些顶点, 以破坏当前候选解的结构, 从而获得不同的候选解. 这些被移除的顶点导致当前候选解变得不再连通, 形成了几个弱连通的分量. 因此, 在局部搜索过程中, 尝试找到一些顶点, 使得这些弱连通分量之间能够相互连通.

为了实现这一目标, 本文引用了 Niu 等人[16]提出的 GRASP_impLS 算法中的集合 *SWCE*. 集合 *SWCE* 包含了当前候选解的弱连通元素. 它由每个弱连通分量的两层邻居中出现次数最多的顶点组成. 假设该候选解 $W = W_1 \cup W_2 \cup \ldots \cup W_k$, 其中 $W_1, W_2, \ldots, W_k$ 是 k 个弱连通分量, $SWCE = \{v \mid sum\_A(v) = \max\{sum\_A(n) = \sum_{i=1,\ldots,k} A(n, i), n \in V \backslash W\}, v \in V \backslash W\}$, 其中如果 n 出现在 $N_2(W_i)$ 中, $A(n, i) = 1$, 否则, $A(n, i) = 0$.

3.3 扰动策略

在运用上述策略后, 通常可以得到一些高质量的解. 然而, 这些解仍然可能受限于局部最优的问题. 在本文的算法中, 当连续迭代后最优解的质量未能提升时, 搜索进程可能会停滞, 无法进一步优化结果. 为了帮助局部搜索摆脱这种陷入局部最优的情况, 本文提出了一种扰动策略来解决这个问题. 这个扰动策略通过翻转一些顶点来进行.

扰动策略通过在算法陷入局部最优时随机改变解的状态, 引入了一定程度的不确定性, 从而主动增强了算法

的探索能力. 通过这种方法, 算法得以在已知的解空间附近进行探索, 而不是简单地陷入重复无效的局部搜索循环中. 这有助于发现新的搜索路径并跳出现有的搜索局限. 扰动的动态性体现在扰动强度的设定上, 它可以根据当前搜索状态的特征动态地调整, 以更好地平衡探索和利用. 扰动强度是扰动策略中一个重要的参数, 它对扰动效果产生极大影响. 如果扰动强度过大, 将导致算法重新启动; 而如果扰动强度过小, 则无法跳出局部最优值, 或很容易陷入循环. 为了适应不同规模的实例, 本文设置扰动的顶点数量为 $0.01 \times n$ (n 为顶点数). 大量实验证明, 这样的扰动强度能够适应不同情况, 既不过于剧烈也不过于微弱, 确保了在不完全放弃已取得进展的同时, 也能够探索新的解空间. 因此, 当当前候选解陷入局部最优时, 会随机翻转 $0.01 \times n$ 个顶点, 以跳出局部最优, 通过这种随机翻转一定比例的顶点来扰动候选解, 使得算法能够在更大范围内重新评估和优化解结构, 从而避免了循环现象, 增加了跳出局部最优的可能性. 这种策略增强了算法的全局搜索能力, 并且确保即使对于 NP 难问题也能得到较高质量的近似解. 此外, 如果当前找到的最小解在 50 次循环内未能提升解的质量, 则会进行一次扰动.

3.4 反馈机制

本文提出的算法框架分为两个主要阶段: 构建初始解阶段和局部搜索阶段. 在构建初始解阶段, 希望选择加入候选解的顶点具有很高的概率出现在最优解中, 以获得更好的初始解质量, 从而提高最终优化结果的质量. 为了实现这一目标, 本文提出了一种反馈机制. 该机制通过利用上一轮迭代过程中的信息来指导本轮的搜索过程, 即将这些历史学习成果应用于下一轮的顶点选择. 这种机制确保了算法能够利用过去的经验, 尤其是那些在多次被验证为有效顶点的加入候选解中的顶点, 从而提高算法从一个较优初始解开始进行局部搜索的能力. 具体来说, 反馈机制主要体现在对顶点选择策略的影响上.

在局部搜索阶段, 本文将通过一些策略来选择要添加到当前解中的顶点. 如果一个顶点多次被加入当前候选解中, 那么它出现在最优解中的概率也会更大, 这种反馈机制使得算法在新一轮搜索开始时就能基于历史经验做出更加明智的选择. 为了衡量顶点加入候选解的频率, 本文提出了一个表示频率的变量 *add_frequency*. 每当一个顶点被添加到候选解中一次, 该顶点的 *add_frequency* 的值就增加 1.

随后, 将每个顶点的频率信息在算法重新启动后的构建初始解阶段进行反馈. 在这个阶段, 在面对多个 *Dscore* 值相同的顶点选择时, 优先考虑 *add_frequency* 值高的顶点, 这是一种基于历史成功经验的智能决策. 它减少了随机性, 使得算法的顶点选择更为明智, 这在算法的初期尤为重要, 因为高质量的初始解能有效引导后续的搜索方向. 并且通过反馈机制, 算法能够避免重复探索那些在过往迭代中已经证明不太可能有效或低效的顶点, 这大大降低了搜索空间, 缩短了达到较优解所需的时间, 提升了算法效率. 同时, 该机制使得算法能够根据问题的具体实例动态调整搜索路径, 对于不同的图结构, 频繁出现在解中的顶点可能指示了特定的结构特征, 算法通过学习这些特征动态优化搜索策略, 更加聚焦于关键顶点的选择, 避免了盲目探索. 需要注意的是, 每次反馈频率信息后, *add_frequency* 值会被重置为 0, 这确保了每轮迭代都能基于最新的反馈信息作出决策, 避免了过时信息的干扰, 保持算法的灵活性和适应性.

3.5 顶点选择方法

本文提出的算法一共包括两个阶段: 一个是构建初始解阶段, 一个是局部搜索阶段. 在构建初始解阶段中, 需要挑选顶点加入解中, 那么选择哪些顶点就变成极其重要, 因为初始解影响最终找到的最优解的质量.

在局部搜索阶段, 算法会不停地交换当前候选解内的顶点与当前候选解以外的顶点, 以优化初始解. 因此, 在此阶段选择一些合适的顶点进行交换就变得尤为重要, 因为这可以尽可能避免算法陷入局部最优, 从而整体提高算法的性能. 因此, 算法结合了 *Dscore*, *Nscore* 以及上述所提到的所有策略, 提出了以下几个顶点的选择规则.

添加顶点规则 1: 选择一个 *Dscore* 值最大且 *ConfChange* 值为 1 的顶点, 如果有多个顶点具有相同的最大的 *Dscore* 值, 则随机选择一个 *add_frequency* 值最大的顶点.

添加顶点规则 2: 选择一个 *Dscore* 值最大且 *ConfChange* 值为 1 的顶点, 如果有多个顶点具有相同的最大的 *Dscore* 值, 则从中随机选择一个顶点.

添加顶点规则 3: 选择一个 *Dscore* 值最大且 *ConfChange* 值为 1 的顶点, 如果有多个顶点具有相同的最大的

Dscore 值, 则选择一个 *age* 值最大的顶点.

移出顶点规则 1: 设置固定概率 *wp*, 随机产生一个概率 *p*, 若 $p < wp$, 选择一个 *Dscore* 值最大的顶点, 若 $p \geqslant wp$, 选择一个 *Nscore* 值最小的顶点, 如果有多个满足条件, 则选择一个 *age* 值最大的顶点, 但该顶点不能是锁定顶点.

移出顶点规则 2: 此选择规则与移出顶点规则 1 相同, 但所选的顶点不是锁定顶点, 且不在禁忌列表中.

3.6 基于反馈机制和扰动策略的局部搜索算法

3.6.1 FPLS 算法框架

该算法是一种多启动元启发式算法, 它一共包括两个过程: 基于锁定顶点和反馈机制的初始候选解构建过程 (an initial construction procedure based on the locked vertex and the feedback mechanism, LF_InitConstruct) 和基于的反馈机制和扰动策略的局部搜索过程 (a local search procedure based on the feedback mechanism and the perturbation strategy, FPLS_Search), 共同求解最小弱连通支配集问题. 当 FPLS_Search 过程达到最大迭代次数时, 算法重新启动并从一个新的初始候选解开始搜索, 直到达到算法终止条件, 算法停止. 在算法 1 中描述了 FPLS 算法的框架.

算法 1. FPLS 算法.

输入: 无向图 $G(V, E)$, 算法最大的内外层迭代次数 *ITER_NUM* 和 *OUTTER_STEP*, 算法最长运行时间 *cutoff*;
输出: 全局最优的弱连通支配集 *W**.

```
1. 图中每个顶点的 add_frequency[v] 值初始化为 0;
2. 外层循环次数 step 初始化为 0;
3. W* ← V;
4. WHILE step < OUTTER_STEP 且 elapsed time < cutoff DO
5.         IF step%2 == 0 THEN
6.             图中每个顶点的 add_frequency[v] 值初始化为 0;
7.         W ← LF_InitConstruct( ); /*构建初始解*/
8.         W' ← FPLS_Search(W, ITER_NUM); /*优化初始解*/
9.         IF |W'| < |W*| THEN
10.            W* ← W'; /*更新最小解*/
11.        step++;
12. RETURN W*;
```

如算法 1 所示, 第 1 行中, 每个顶点的 *add_frequency* 值初始化为 0, 表示每个顶点在局部搜索过程中还未被添加到候选解中. 第 2 行中, 循环步数被初始化为 0. 主循环从第 4–11 行, 在满足运行时间小于截止时间或循环次数小于最大迭代次数 *OUTTER_STEP* 的循环条件下执行伪代码. 在第 5/6 行中, 对每个顶点的 *add_frequency* 进行初始化, 表示每反馈一次顶点信息, *add_frequency* 就要被初始化一次. 第 7 行调用函数使用贪心策略构造初始解, 第 8 行调用函数对初始解进行优化. 在第 9/10 行中, 当找到更好的解时, 当前解会被更新为最小解, 并且在第 12 行返回所找到的最小解.

