中国科学: 信息科学 2016 年 第 46 卷 第 4 期: 445–460
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS

论 文

求解最小连通支配集问题的变深度邻域搜索算法

王灵敏[①], 周淘晴[②③], 吴歆韵[③*], 吕志鹏[③]

① 中国船舶工业系统工程研究院, 北京 100036
② 浙江农林大学信息工程学院计算机系, 杭州 311300
③ 华中科技大学计算机科学与技术学院智慧计算与优化实验室, 武汉 430074
* 通信作者. E-mail: xavier@hust.edu.cn

收稿日期: 2015–08–03; 接受日期: 2015–09–20; 网络出版日期: 2016–04–13
国家自然科学基金 (批准号: 61370183, 61100144) 和 2013 教育部新世纪优秀人才支持计划资助项目

摘要 本文提出了一种求解最小连通支配集问题的变深度邻域搜索 (VDNS) 算法. 结合最小连通支配集问题的特点, VDNS 算法采用了一种高效的邻域结构, 该邻域结构由一系列基础邻域动作组成, 合理地限制了搜索空间, 提高了算法的搜索效率. 同时, 本文还提出了两种提高算法搜索效率的方法: 修剪搜索分支以及增量评估更新技术. 用本文提出的 VDNS 算法对当前国际文献公开的共 91 个算例进行了测试, VDNS 算法能够在非常短的计算时间内改进其中 38 个算例, 优于此前国际文献中报道的最好结果, 表明了本文所提出的 VDNS 算法的有效性.

关键词 元启发式算法 变深度邻域搜索 邻域结构 最小连通支配集 增量更新

1 引言

本文主要研究求解最小连通支配集问题的高效算法. 由于其与移动自组织网络的密切联系, 最小连通支配集问题近年来受到越来越广泛的关注. 在移动自组织网络的建设中, 连通支配集在提高多播/广播路由效率上起了关键作用. 使用连通支配集可有效缓解移动自组织网络中的广播风暴问题 [1]. 对于支配集问题的研究, 最早可以追溯到雷达定位问题 [2], 以及文献 [3] 中介绍的特定网络通信问题. 现如今, 该问题的应用已渗透到技术创新的各个方面 [4,5], 例如市场管理 [6]、电力系统故障管理 [7]、网页图问题 [8]、传染性疾病的传播 [9,10] 以及利用社交网络缓解社会问题 [11] 等等, 都与连通支配集问题有较强的关联. 而与最小连通支配集问题直接相关的实际应用为无线传感器网络的规划问题 [12].

最小连通支配集问题与最多叶子生成树问题关系十分密切. 从此前的研究中得知, 这两个问题是等价的 [13]. 文献 [14] 证明了这两个问题都是 NP 难的. 目前, 国内外文献中已有大量关于最小连通支配集问题与最多叶子生成树问题的研究. 文献 [13] 介绍了用于求解最多叶子生成树问题的启发式算法. 文献 [15] 介绍了一种用于在单位圆图中求解最小连通支配集问题的近似算法. 文献 [16] 介绍了两个用于在常规图中求解连通支配集问题的近似算法. 该算法的基本思想为: 以图中度数最大的节点为根节点产生一棵生成树, 在这颗树中的非叶子节点的集合构成一个连通支配集, 该算法的近似比为

引用格式: 王灵敏, 周淘晴, 吴歆韵, 等. 求解最小连通支配集问题的变深度邻域搜索算法. 中国科学: 信息科学, 2016, 46: 445–460, doi: 10.1360/N112015-00128