3.6.2 LF_InitConstruct 过程

初始解的质量直接影响到局部搜索算法改进的结果. 为得到较高质量的初始解, 本文提出了名为 LF_InitConstruct 的初始构造过程, 它基于锁定顶点和顶点的反馈信息. 在该过程中, 采用贪心策略来构造初始解, 利用顶点的 *Dscore* 值和上一轮迭代反馈的信息 (顶点加入候选解的频率) 来指导如何选择顶点. 在此过程中, 如果一个顶点满足锁定顶点的属性, 则该点将被优先添加到候选解中. 原因在于, 如果图上的度为 1 的顶点的邻居 (锁定顶点) 不添加到解中, 那么度为 1 的顶点及其第 2 层邻居必须在解中, 这样才能保证候选解的弱连通性, 但会导致候选解中顶点过多. 但是如果度为 1 的顶点的邻居 (锁定顶点) 在解中, 那么度为 1 的顶点和它的第 2 层邻居就可以不在候选解中, 这样候选解中的顶点数相对较少. 因此, 认为度为 1 的顶点的邻居 (锁定顶点) 加入候选解中, 在算

法后续的执行过程中可以不用关注这些度为 1 的顶点, 锁定顶点及其邻居顶点. 这样的做法可以有效地减少局部搜索阶段的搜索空间, 从而有利于算法更快地找到更优的解. 具体地, LF_InitConstruct 过程在算法 2 中得到了详细描述.

算法 2. LF_InitConstruct 过程.

输入: 无向连通图 $G(V, E)$;
输出: 初始的弱连通支配集 W.

1. 每个顶点 $Dscore(v)$ 值初始化为$|N[v]|$;
2. 图中每个顶点的 $ConfChange[v]$ 值初始化为 1;
3. 图中每个顶点的 $age[v]$ 值初始化为 0;
4. $W \leftarrow \varnothing$;
5. 根据添加顶点规则 2 从所有顶点中挑选一个顶点 w;
6. $W \leftarrow W \cup \{w\}$;
7. 顶点 w 的年龄值 $age[w]$ 初始化为 1, 其余顶点 v 的年龄值 $age[v]$ 加 1;
8. **WHILE** 当前解 W 不是一个弱连通支配集 **THEN**
9. **IF** 当前解 W 的 2 层邻居内存在一个顶点 v 为锁定顶点 **THEN**
10. $w \leftarrow v$;
11. **ELSE**
12. 根据添加顶点规则 1 从当前解 W 的 2 层邻居中挑选一个顶点 w;
13. $W \leftarrow W \cup \{w\}$;
14. 根据年龄更新规则 2 更新顶点 w 的年龄值 $age[w]$ 为 1, 其余顶点 v 的年龄值 $age[v]$ 加 1;
15. **RETURN** W;

在算法 2 的第 1 行中, 将每个顶点的 $Dscore(v)$ 值初始化为其度数加 1. 在第 2 行中, 每个顶点的 *ConfChange* 值初始化为 1, 表示每个顶点都可以被添加到候选解中. 第 3 行中, 每个顶点的 *age* 值被初始化为 0, 表示每个顶点都没有被添加到候选解中. 在第 4 行中, 将当前的候选解 W 设置为空集. 然后, 在第 5 行中, 为了打破随机性, 根据添加顶点规则 2, 从顶点集 V 中选择具有最大 *Dscore* 值的初始顶点. 接着, 在第 6/7 行中, 选定的顶点 w 被添加到集合 W 中, 并使用年龄更新规则 2 对每个顶点的年龄值 *age* 进行更新. 接下来, 该算法逐个将顶点添加到候选解中, 直到候选解成为一个可行解 (第 8–14 行). 在此阶段, 如果 $N_2(W)$ 中包含一个锁定顶点, 则选择该顶点 (第 9/10 行); 否则, 选择 $N_2(W)$ 中具有最大 *Dscore* 值的顶点. 如果存在多个顶点具有相等的 *Dscore* 值, 则选择 *add_frequency* 值最大的顶点 (第 12 行). 在第 13/14 行中, 将选定的顶点 w 添加到候选解 W 中, 并根据年龄更新规则 2 更新每个顶点的年龄值 *age*. 最终, 在第 15 行中, 当当前解 W 成为弱连通支配集时, 跳出循环并返回初始解 W.

3.6.3 FPLS_Search 过程

根据算法 1, 在 LF_InitConstruct 过程中构造了初始解后, 需要局部搜索过程 FPLS_Search 中对该初始解进行优化, 以得到一个规模更小的弱连通支配集. 为了避免循环问题, 本文结合了年龄属性、双层格局检测策略和禁忌策略, 并使用扰动策略使得当前候选解跳出局部最优. 在 FPLS_Search 过程中, 若连续 *Noimprovestep* 次迭代候选解的质量没有优化, 则认为当前解陷入到了局部最优, 执行扰动策略. 根据实验测得, *Noimprovestep* 值设为 50. 此外, 本文还引入了反馈机制, 以便在重启算法的新一轮 LF_InitConstruct 过程中选择大概率在最优解中的顶点, 从而获得质量更高的初始解.

在移除顶点的过程中, 如果只选择一个 *Dscore* 值最大的顶点, 就会增加 FPLS_Search 过程所面临的循环问题. 因此, 本文引入了 *Nscore* 函数, 它增加了所选顶点的多样性, 从而可以选择在前一个循环中未被选择为候选顶点的顶点. 同时, 本文引入了一个概率参数“*wp*”, 用来判断是选择一个最大的 *Dscore* 值或最小的 *Nscore* 值的顶点.

最后, 本文引用了一个候选集合 *SWCE* 来保存当前解的弱连通元素, 以确保添加的顶点能够使解变得可行. 基于以上讨论, FPLS_Search 过程在算法 3 中得到了详细描述.

算法 3. FPLS_Search 过程.

输入: 初始解 *W*, 最大循环迭代次数 *ITEM_NUM*;
输出: 局部最优的弱连通支配集 *W*′.

```
1. W′ ← W;
2. it ← 1;
3. Noimprovestep ← 1;
4. WHILE it < ITEM_NUM DO
5.    WHILE 当前解 W 是一个弱连通支配集 DO
6.       IF |W| < |W′| THEN
7.          W′ ← W; /*更新最小解*/
8.          Noimprovestep ← 1;
9.       ELSE
10.         Noimprovestep ++;
11.      根据移除顶点规则 1 从当前解 W 中挑选一个顶点 w;
12.      W ← W\{w};
13.      根据格局更新规则 3 更新顶点 w 的 2 层邻居内所有顶点 v 的格局值 ConfChange;
14.      根据年龄更新规则 3 更新顶点 w 的年龄值 age[w] 为 1;
15.   根据移除顶点规则 2 从当前解 W 中挑选一个顶点 w;
16.   W ← W\{w};
17.   根据格局更新规则 3 更新顶点 w 的 2 层邻居内所有顶点 v 的格局值 ConfChange;
18.   根据年龄更新规则 3 更新顶点 w 的年龄值 age[w] 为 1;
19.   IF Noimprovestep%50 = 0 THEN
20.      FOR i = 1 TO 0.01×|V|
21.         根据移出顶点规则 2 从当前解 W 中挑选一个顶点 w;
22.         将挑选的顶点 w 从当前解 W 中移出;
23.         根据格局更新规则 3 更新每一个移出顶点 w 的 2 层邻居内所有顶点 v 的格局值 ConfChange;
24.         根据年龄更新规则 3 更新顶点 w 的年龄值 age[w] 为 1;
25.   WHILE 存在没有被支配的顶点 DO
26.      根据添加顶点规则 2 从当前解 W 的 2 层邻居中挑选一个顶点 w;
27.      W ← W ∪ {w};
28.      根据年龄更新规则 2 更新顶点 w 的年龄值 age[w] 为 1, 其余顶点 v 的年龄值 age[v] 加 1;
29.      根据格局更新规则 2 更新顶点 w 的 2 层邻居内所有顶点 v 的格局值 ConfChange;
30.      add_frequency[w]++;
31.   WHILE 当前解 W 不是一个弱连通支配集 DO
32.      更新当前解的弱连通元素集合 SWCE;
33.      根据添加顶点规则 3 从当前解的弱连通元素集合 SWCE 中挑选一个顶点 w;
34.      W ← W ∪ {w};
35.      根据年龄更新规则 2 更新顶点 w 的年龄值 age[w] 为 1, 其余顶点 v 的年龄值 age[v] 加 1;
```

36. 根据格局更新规则 2 更新顶点 w 的二层邻居内所有顶点 v 的格局值 *ConfChange*;
37. *add_frequency*[w]++;
38. 将顶点 w 添加到 tabu 列表中;
39. *it*++;
40. **RETURN** W';

在算法 3 的描述中, 第 1–3 行将局部最优解初始化为当前解, 算法迭代步数和当前候选解的质量没有改善的迭代步数都初始化为 1. 外层循环从第 4–40 行. 在第 6–10 行中, 当当前候选解 W 为弱连通支配集时, 将 W 与局部最优解 W'进行比较. 如果 W 优于 W', 则将 W'更新为 W, 并将最小解没有改善的迭代次数 *Noimprovestep* 值初始化为 1; 否则, 将 *Noimprovestep* 值加 1. 然后, 在第 11/12 行中, 算法逐个从候选解中删除顶点, 直到候选解变得不可行. 在这个阶段, 算法根据移出顶点规则 1 从候选解中选择顶点. 具体来说, 随机生成一个概率 p, 如果 p 小于 wp, 则从候选解中选择一个具有最大 *Dscore* 值的顶点; 否则, 从候选者中选择一个具有最小 *Nscore* 值的顶点. 如果有多个顶点满足该条件, 则选择一个年龄值 *age* 最大的顶点, 但不能是锁定顶点. 接下来, 根据格局更新规则 3, 更新每个顶点 u 在 $N_2(w)$ 中的 *ConfChange*[u] 值 (第 13 行), 根据年龄更新规则 3 更新顶点 w 的年龄值 *age*[w] 为 1 (第 14 行). 在第 15/16 行中, 当 W 不是弱连通支配集时, 根据移出顶点规则 2 从候选解中删除一个顶点. 具体而言, 随机生成一个概率 p, 如果 p 小于 wp, 则选择一个具有最大 *Dscore* 值的顶点; 否则, 选择一个具有最小 *Nscore* 值的顶点. 如果有多个顶点满足该条件, 则选择一个年龄值 *age* 最大的顶点, 但该顶点不能是锁定顶点, 并且不能在禁忌列表中. 然后根据格局更新规则 3, 更新每个顶点 u 在 $N_2(w)$ 中的 *ConfChange*[u] 值 (第 17 行), 根据年龄更新规则 3 更新顶点 w 的年龄值 *age*[w] 为 1 (第 18 行). 在第 19–22 行中, 当当前候选解的质量没有改善的迭代步数达到 50 步时, 根据移出顶点规则 2 从候选解中删除 $0.01\times|V|$个顶点. 接着根据格局更新规则 3, 更新每个顶点 u 在 $N_2(w)$ 中的 *ConfChange*[u] 值 (第 23 行), 根据年龄更新规则 3 更新顶点 w 的年龄值 *age*[w] 为 1 (第 24 行). 在第 25–27 行中, 当存在没有被支配的顶点时, 根据添加顶点规则 2, 算法从顶点集合 $N_2(W)$ 中随机选择一个具有最大 *Dscore* 值的顶点. 然后将该顶点添加到候选解中, 并根据年龄更新规则 2 和格局更新规则 2 (第 28/29 行), 更新每个顶点的年龄值 *age* 以及每个顶点 u 在 $N_2(w)$ 中的 *ConfChange*[u] 值. 在第 30 行中, *add_frequency*[w] 值加 1. 接下来, 在第 32 行中, 更新集合 *SWCE*. 在第 31–38 行中, 当 W 不是弱连通支配集时, 算法将逐一添加一些顶点到候选解中, 直到候选解成为一个可行的弱连通支配集. 在这个阶段, 算法根据添加顶点规则 3 从集合 *SWCE* 中选择顶点. 具体而言, 选择一个具有最大 *Dscore* 值的顶点. 如果有几个顶点具有相同的最大值, 并且其中有一个 *ConfChange* 值等于 1 的顶点, 则选择一个年龄值 *age* 最大的顶点 (第 33 行). 然后根据年龄更新规则 2 和格局更新规则 2, 更新每个顶点的年龄值 *age* 以及每个顶点 u 在 $N_2(w)$ 中的 *ConfChange*[u] 值 (第 35/36 行). 在第 37 行中, *add_frequency*[v] 值加 1. 然后将添加到解中的顶点添加到 tabu 列表中 (第 38 行). 最后, 在第 40 行中, 返回局部最优解 W'.

4 实验比较

本节进行多组实验来评估本文提出的算法 FPLS. 在随机 UDG 实例、LPNMR'09 实例、一般 UDG 实例和 NDR 实例上对算法进行测试. 4 组实例均由牛当当为本文提供. 将 FPLS 与 CPLEX 精确求解器和 MWCDS、MA、GRASP 及 GRASP_impLS 这 4 种最先进的算法进行比较. 接下来, 对 4 组实例、CPLEX 求解器及 4 种对比算法进行介绍.

4.1 实例介绍

随机 UDG 实例: 该组基准实例采用 Jovanovic 等人[25]提出的方法随机生成的, 该方法来自自组织网络聚类问题. 该问题是最小弱连通支配集问题的重要应用场景. 该组基准实例共有 24 个, 实例的顶点数取值范围为 15–30, 规模相对较小.

LPNMR'09 实例: 该组基准实例源于第 10 届国际逻辑规划和非单调推理会议, 该组基准实例共有 9 个, 实例的顶点数取值范围为 40–90, 规模相对较小.

一般 UDG 实例: 该组基准实例最初由 Jovanovic 等人[25]提出, 实例来源于自组织网络聚类问题. 该组基准实

例共有 41 个, 实例的顶点数取值范围为 80–400, 规模相对较大.

NDR 实例: 该组基准实例是由在线网络数据存储库 (network data repository, NDR) 中具有 100–21 498 个顶点的无权连通图转换而来的, 该组基准实例共有 24 个, 规模分布较广.

4.2 比较算法

CPLEX 是一种通用的混合整数规划求解器, 被广泛用于求解组合优化问题. 若给予足够的求解时间和空间, 则 CPLEX 最终一定能找到精确解. 在本文的实验中, CPLEX 的结果是由 CPLEX v12.9[16]求解得到.

MWCDS 是由 Ding 等人[13]提出的一种线性时间自稳定算法, 而 MWCDS 是一种线性时间复杂度算法. MWCDS 对具有 n 个顶点的任意连通图使用同步守护进程在 O(n) 步骤中终止.

MA[15]是一种求解最小弱连通支配集问题的模因算法, 它首先初始化种群, 然后通过选择、交叉、变异等操作产生一个不可行解, 再通过 MWCDS 算法将该不可行解变成可行解, 随后通过一个简单的局部搜索过程来优化该可行解.

GRASP[16]是一种求解最小弱连通支配集问题的贪婪随机自适应搜索算法, 它利用平衡随机和贪婪策略构造初始解, 然后利用简单的局部搜索对初始解进行优化.

GRASP_impLS[16]是对 GRASP 算法的改进, 改进部分主要在局部搜索阶段, 它引用了概率参数来增加删除顶点的多样性, 同时如果当前解从不可行到可行时, 首先从当前候选解的 2 层邻居内选择顶点使当前候选解变为支配集, 然后从集合 *SWCE* 中选择顶点使当前的候选解变成一个弱连通支配集.

4.3 实验环境与参数

本文提出的 FPLS 算法用 C++编程语言实现的, 并由 GNU g++使用-O 选项进行编译. 本文的算法实验是在一台具有 Intel(R) Core(TM) i5-11400H@2.70 GHz 和 8 GB 内存的计算机上进行的. CPLEX、MWCDS、MA、GRASP 和 GRASP_impLS 的实验结果见文献 [16]. 对于 CPLEX 求解器, 设置了 3 600 s 的时间限制. 本文提出的算法和其他 4 个对比算法在每个实例上都运行了 10 次, 并记录找到的最小解和平均解, 并将求解每个实例的截止时间设置为 300 s. 为了获得更多高质量的候选解, 针对所提出的算法, 进行了多次实验, 最终确定了其中涉及的参数. 其中, 外层迭代次数和内层迭代次数均设置为 1 000, 概率参数 wp 设置为 0.35, 扰动顶点个数设置为 $0.01\times|V|$.

4.4 在随机 UDG 实例上的实验结果

表 1 呈现出了 CPLEX 求解器和其他算法在随机 UDG 实例上的实验结果. “Nodes”列表示 Area($N\times N$) 领域范围内的顶点数量. R 和$|E|$列分别表示文献 [25] 中提到的半径值和实例的边数. CPLEX 求解器求解出来的结果中, “Solu”列表示在指定的时间内求解出来的最好的解, 当求解时间超过了 3 600 s 依然没有找到一个可行解, 那么将这个解记录为“N/A”, “Stat”列表示找到最优解的时间, 当运行时间达到 3 600 s, CPLEX 停止运行, 此种情况无法保证找到解的最优性, 此时会被记录为“Feasible”, 这些符号定义用于表 1–表 4. 对于 MWCDS、MA、GRASP、GRASP_impLS 和 FPLS 算法, “Best”列表示对应算法找到的最小解, “Avg”列表示 10 次运行所得到的平均解. 在 MWCDS 中, “time”列表示找到最小解所需的时间. 每个表的底部还求出了每个算法在每组实例上的“最小值”“最大值”“平均值”和“标准差”.

从表 1 中可以看出, 对于该组的 24 个实例, CPLEX 求解器在 14 个实例上得到了最优解, 在 7 个实例上给出了可行解, 但并未能证明其最优性, 在其余的 3 个实例上, 无法在给定的时间内找到一个可行解. 对比 5 个启发式算法可发现, MA、GRASP_impLS 和 FPLS 能够在所有 24 个随机 UDG 实例上找到最小解. 而 MWCDS 和 GRASP 分别可以找到 13 个和 18 个最小解. 平均解方面, 本文提出的算法 FPLS 能找到更优的平均解, 除了在实例 (100_25_25_45) 上, 在其他 23 个实例上 MA、GRASP_impLS 和 FPLS 得到的平均解相等. 而 MWCDS 和 GRASP 分别在 5 个和 18 个实例上能得到与 FPLS 相等的平均解. 总的来说, MA、GRASP_impLS 和 FPLS 在随机 UDG 实例上的最小解和平均解均优于 MWCDS 和 GRASP 算法. 此外, MA、GRASP_impLS 和 FPLS 算法在这组实例上的“最小值”“最大值”“平均值”和“标准差”均相等, 并且优于 MWCDS 和 GRASP 算法.