$2(H(\Delta)+1)$. 文献 [17] 介绍了一种用于求解最小连通支配集问题的启发式算法, 该算法同样基于生成树算法. 文献 [18,19] 介绍的算法使用删除节点的方式来得到最小连通支配集. 文献 [20] 将文献 [16] 中的算法进行了扩展, 并融入了蚁群算法框架, 弥补了原始算法容易陷入局部极值的缺陷. 文献 [21] 介绍了求解最小连通支配集问题的分支定界算法, 该算法在文献 [22] 中得到了改进. 文献 [22] 提出的两种精确算法是目前已知的求解最小连通支配集问题的最好算法. 另一个与最小连通支配集问题较为相似的问题是顶点覆盖问题, 顶点覆盖问题的研究可参见文献 [23,24]. 其中文献 [23] 介绍了一种用于求解顶点覆盖问题的局部搜索算法, 该算法采用了一种称为边权策划的技术, 能够自适应地更新边的权重以避免陷入局部极值.

本文提出了一种基于变深度邻域搜索的算法. 本算法采用了一种适合最小连通支配集问题的邻域结构, 并给出了适用于此结构的增量更新评估方法. 基于该邻域结构, 本文提出的算法具有较高的求解效率. 在与前人算法的对比中, 本文给出的变深度邻域算法具有一定的竞争力, 改进了 38 个算例的当前最优解.

本文的组织结构如下: 第 2 节介绍最小连通支配集问题; 第 3 节介绍求解最小连通支配集问题的变深度邻域搜索算法; 第 4 节为实验与分析, 详细对比了本文提出的 VDNS 算法与前人的算法; 第 5 节为分析和讨论, 对本文提出的算法的特性进行了分析; 第 6 节对全文进行了总结.

2 问题模型

对于一个无向图 $G=(V,E)$, 其支配集 D^* 为满足如下条件的集合: $D^*\in V$, 且图中所有节点到 D^* 的距离不超过 1. 若一个支配集 D^*, 由其节点组成的子图 $G(D^*)=(D,E(D^*))$ 是连通图, 那么这个支配集称为一个连通支配集 (CDS). 最小连通支配集问题可描述为, 找到给定无向图中拥有节点数最少的连通支配集.

假设逻辑变量 y_i 为节点 i 的决策变量, 若节点 i 是支配集中的节点, 则 $y_i=1$, 否则 $y_i=0$. 定义集合 $\Gamma(i)$ 为包含节点 i 以及与节点 i 相邻的节点的集合. 那么最小连通支配集问题的模型如下:

$$\text{minimize}: \sum_{i\in V} y_i, \tag{1}$$

$$\sum_{j\in\Gamma(i)} y_j \geqslant 1, \quad \forall i\in V, \tag{2}$$

$$\text{connected}(y), \tag{3}$$

$$y_i\in\{0,1\}, \quad i\in V. \tag{4}$$

约束 (1) 为最小连通支配集问题的目标函数; 约束 (2) 保证每一个节点都被支配集所支配; 约束 (3) 保证支配集的连通性, 其具体内容参见文献 [22].

3 变深度邻域搜索算法

最小连通支配集问题是一个典型的 NP 难组合优化问题. 对于组合优化问题, 一种较为有效的方案是将其转化为一系列的判定问题来求解, 这两者在本质上是一致的. 在某些情况下, 判定问题更容易设计出有针对性的邻域结构. 例如, 目前求解顶点覆盖问题的最好的启发式算法都是基于这种方案, 可参见文献 [23,24].

本文所研究的最小连通支配集问题, 其对应的判定问题为给定大小的连通支配集问题, 即是否能在给定无向图中找到节点数为给定值的连通支配集. 对于一个最小连通支配集问题, 求解方法如下: 给定一个足够大的常数 k, 求解是否能够找到节点数为 k 的连通支配集 (k-CDS); 若能够找到节点数为 k 的连通支配集, 则将 k 值减小到 $k-1$, 并重新求解节点数为 k 的连通支配集; 重复上述过程直到满足终止条件. 算法总体框架见算法 1.