表 1 在随机 UDG 实例上的实验结果

Nodes	R	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		FPLS	
			Solu	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	Best	Avg
15 (N=80)	15	18	**6**	5.5	**6**	**6.6**	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	16	22	**6**	4.42	**6**	**6.0**	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	17	22	**6**	4.11	**6**	**6.0**	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	18	19	**6**	3.54	**6**	**6.0**	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	19	19	**6**	3.23	**6**	**6.0**	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	20	22	**4**	0.69	5	5.5	<0.01	**4**	**4.0**	**4**	**4.0**	**4**	**4.0**	**4**	**4.0**
20 (N=100)	20	27	**8**	Feasible	**8**	8.5	<0.01	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**
	21	32	**6**	58.86	**6**	6.8	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	22	35	**5**	6.46	**5**	6.6	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
	23	34	**6**	62.77	7	7.4	<0.01	**6**	**6.0**	7	7.0	**6**	**6.0**	**6**	**6.0**
	24	29	**6**	47.91	**6**	7.3	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	25	34	**5**	5.19	6	6.4	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
25 (N=100)	20	44	**7**	Feasible	9	9.0	<0.01	**7**	**7.0**	8	8.0	**7**	**7.0**	**7**	**7.0**
	21	47	**7**	Feasible	8	8.8	<0.01	**7**	**7.0**	8	8.0	**7**	**7.0**	**7**	**7.0**
	22	49	**6**	753.01	8	8.8	<0.01	**6**	**6.0**	7	7.0	**6**	**6.0**	**6**	**6.0**
	23	53	**6**	1 296.45	7	8.5	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	24	59	**6**	2 240.62	7	7.8	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	25	45	**7**	Feasible	8	8.7	<0.01	**7**	7.1	**7**	**7.0**	**7**	7.1	**7**	**7.0**
30 (N=150)	25	46	N/A	N/A	12	12.3	<0.01	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**
	26	42	**9**	Feasible	10	12.0	<0.01	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**
	27	40	N/A	N/A	**11**	12.6	<0.01	**11**	**11.0**	13	13.0	**11**	**11.0**	**11**	**11.0**
	28	43	N/A	N/A	**10**	11.9	<0.01	**10**	**10.0**	11	11.0	**10**	**10.0**	**10**	**10.0**
	29	49	**9**	Feasible	**9**	10.6	<0.01	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**
	30	49	**9**	Feasible	**9**	10.6	<0.01	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**
最小值			—	—	5	5.5	—	4	4.0	4	4.0	4	4.0	4	4.0
最大值			—	—	12	12.6	—	11	11.0	13	13.0	11	11.0	11	11.0
平均值			—	—	7.54	8.36	—	7.00	7.00	7.29	7.29	7.00	7.00	7.00	7.00
标准差			—	—	1.91	2.24	—	1.89	1.89	2.16	2.16	1.89	1.89	1.89	1.89

注: 加粗数值是对应算法找到的最优值

表 2 在 LPNMR'09 实例上的实验结果

实例	$\|V\|$	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		FPLS	
			Solu	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	Best	Avg
40 × 200	40	200	**5**	Feasible	8	9.4	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
45 × 250	45	250	**5**	Feasible	8	9.2	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
50 × 250(1)	50	236	N/A	N/A	11	11.9	<0.01	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**
50 × 250(2)	50	240	N/A	N/A	9	11.3	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
55 × 250	55	250	N/A	N/A	10	12.3	<0.01	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**
60 × 400	60	400	N/A	N/A	10	11.0	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
70 × 250	70	250	N/A	N/A	17	19.4	<0.01	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**
80 × 500	80	500	N/A	N/A	14	15.9	<0.01	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**
90 × 600	90	600	N/A	N/A	16	17.8	<0.01	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**	**9**	**9.0**
最小值			—	—	8	9.2	0.00	5	5.0	5	5.0	5	5.0	5	5.0
最大值			—	—	17	19.4	0.00	11	11.0	11	11.0	11	11.0	11	11.0
平均值			—	—	11.44	13.13	0.00	7.11	7.11	7.11	7.11	7.11	7.11	7.11	7.11
标准差			—	—	3.40	3.68	0.00	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96

注: 加粗数值是对应算法找到的最优值

表 3 在一般 UDG 实例上的实验结果

Nodes	R	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		FPLS	
			Solu	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	Best	Avg
80 (N=400)	60	262	N/A	N/A	18	20.1	<0.01	**13**	**13.0**	**13**	**13.0**	**13**	**13.0**	**13**	**13.0**
	70	329	N/A	N/A	16	17.9	<0.01	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**
	80	400	N/A	N/A	13	15.7	<0.01	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**
	90	474	N/A	N/A	11	12.5	<0.01	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**	**8**	**8.0**
	100	563	N/A	N/A	9	11.1	<0.01	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**	**7**	**7.0**
	110	647	N/A	N/A	8	9.6	<0.01	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**	**6**	**6.0**
	120	735	N/A	N/A	8	9.2	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
100 (N=600)	80	335	N/A	N/A	21	23.5	<0.01	**15**	**15.0**	**15**	15.6	**15**	**15.0**	**15**	**15.0**
	90	368	N/A	N/A	21	22.5	<0.01	**14**	**14.0**	15	15.0	**14**	**14.0**	**14**	**14.0**
	100	435	N/A	N/A	18	20.0	<0.01	**12**	**12.0**	**12**	**12.0**	**12**	**12.0**	**12**	**12.0**
	110	500	N/A	N/A	17	18.5	<0.01	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**	**11**	**11.0**
	120	575	N/A	N/A	14	16.3	<0.01	**10**	**10.0**	**10**	**10.0**	**10**	**10.0**	**10**	**10.0**
200 (N=700)	70	756	N/A	N/A	43	47.3	<0.01	29	29.0	30	30.6	**28**	28.4	**28**	**28.0**
	80	921	N/A	N/A	37	39.0	<0.01	**24**	24.8	26	26.0	**24**	**24.0**	**24**	**24.0**
	90	1 113	N/A	N/A	29	32.7	<0.01	**20**	**20.0**	21	21.0	**20**	**20.0**	**20**	**20.0**
	100	1 305	N/A	N/A	27	28.6	<0.01	**18**	**18.0**	19	19.0	**18**	**18.0**	**18**	**18.0**
	110	1 501	N/A	N/A	23	25.3	<0.01	**16**	**16.0**	**16**	16.3	**16**	**16.0**	**16**	**16.0**
	120	1 730	N/A	N/A	20	22.5	<0.01	**13**	13.8	14	14.0	**13**	13.3	**13**	**13.0**
200 (N=1000)	100	756	N/A	N/A	43	47.3	<0.01	29	29.0	30	30.6	**28**	28.4	**28**	**28.0**
	110	871	N/A	N/A	40	42.5	<0.01	26	26.0	27	27.0	**25**	**25.0**	**25**	**25.0**
	120	997	N/A	N/A	32	36.2	<0.01	**23**	**23.0**	24	24.0	**23**	**23.0**	**23**	**23.0**
	130	1 127	N/A	N/A	29	32.9	<0.01	**20**	**20.0**	21	21.0	**20**	**20.0**	**20**	**20.0**
	140	1 269	N/A	N/A	28	29.7	<0.01	**18**	**18.0**	19	19.0	**18**	**18.0**	**18**	**18.0**
	150	1 400	N/A	N/A	26	27.6	<0.01	**16**	16.2	17	17.5	**16**	**16.0**	**16**	**16.0**
	160	1 541	N/A	N/A	23	24.8	<0.01	**15**	15.0	16	16.0	**15**	**15.0**	**15**	**15.0**
250 (N=1500)	130	903	N/A	N/A	57	60.7	<0.01	37	37.8	39	39.3	37	37.0	**36**	**36.0**
	140	1 011	N/A	N/A	52	54.1	<0.01	34	34.8	36	36.7	34	34.0	**33**	**33.0**
	150	1 119	N/A	N/A	44	49.2	<0.01	31	31.5	33	33.2	31	31.0	**30**	**30.0**
	160	1 246	N/A	N/A	44	45.4	<0.01	29	29.0	30	30.5	**28**	**28.0**	**28**	**28.0**
300 (N=2000)	200	1 577	N/A	N/A	50	53.5	<0.01	32	32.8	34	34.7	32	32.0	**31**	**31.0**
	210	1 710	N/A	N/A	48	50.4	<0.01	30	30.8	32	32.3	30	30.0	**29**	**29.0**
	220	1 849	N/A	N/A	46	47.0	<0.01	28	28.5	29	29.9	**27**	27.3	**27**	**27.0**
	230	1 990	N/A	N/A	41	43.7	<0.01	26	26.5	28	28.0	26	26.0	**25**	**25.6**
350 (N=2500)	200	1 461	N/A	N/A	72	76.7	<0.01	49	49.8	51	51.8	46	46.3	**45**	**45.0**
	210	1 555	N/A	N/A	69	73.2	<0.01	44	45.5	48	48.3	43	43.7	**42**	**42.0**
	220	1 668	N/A	N/A	65	69.1	<0.01	42	42.8	46	46.0	40	40.0	**39**	**39.0**
	230	1 787	N/A	N/A	60	64.4	<0.01	40	40.8	43	43.3	39	39.0	**38**	**38.0**
400 (N=3000)	210	1 522	N/A	N/A	90	94.4	<0.01	61	62.0	62	63.5	57	57.6	**55**	**55.0**
	220	1 621	N/A	N/A	84	88.6	<0.01	59	59.2	61	61.2	55	55.1	**53**	**53.0**
	230	1 750	N/A	N/A	79	85.0	<0.01	54	55.2	56	57.0	51	51.7	**50**	**50.0**
	240	1 880	N/A	N/A	77	80.1	<0.01	51	51.2	53	53.5	47	48.6	**46**	**46.2**
最小值			—	—	8	9.2	0.00	5	5.0	5	5.0	5	5.0	5	5.0
最大值			—	—	90	94.4	0.00	61	62.0	62	63.5	57	57.6	55	55.0
平均值			—	—	37.85	40.71	0.00	25.22	25.54	26.39	26.65	24.56	24.69	24.17	24.19
标准差			—	—	22.67	23.57	0.00	15.15	15.44	15.94	16.17	14.14	14.32	13.47	13.48

注: 加粗数值是对应算法找到的最优值

表 4 在 NDR 实例上的实验结果

实例	$\|V\|$	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		FPLS	
			Solu	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	Best	Avg
ba_1k_30k	1 000	30 039	N/A	N/A	114	121.5	<0.01	34	35.2	**21**	21.7	**21**	**21.0**	**21**	**21.0**
ba_1k_6k	1 000	5 964	N/A	N/A	273	284.0	<0.01	172	176.0	89	133.3	82	83.0	**79**	**79.9**
c-fat200-1	200	1 534	N/A	N/A	**18**	**18.0**	<0.01	**18**	**18.0**	**18**	**18.0**	**18**	**18.0**	**18**	**18.0**
DD_g140	560	2 710	N/A	N/A	164	167.6	<0.01	137	137.5	131	137.0	131	131.6	**118**	**118.7**
DD_g142	288	1 290	N/A	N/A	88	90.6	<0.01	73	73.5	72	73.3	70	71.0	**64**	**64.0**
DD_g143	414	2 088	N/A	N/A	112	116.4	<0.01	93	94.2	96	98.2	91	91.3	**83**	**83.9**
DD_g144	288	1 514	N/A	N/A	77	79.3	<0.01	60	60.2	60	60.0	58	58.0	**56**	**56.1**
DD_g145	404	2 320	N/A	N/A	96	99.7	<0.01	76	76.8	79	79.3	74	74.6	**70**	**70.8**
DD_g146	327	1 506	N/A	N/A	95	99.2	<0.01	77	79.0	78	79.4	76	76.5	**69**	**69.8**
delaunay_n11	2048	6 127	N/A	N/A	515	520.7	0.01	458	459.0	394	395.6	387	388.0	**333**	**335.4**
delaunay_n12	4 096	12 264	N/A	N/A	1 034	1 043.5	0.03	965	969.0	801	803.7	786	791.3	**681**	**683.3**
delaunay_n13	8 192	24 547	N/A	N/A	2 062	2 090.3	0.13	2003	2 004.0	1 596	1 751.6	1 635	1 721.5	**1 369**	**1 375.0**
DSJC500-5	500	125 248	N/A	N/A	8	8.8	<0.01	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
er_graph_1k_6k	1 000	6 000	N/A	N/A	198	213.1	<0.01	153	158.0	127	127.3	118	118.6	**108**	**109.4**
socfb-Haverford76	1 446	59 589	N/A	N/A	190	206.2	0.01	115	116.2	101	103.3	85	89.0	**58**	**58.5**
str_0	363	2 454	N/A	N/A	104	127.7	<0.01	59	60.0	58	58.7	57	57.0	**55**	**55.0**
str_200	363	3 068	N/A	N/A	102	110.8	<0.01	52	53.2	50	50.7	48	48.3	**47**	**47.0**
str_400	363	3 157	N/A	N/A	106	113.4	<0.01	53	53.8	51	51.0	48	48.0	**46**	**46.0**
str_600	363	3 279	N/A	N/A	98	108.2	<0.01	44	46.5	44	45.1	42	42.3	**41**	**41.0**
SW-1000-6-0d3-trial3	1 000	3 000	N/A	N/A	267	278.7	<0.01	227	227.5	197	197.9	183	183.4	**168**	**168.6**
SW-100-6-0d3-trial3	100	300	N/A	N/A	26	27.3	<0.01	**16**	**16.0**	**16**	**16.0**	**16**	**16.0**	**16**	**16.0**
t3dl_a	20 360	265 113	N/A	N/A	2 219	2 262.0	1.12	2 089	2 108.5	1 368	1 884.0	1 415	1 636.0	**1 044**	**1 049.4**
tube1	21 498	459 277	N/A	N/A	1 007	1 037.8	1	939	944.2	743	755.5	689	690.2	**642**	**649.4**
vibrobox	12 328	177 578	N/A	N/A	1 362	1 382.8	0.41	1 296	1 310.6	964	1 043.0	971	1 004.0	**779**	**785.4**
最小值			—	—	8	8.8	0.00	5	5.0	5	5.0	5	5.0	5	5.0
最大值			—	—	2 219	2 262.0	1.12	2 089	2 108.5	1 596	1 884.0	1 635	1 721.5	1 369	1 375.0
平均值			—	—	430.63	441.98	0.11	383.92	386.75	298.29	332.86	296.08	310.98	248.75	250.28
标准差			—	—	635.11	643.97	0.30	615.71	619.06	451.73	534.26	460.32	497.77	363.43	365.44

注: 加粗数值是对应算法找到的最优值

4.5 在 LPNMR'09 实例上的实验结果

表 2 展示了 CPLEX 求解器和其他高效的局部搜索算法在 LPNMR'09 实例上的实验结果. 表中"$|V|$"列表示图 $G(V, E)$ 的顶点数, 其他行列表示的含义与表 1 相同. 在该组实例中, CPLEX 求解器仅能求得 2 个实例的可行解, 且无法证明其最优性, 而对于其他 7 个实例, CPLEX 无法在给定的时间内找到可行解. 对比 5 个启发式算法可发现, 算法 MA、GRASP、GRASP_impLS 和 FPLS 能够在所有实例上找到相同的最小解和平均解. 然而 MWCDS 算法无法匹配其中任何一个实例的结果. 此外, 除了 MWCDS 算法, 其余 4 个算法在 LPNMR'09 实例上所得到的"最小值""最大值""平均值"和"标准差"结果均相等.

4.6 在一般 UDG 实例上的实验结果

表 3 展示了 CPLEX 求解器和其他算法在一般 UDG 实例上的实验结果, 其中每列和每行的含义与表 1 中相同. 对于该组实例, CPLEX 求解器未能找到任何一个最优解或可行解, 原因在于该组实例规模相对较大. 其他 5 个启发式算法, 即 MWCDS、MA、GRASP、GRASP_impLS 和 FPLS, 分别在 0、22、12、27 和 41 个实例上找到了最小解. 此外, 这些算法获得的平均解与 FPLS 算法的平均解相匹配的实例数量分别为 0、18、10 和 23.

与此同时, FPLS 算法在这组实例上得到的"平均值"和"标准差"结果均优于其他算法. 实验结果表明, 本文提

出的算法在效果上更为出色, 并且随着问题规模的增加, 所提出算法在求解能力上的优势变得更加明显.

4.7 在 NDR 实例上的实验结果

表 4 展示了 CPLEX 求解器和其他算法在 NDR 实例上的实验结果, 其中每列和每行的含义与表 2 相同. 对于该组实例, 精确求解器 CPLEX 在 24 个实例上无法获得任何最优解或可行解. 对于启发式算法来说, MWCDS 算法所得到的最小解和平均解不及其他 4 个对比算法. 在实例“c-fat200-1”“DSJC500-5”“SW-100-6-0-d3-trial3”和“ba_1k_30k”上, GRASP、GRASP_impLS 和 FPLS 算法可以找到相同的最小解, MA 算法能在其中的 3 个实例上找到最小解. 在其他 20 个实例上, FPLS 算法能够找到优于其他对比算法的最小解. 在平均解方面, FPLS 算法也明显优于 MWCDS、MA、GRASP 和 GRASP_impLS 算法.

最后, FPLS 算法在这组实例上求得的“最大值”“平均值”和“标准差”结果均较优. 总体来说, 本文提出的算法在该组实例上具有较好的效果. 同时, 随着原始图中顶点数量的增加, 本文提出的算法改进效果越好, 尤其是在“delaunay_n11”“delaunay_n12”“delaunay_n13”“t3dl_a”“tube1”和“vibrobox”这 6 个实例上.

综合这 4 组实例的结果, 本文提出的算法在与其他算法的比较中表现出色. 不仅在小图实例中找到了已知的最小解, 而且在大图实例中找到了更小的解. 因此, 本文提出的算法对于求解最小弱连通支配集问题具有很好的效果.

5 讨 论

5.1 消融实验

为了解决局部搜索过程中的循环问题, 本文提出了基于年龄属性、双层格局检测策略和禁忌策略的避免循环方法. 为使当前解能够有效地跳出局部最优, 本文提出了扰动策略. 为了利用上一次迭代的搜索信息来指导搜索, 使得算法能从一个较优的初始解开始进行搜索, 本文提出了反馈机制. 基于以上策略提出了高效的局部搜索算法求解最小弱连通支配集问题. 本节将对避免循环搜索方法、扰动策略和反馈机制的有效性进行验证. 本节选择规模分布较广的 NDR 基准测试实例进行测试. 对比算法在不同随机种子下运行 10 次, 内循环和外循环迭代次数均设为 1 000 次, 并将求解每个实例的截止时间设置为 300 s.

5.1.1 反馈机制的有效性

为了验证反馈机制的有效性, 本文对 FPLS 算法进行了改写, 得到了算法 APLS. APLS 算法与 FPLS 算法的区别在于 APLS 算法中没有使用反馈机制, 仅使用了基于年龄属性、双层格局检测策略及禁忌策略的避免循环搜索方法和扰动策略. APLS 在构建初始解时, 不会考虑上一轮在局部搜索过程中顶点的反馈信息, 因此在本轮构建初始解时容易错过大概率在最优解中的顶点, 而 FPLS 在选择顶点时则考虑了此信息. 具体来说, APLS 首先随机选择一个 *Dscore* 值最大的顶点加到解中, 接着从当前解的二层邻居内随机选择一个 *Dscore* 值最大的顶点加到解中, 若当前解的 2 层邻居内有锁定顶点, 优先将锁定顶点加到解中, 而 FPLS 首先随机选择一个 *Dscore* 值最大的顶点加到解中, 接着从当前解的 2 层邻居内选择一个 *Dscore* 最大的顶点加到解中, 若有多个顶点满足相同条件, 优先选择 *add_frequency* 值较大的顶点, 若仍有多个顶点满足该条件时才会选择随机添加一个满足该条件的顶点, 若当前解的 2 层邻居内有锁定顶点, 优先将锁定顶点加到解中. 通过对比 FPLS 和 APLS 算法的求解效果, 可以评估反馈机制对算法性能的影响.

APLS 和 FPLS 算法的实验结果如表 5 所示. 在此表中, 每列和每行的含义与表 2 中的含义相同. 结果表明, 除“er_graph_1k_6k”“SW-1000-6-0d3-trial3”实例外, FPLS 在其余 22 个实例上都能找到更小的解或已知的最小解. 在平均解方面, 除“er_graph_1k_6k”“SW-1000-6-0d3-trial3”和“tube1”实例外, FPLS 在其余 21 个实例上能找到更小的平均解或已知的最小平均解. 因此可以得出结论, 反馈机制是算法设计中的一项关键优化策略, 它通过历史信息的累积和应用, 为算法提供了智能决策的基础, 减少了搜索成本, 加速了收敛过程, 增强了算法对不同实例的适应性. 在最小弱连通支配集问题的求解中, 反馈机制展现了其在提升算法性能、效率和解质量方面的巨大潜力, 是算法创新和优化的重要方向.

表 5 反馈机制有效性验证实验结果

实例	|V|	|E|	APLS		FPLS	
			Best	Avg	Best	Avg
ba_1k_30k	1 000	30 039	**21**	**21.0**	**21**	**21.0**
ba_1k_6k	1 000	5 964	80	80.0	**79**	**79.9**
c-fat200-1	200	1 534	**18**	**18.0**	**18**	**18.0**
DD_g140	560	2 710	**118**	118.9	**118**	**118.7**
DD_g142	288	1 290	**64**	**64.0**	**64**	**64.0**
DD_g143	414	2 088	**83**	84.7	**83**	**83.9**
DD_g144	288	1 514	**56**	56.5	**56**	**56.1**
DD_g145	404	2 320	71	71.1	**70**	**70.8**
DD_g146	327	1 506	**69**	69.9	**69**	**69.8**
delaunay_n11	2 048	6 127	**333**	337.1	**333**	**335.4**
delaunay_n12	4 096	12 264	683	687.2	**681**	**683.3**
delaunay_n13	8 192	24 547	1 376	1 381.8	**1 369**	**1 375.0**
DSJC500-5	500	125 248	**5**	**5.0**	**5**	**5.0**
er_graph_1k_6k	1 000	6 000	**107**	**108.9**	108	109.4
socfb-Haverford76	1 446	59 589	**58**	**58.5**	**58**	**58.5**
str_0	363	2 454	**55**	55.2	**55**	**55.0**
str_200	363	3 068	**47**	**47.0**	**47**	**47.0**
str_400	363	3 157	**46**	**46.0**	**46**	**46.0**
str_600	363	3 279	**41**	**41.0**	**41**	**41.0**
SW-1000-6-0d3-trial3	1 000	3 000	**166**	**168.1**	168	168.6
SW-100-6-0d3-trial3	100	300	**16**	**16.0**	**16**	**16.0**
t3dl_a	20 360	265 113	1 052	1 057.3	**1 044**	**1 049.4**
tube1	21 498	459 277	644	648.5	**642**	649.4
vibrobox	12 328	177 578	780	786.1	**779**	**785.4**
最小值			5	5.0	5	5.0
最大值			1 376	1 381.8	1 369	1 375.0
平均值			249.54	251.16	248.75	250.28
标准差			365.31	367.21	363.43	365.44

注: 加粗数值是对应算法找到的最优值

5.1.2 扰动策略的有效性

为了验证扰动策略的有效性, 对 APLS 算法进行了改写, 得到了算法 ALS. ALS 算法与 APLS 算法的区别在于 ALS 算法中没有使用扰动策略, 仅使用了基于年龄属性、双层格局检测策略及禁忌策略的避免循环搜索方法. 具体来说, APLS 在求解的过程中, 算法会记录每一次求得的最小解, 若本次得到的解比之前的解好, 那将没有改善解的迭代步数初始化为 1, 若本次所得到解没有比之前的解好, 那将没有改善解的迭代步数加 1. 当没有改善解的迭代步数达到了 50 时, 该算法采用了移出顶点规则 1 来翻转一些顶点, 从而扩大搜索空间. 而 ALS 在求解过程中, 可能会一直陷入局部最优的情况, 导致算法停滞并难以跳出. 通过对比 APLS 和 ALS 算法的求解效果, 可以评估扰动策略对算法性能的影响.

ALS 和 APLS 算法的实验结果如表 6 所示. 在此表中, 每列和每行的含义与表 2 中的含义相同. 从表 6 中可以看出, 除“DD_g145”和“tube1”实例外, APLS 在其余 22 个实例上都能找到更小的解或已知的最小解. 而在平均解方面, 除“DD_g140”“DD_g144”“DD_g145”和“tube1”实例外, APLS 在其余 20 个实例上都能找到更小的平均解或已知的最小平均解. 因此可以得出结论, 通过引入扰动策略, 当算法在迭代过程中陷入局部最优状态, 无法进一步提升解的质量时, 该策略能够有效地打破当前的搜索空间限制, 扩大搜索范围, 帮助算法探索更多的解空间, 使算法的全局搜索能力得以增强, 进而可能发现更优解. 在应用扰动策略的 APLS 算法与不使用扰动策略的对比算法

(ALS) 之间进行比较时, APLS 算法在多个经典基准测试实例上均表现出更好的性能. 这表明扰动策略提高了算法在寻找最小弱连通支配集问题上解的质量.

表 6 扰动策略有效性验证实验结果

实例	$\|V\|$	$\|E\|$	ALS		APLS	
			Best	Avg	Best	Avg
ba_1k_30k	1 000	30 039	**21**	**21.0**	**21**	**21.0**
ba_1k_6k	1 000	5 964	**80**	**80.0**	**80**	**80.0**
c-fat200-1	200	1 534	**18**	**18.0**	**18**	**18.0**
DD_g140	560	2 710	**118**	**118.6**	**118**	118.9
DD_g142	288	1 290	**64**	**64.0**	**64**	**64.0**
DD_g143	414	2 088	84	**84.7**	**83**	**84.7**
DD_g144	288	1 514	**56**	**56.1**	**56**	56.5
DD_g145	404	2 320	**70**	**70.9**	71	71.1
DD_g146	327	1 506	70	70.0	**69**	**69.9**
delaunay_n11	2 048	6 127	334	337.3	**333**	**337.1**
delaunay_n12	4 096	12 264	690	693.6	**683**	**687.2**
delaunay_n13	8 192	24 547	1 403	1 407.3	**1 376**	**1 381.8**
DSJC500-5	500	125 248	**5**	**5.0**	**5**	**5.0**
er_graph_1k_6k	1 000	6 000	109	109.6	**107**	**108.9**
socfb-Haverford76	1 446	59 589	**58**	**58.5**	**58**	**58.5**
str_0	363	2 454	**55**	55.3	**55**	**55.2**
str_200	363	3 068	**47**	**47.0**	**47**	**47.0**
str_400	363	3 157	**46**	**46.0**	**46**	**46.0**
str_600	363	3 279	**41**	**41.0**	**41**	**41.0**
SW-1000-6-0d3-trial3	1 000	3 000	167	168.8	**166**	**168.1**
SW-100-6-0d3-trial3	100	300	16	16.0	**16**	**16.0**
t3dl_a	20 360	265 113	1 171	1 182.4	**1 052**	**1 057.3**
tube1	21 498	459 277	**634**	**637.1**	644	648.5
vibrobox	12 328	177 578	790	794.9	**780**	**786.1**
最小值			5	5.0	5	5.0
最大值			1 403	1 407.3	1 376	1 381.8
平均值			256.13	257.63	249.54	251.16
标准差			380.60	382.81	365.31	367.21

注: 加粗数值是对应算法找到的最优值

5.1.3 避免循环方法的有效性

为了验证基于年龄属性, 双层格局检测策略与禁忌策略的避免循环方法的有效性, 对 ALS 算法进行了改写, 得到了算法 LS_Tabu 和 CC^{2}LS_Tabu. LS_Tabu 算法和 ALS 算法的区别在于 LS_Tabu 算法仅使用基于禁忌策略的方法来避免循环问题, CC^{2}LS_Tabu 算法使用基于禁忌策略和双层格局检测策略的方法来避免循环问题, 而在 ALS 算法中则结合了年龄属性, 双层格局检测策略与禁忌策略一起来减少循环问题. 具体而言, 在 ALS 算法中, 刚加入解的顶点被放置在禁忌列表中, 以防止其被移出解, 从而避免循环问题的发生. 同时, 每个顶点都具有一个格局值和年龄值, 在选择要加入解的顶点时, 考虑了这两个值, 若顶点的格局值为 1, 则允许将其加入解中, 否则不允许. 在选择加入或移出解的顶点时, ALS 算法会选择具有最大 *Dscore* 值或最小 *Nscore* 值的顶点, 若存在多个满足条件的顶点, 则选择年龄值最大的顶点. 通过对比 ALS、CC^{2}LS_Tabu 和 LS_Tabu 算法的求解效果, 可以评估年龄属性和双层格局检测策略结合禁忌策略对算法性能的影响.

LS_Tabu、CC^{2}LS_Tabu 和 ALS 算法的实验结果如表 7 所示. 在此表中, 每列和每行的含义与表 2 中的含义相同. 从表 7 中可以看出, 在实例“ba_1k_30k”“c-fat200-1”“DSJC500_5”和“SW-100-6-0d3-trial3”上, LS_Tabu、CC^{2}LS_Tabu 和 ALS 算法可以找到相同的最小解. 在其他 18 个实例上, ALS 算法能够找到优于其他两种算法的

最小解, 在实例“DD_g140”“DD_g143”“DD_g144”“DD_g145”“delaunay_n13”“SW-1000-6-0d3-trial3”和“tube1”上, CC^2LS_Tabu 算法能够找到优于 LS_Tabu 算法的最小解. 在平均解方面, ALS 算法也明显优于 LS 和 CC^2LS_Tabu 算法, CC^2LS_Tabu 算法也优于 LS_Tabu 算法. 因此可以得出结论, 年龄属性作为节点选择策略的一个关键组成部分, 在解决该问题时对于提高算法收敛速度和避免循环具有积极影响. 年龄属性通过给每个顶点分配一个年龄值来记录它在解中或不在解中的持续时间, 并据此优先选择年龄较大的顶点进行状态改变 (添加或删除). 这种策略有助于打破搜索过程中的循环现象, 即防止算法反复尝试相同的顶点操作, 从而提高了搜索效率, 使得算法能够更快地遍历不同的解空间区域.

表 7　避免循环策略有效性验证实验结果

实例	$\|V\|$	$\|E\|$	LS_Tabu		CC^2LS_Tabu		ALS	
			Best	Avg	Best	Avg	Best	Avg
ba_1k_30k	1 000	30 039	**21**	**21.0**	**21**	**21.0**	**21**	**21.0**
ba_1k_6k	1 000	5 964	81	81.9	81	81.9	**80**	**80.0**
c-fat200-1	200	1 534	**18**	**18.0**	**18**	**18.0**	**18**	**18.0**
DD_g140	560	2 710	126	126.0	121	121.9	**118**	**118.6**
DD_g142	288	1 290	66	66.9	66	66.8	**64**	**64.0**
DD_g143	414	2 088	85	86.3	**84**	84.8	**84**	**84.7**
DD_g144	288	1 514	57	57.1	**56**	56.9	**56**	**56.1**
DD_g145	404	2 320	73	73.8	71	72.0	**70**	**70.9**
DD_g146	327	1 506	73	73.7	73	73.9	**70**	**70.0**
delaunay_n11	2 048	6 127	357	358.5	357	357.5	**334**	**337.3**
delaunay_n12	4 096	12 264	747	748.9	750	750.6	**690**	**693.6**
delaunay_n13	8 192	24 547	1 484	1 485.8	1 464	1 466.1	**1 403**	**1 407.3**
DSJC500-5	500	125 248	**5**	**5.0**	**5**	**5.0**	**5**	**5.0**
er_graph_1k_6k	1 000	6 000	112	113.1	112	113.0	**109**	**109.6**
socfb-Haverford76	1 446	59 589	60	60.8	60	60.6	**58**	**58.5**
str_0	363	2 454	56	56.8	57	57.0	**55**	**55.3**
str_200	363	3 068	49	49.0	49	49.0	**47**	**47.0**
str_400	363	3 157	47	47.4	47	47.4	**46**	**46.0**
str_600	363	3 279	42	42.0	42	42.0	**41**	**41.0**
SW-1000-6-0d3-trial3	1 000	3 000	176	178.6	175	178.2	**167**	**168.8**
SW-100-6-0d3-trial3	100	300	**16**	**16.0**	**16**	**16.0**	**16**	**16.0**
t3dl_a	20 360	265 113	1 258	1 271.4	1 258	1 264.1	**1 171**	**1 182.4**
tube1	21 498	459 277	674	680.6	667	678.8	**634**	**637.1**
vibrobox	12 328	177 578	832	837.8	835	837.7	**790**	**794.9**
最小值			5	5	5	5.0	5	5.0
最大值			1 457	1 468.4	1 464	1 466.1	1 403	1 407.3
平均值			271.46	273.18	270.21	271.68	256.13	257.63
标准差			405.89	408.00	403.59	404.96	380.60	382.81

注: 加粗数值是对应算法找到的最优值

年龄得分置换节点的策略鼓励了搜索过程中更多样化的决策, 增加了跳出局部最优解的可能性, 因为较老的顶点如果被改变, 则更可能引入新的结构信息, 引导算法向全局最优解或者更好的次优解靠近. 因此, 该策略对算法收敛速度有正面促进作用, 尤其是在大规模、复杂网络结构的问题实例上, 能有效缩短达到满意解所需的时间, 并且提高最终找到解的质量. 实验结果显示, 结合年龄属性、禁忌策略和双层格局检测策略的局部搜索算法, 相对于未使用这些策略的版本, 在多个基准测试实例上均表现出更快的收敛速度和更好的解质量.

5.2 对解质量的分析

根据第 4.3–4.6 节的实验结果, 在 4 组经典基准测试实例上, 本文提出的 FPLS 算法能够找到与比较算法相同

或更小的解. 对于小规模图而言, 所比较的算法也能够找到很好的解, 本文的算法在小图上的优势并不十分明显. 但在大规模图上, 本文的算法展现出了更加显著的优势.

为了更好地展示统计实验结果, 本文针对 4 种启发式算法 (MA、MWCDS、GRASP、GRASP_impLS) 与 FPLS 算法在同一例子上获得的最小解进行了比较. 在图 1 中, 纵轴表示对比算法找到的最小解与 FPLS 算法找到的最小解之间的比值, 记作 *BEST* (对比算法)/*BEST* (FPLS), 横轴表示实例序号. 其中, *BEST* (对比算法) 表示某个比较算法在某个实例上的最小值. 当纵轴的值大于 1 时, 表示 FPLS 算法得到的最小解更优. 当纵轴的值为 1 时, 表示 FPLS 算法与比较算法得到的最小解相同. 否则, 对比算法效果更好.

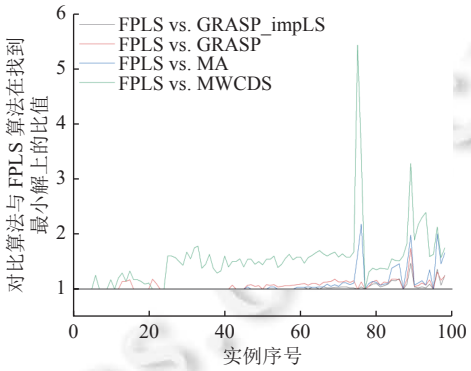

图 1　在 4 组经典基准测试实例上获得的最小值的比较

从图 1 中可以观察到, FPLS 算法在整体上表现出更好的性能, 该算法得到的最小解均优于其他比较算法得到的最小解. 这主要归因于在构造初始解阶段采用了锁定顶点的方法和反馈机制, 同时, 在局部搜索阶段通过扰动策略和一系列避免循环问题的策略来处理循环问题. 因此, 本文提出的算法在最小解的质量方面具备显著的优势.

5.3　对运行时间的分析

图 2 显示了 FPLS 在 4 组经典基准测试实例上的运行时间. 在每个实例上运行 10 次, 并计算在每个实例上获得最小解的平均时间. 很明显在 96 个实例上获得最小解的平均时间不超过 200 s. 对于前 60 个相对较小的实例, 算法 FPLS 可以非常快速地找到最小解. 对于规模较大的 38 个实例, FPLS 算法在大多数实例上能够在 100 s 内找到最小解, 少数实例所需的时间大于 100 s.

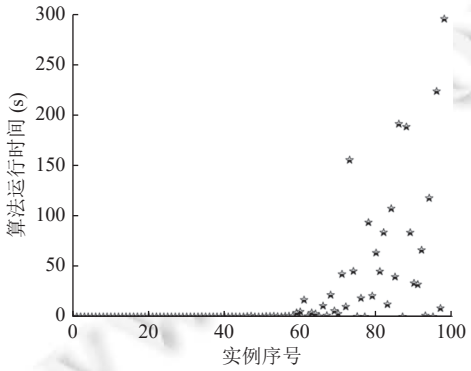

图 2　FPLS 算法在 4 组经典基准测试实例上的运行时间

图 3 显示了 FPLS 算法与对比算法在 4 组经典基准测试实例上的运行时间分布的比较. 由于算法 MWCDS 和 MA 都没有可执行代码, 本文只比较 FPLS 与 GRASP_impLS 和 GRASP 得到最小解的运行时间. 在图 3 中, 横轴和纵轴分别表示 FPLS 和对比算法得到最小解的平均时间. 可以看出, 大多数的图例分布在 X 的上方, 表明在同

一个例子上, FPLS 获得最小解的平均时间比 GRASP_impLS 和 GRASP 获得最小解的平均时间更短. 这可能是因为在构造初始解的过程中简化了原始图且选择较大概率存在于最优解中的顶点, 使其搜索空间更小, 其次, 本文提出的基于年龄属性, 两层格局检测策略与禁忌策略的避免循环方法、扰动策略、反馈机制, 使本文提出的算法效率更高.

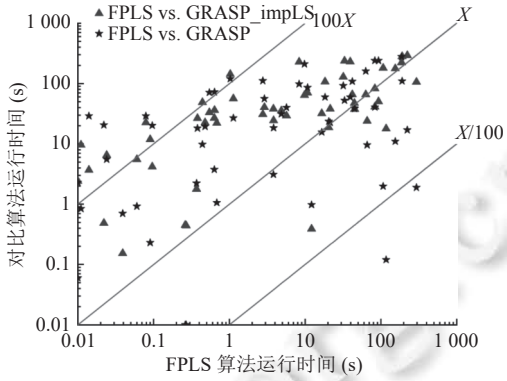

图 3 在 4 组经典基准测试实例上的运行时间分布的比较

6 结 论

本文提出了名为 FPLS 的新型局部搜索算法, 用于求解最小弱连通支配集问题. 在初始解构造阶段, 提出了优先选择锁定顶点的选择方法, 以简化无向连通图. 通过采用反馈机制、扰动策略、年龄属性、双层格局检测策略及禁忌策略等综合手段有效避免了循环搜索和陷入局部最优. 这些方法能够确保算法在面对不同的实例输入时具有较好的适应性和稳定性, 即使对于 NP 难问题也能寻找到高质量的解, 表明该算法具备良好的鲁棒性. 对于年龄属性的设计使得算法倾向于选择长时间未改变状态的顶点进行操作, 增加了解空间探索的多样性和可能性. 同时, 扰动策略能有效地跳出局部最优解, 进一步促进了不同解的生成. 基于上述策略和特性, 设计了多重启元启发式局部搜索算法 FPLS, 并将其与 CPLEX 求解器及其他 4 种先进的局部搜索算法进行了比较, 实验结果显示, 在多个基准测试实例上, 提出的 FPLS 算法相比其他对比算法能找到更小的解或已知最小解, 显示出解的质量更高. 具体而言, FPLS 算法能够找到随机 UDG 实例和 LPNMR'09 实例上已知的最小解, 在一般 UDG 实例上获得了 2 个最小解的新上界以及 9 个平均解的新上界, 在 NDR 实例上获得了 20 个最小解的新上界和 20 个平均解的新上界. 在求解时间方面, FPLS 算法在大部分基准测试实例上都能在较短的时间内 (平均不超过 200 s) 找到最小解, 尤其对于规模较小的实例更是表现出高效快速的特点. 即使是对于较大规模的实例, 多数情况下也能在合理的时间内 (通常在 100 s 内) 完成求解任务. 相较于 GRASP_impLS 和 GRASP 等对比算法, FPLS 在平均运行时间上表现更优, 说明该算法在时间效率方面有显著优势.

对于未来的研究工作, 本文提出的算法框架也能扩展到其他组合优化问题上, 例如最小加权连通支配集问题[26], 广义顶点覆盖问题[27], 最小总支配集问题[28]和最小支配树问题[29,30]. 同时, 相信在解决大规模图实例方面还存在改进空间. 因此, 将继续探索其他策略, 以进一步改进 FPLS 算法, 并解决更多大规模图实例的问题.

References:

[1] Dunbar JE, Grossman JW, Hattingh JH, Hedetniemi ST, McRae AA. On weakly connected domination in graphs. Discrete Mathematics, 1997, 167–168: 261–269. [doi: 10.1016/S0012-365X(96)00233-6]

[2] Du HJ, Wu WL, Shan S, Kim D, Lee W. Constructing weakly connected dominating set for secure clustering in distributed sensor network. Journal of Combinatorial Optimization, 2012, 23(2): 301–307. [doi: 10.1007/s10878-010-9358-y]

[3] Wang Y, Su R, Lv MS, Guan N. A multi-step estimation approach for optimal control strategies of interconnected systems with weakly connected topology. Automatica, 2023, 148: 110791. [doi: 10.1016/j.automatica.2022.110791]

[4] Mai VS, La RJ, Battou A. Optimal cybersecurity investments using SIS model: Weakly connected networks. In: Proc. of the 2022 IEEE Global Communications Conf. Rio de Janeiro: IEEE, 2022. 6097–6102. [doi: 10.1109/GLOBECOM48099.2022.10001358]

[5] Sou KC, Lu J. Relaxed connected dominating set problem for power system cyber-physical security. IEEE Trans. on Control of Network Systems, 2022, 9(4): 1780–1792. [doi: 10.1109/TCNS.2022.3165088]

[6] Probierz B, Hrabia A, Kozak J. A new method for graph-based representation of text in natural language processing. Electronics, 2023, 12(13): 2846. [doi: 10.1016/j.automatica.2022.11079]

[7] Han B, Jia WJ. Clustering wireless ad hoc networks with weakly connected dominating set. Journal of Parallel and Distributed Computing, 2007, 67(6): 727–737. [doi: 10.1016/j.jpdc.2007.03.001]

[8] Domke GS, Hattingh JH, Markus LR. On weakly connected domination in graphs II. Discrete Mathematics, 2005, 305(1–3): 112–122. [doi: 10.1016/j.disc.2005.10.006]

[9] Raczek J, Cyman J. Weakly connected roman domination in graphs. Discrete Applied Mathematics, 2019, 267: 151–159. [doi: 10.1016/j.dam.2019.05.002]

[10] Sandueta EP. Weakly connected total domination critical graphs. Advances and Applications in Discrete Mathematics, 2020, 25(2): 267–274. [doi: 10.17654/DM025020267]

[11] Chen YZP, Liestman AL. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In: Proc. of the 3rd ACM Int'l Symposium on Mobile Ad Hoc Networking & Computing. Lausanne: Association for Computing Machinery, 2002. 165–172. [doi: 10.1145/513800.51382]

[12] Dubhashi D, Mei A, Panconesi A, Radhakrishnan J, Srinivasan A. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. Journal of Computer and System Sciences, 2005, 71(4): 467–479. [doi: 10.1016/j.jcss.2005.04.002]

[13] Ding YH, Wang JZ, Srimani PK. A linear time self-stabilizing algorithm for minimal weakly connected dominating sets. Int'l Journal of Parallel Programming, 2016, 44(1): 151–162. [doi: 10.1007/s10766-014-0335-4]

[14] Torkestani JA, Meybodi MR. Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs. Int'l Journal of Uncertainty, Fuzziness and Knowledge-based Systems, 2010, 18(6): 721–758. [doi: 10.1142/S0218488510006775]

[15] Niu DD, Yin MH. A self-stabilizing memetic algorithm for minimum weakly connected dominating set problems. In: Proc. of the 2nd Int'l Workshop on Heuristic Search in Industry (HSI). 2022.

[16] Niu DD, Nie XL, Zhang LL, Zhang HM, Yin MH. A greedy randomized adaptive search procedure (grasp) for minimum weakly connected dominating set problem. Expert Systems with Applications, 2023, 215: 119338. [doi: 10.1016/j.eswa.2022.119338]

[17] Albuquerque M, Vidal T. An efficient matheuristic for the minimum-weight dominating set problem. Applied Soft Computing, 2018, 72: 527–538. [doi: 10.1016/j.asoc.2018.06.052]

[18] Li RZ, Hu SL, Liu H, Li RT, Ouyang DT, Yin MH. Multi-start local search algorithm for the minimum connected dominating set problems. Mathematics, 2019, 7(12): 1173. [doi: 10.3390/math7121173]

[19] Abu-Khzam FA. An improved exact algorithm for minimum dominating set in chordal graphs. Information Processing Letters, 2022, 174: 106206. [doi: 10.1016/j.ipl.2021.106206]

[20] Nakkala MR, Singh A, Rossi A. Swarm intelligence, exact and matheuristic approaches for minimum weight directed dominating set problem. Engineering Applications of Artificial Intelligence, 2022, 109: 104647. [doi: 10.1016/j.engappai.2021.104647]

[21] Glover F. Tabu search—Part I. ORSA Journal on Computing, 1989, 1(3): 190–206. [doi: 10.1287/ijoc.1.3.190]

[22] Cai SW, Su KL, Sattar A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artificial Intelligence, 2011, 175(9–10): 1672–1696. [doi: 10.1016/j.artint.2011.03.003]

[23] Li RZ, Hu SL, Zhang HC, Yin MH. An efficient local search framework for the minimum weighted vertex cover problem. Information Sciences, 2016, 372: 428–445. [doi: 10.1016/j.ins.2016.08.053]

[24] Zhou JP, Ren XL, Yin Q, Li RZ, Yin MH. Algorithm of strengthened configuration checking and clause weighting for solving the minimum satisfiability problem. Chinese Journal of Computers, 2018, 41(4): 745–759 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00745]

[25] Jovanovic R, Tuba M. Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Computer Science and Information Systems, 2013, 10(1): 133–149. [doi: 10.2298/CSIS110927038J]

[26] Li RZ, Wang YP, Liu H, Li RT, Hu SL, Yin MH. A restart local search algorithm with tabu method for the minimum weighted connected dominating set problem. Journal of the Operational Research Society, 2022, 73(9): 2090–2103. [doi: 10.1080/01605682.2021.1952117]

[27] Tai R, Ouyang DT, Li RZ, Zhang LM. ILSGVCP: An improved local search algorithm for generalized vertex cover problem. Journal of the Operational Research Society, 2023, 74(11): 2382–2390. [doi: 10.1080/01605682.2022.2147458]

[28] Hu SL, Liu H, Wang YP, Li RZ, Yin MH, Yang N. Towards efficient local search for the minimum total dominating set problem. Applied Intelligence, 2021, 51(12): 8753–8767. [doi: 10.1007/s10489-021-02305-6]

[29] Hu SL, Liu H, Wu XL, Li RZ, Zhou JP, Wang JN. A hybrid framework combining genetic algorithm with iterated local search for the dominating tree problem. Mathematics, 2019, 7(4): 359. [doi: 10.3390/math7040359]

[30] Niu DD, Liu B, Yin MH, Zhou YP. A new local search algorithm with greedy crossover restart for the dominating tree problem. Expert Systems with Applications, 2023, 229: 120353. [doi: 10.1016/j.eswa.2023.120353]

附中文参考文献:

[24] 周俊萍, 任雪亮, 殷茜, 李睿智, 殷明浩. 求解 MinSAT 问题的加强式格局检测与子句加权算法. 计算机学报, 2018, 41(4): 745–759. [doi: 10.11897/SP.J.1016.2018.00745]

李睿智(1989－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为算法设计与分析, 人工智能.

欧阳丹彤(1968－), 女, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为基于模型诊断, 自动推理, 人工智能.

何锦涛(1999－), 男, 硕士生, 主要研究领域为算法设计与分析, 人工智能.