算法 1 算法主框架

```
Input G = (V, E)
使用贪心算法生成一个合法的 D*
令 k ← |D*| − 1
repeat
    产生一个初始 k-CDS 格局
    使用 VDNS 找到 k-CDS 可行解
    k ← k − 1
until 满足终止条件
return (k + 1)-CDS 可行解
```

继承前次迭代的可行解并去掉一个非关键节点 (非关键节点定义见第 3.3 小节), 通过这种方式可以得到每次迭代的初始格局. 对于第一次迭代, 可继承由贪心算法得到的解. 迭代的终止条件可设定为算法运行到达运行时间限制或 k 小于问题已知下界.

很明显能够看出, k 值越小, 其对应的连通支配集 (k-CDS) 问题越难求解. 本文的算法按照循序渐进的方式逐渐增加问题难度, 最终求解原始问题. 在以下章节中, 本文主要讨论如何求解固定 k 值的连通支配集问题 (k-CDS).

变深度邻域搜索 (VDNS) [25] 是一种可以对大邻域进行搜索并找到高质量局部最优值的超大规模邻域搜索 [26] 算法. VDNS 采用自适应的方式启发式地搜索一系列逐级加深的邻域空间 $N_1, N_2, \ldots, N_m$, 这里 m 指 VDNS 的搜索深度. 一个典型的 VDNS 邻域结构定义如下: 单变邻域结构 N_1, 仅一个变量发生改变的邻域结构; 双变邻域结构 N_2, 交换两个变量的邻域结构. 一般情况下, m 变邻域结构会改变 m 个变量的值. 为了减少搜索所用时间, VDNS 通常并不会搜索全部的邻域结构. VDNS 在多个组合优化问题上得到广泛运用, 例如旅行商问题 (TSP) [25]、带时间窗的车辆路由问题 [27]、广义分配问题 [28] 以及护士排班问题 [29] 等. 关于更多的 VDNS 算法描述, 可参见文献 [26].

3.1 初始化

VDNS 算法使用一个快速的贪心算法确定初始 k 值的大小, 并确定初始格局. 此初始解生成算法基于生成树算法, 其核心思想为用生长的方式构造图 G 的生成树 T, 具体描述如下所述.

首先选择图中度数最大的节点作为 T 的根节点. 在每次迭代中, 选择 T 中的一个节点 v, 将节点 v 所有不在 T 中的邻居节点加入 T; 最终 T 将成为图 G 的生成树. 选择 T 中所有非叶子节点组成初始连通支配集. 初始状态下图 G 中所有节点均处于未标记状态 (记为白色). 当考查某节点时 (将该节点标记为黑色), 将其所有不在 T 中的邻居节点加入 T (标记这些节点为灰色). 由此, 那些处于 T 中且未被考查过的节点为 T 的叶子节点 (灰色节点). 算法迭代考查灰色节点, 直到 G 中所有节点均被标记为黑色或灰色. 至此, 所有黑色节点的集合可认为是 G 的一个连通支配集.

3.2 搜索空间与评估函数

当为一个组合优化问题设计局部搜索算法时, 需要先定义其搜索空间. 在定义本文所介绍的VDNS 算法的搜索空间之前, 首先定义一些重要的符号.

D: 满足如下条件的点集合, $D \subseteq V$, 并且由 D 组成的图 $G(D) = (D, E(D))$ 是原图 G 的一个连通子图.

D^+: 被 D 所支配但不属于 D 的节点的集合, 形式化定义为

$$D^+ = \{j : j \notin D, \{i, j\} \in E, i \in D\}.$$

D^-: 没有被 D^+ 所支配的节点集, 形式化定义为

$$D^- = V - D - D^+.$$

很明显, 这 3 个集合构成了集合 V 的一个划分. 若 D^- 为空集, 那么可以认为 D 为图 G 的一个连通支配集. 本文所介绍的 VDNS 算法的基本思路便是在不改变图 $G(D)$ 的连通性以及 D 中节点个数的情况下, 通过改变 D 中的节点来使得 D^- 中节点个数降为 0, 使得 D 成为图 G 的连通支配集, 从而求解 k-CDS 问题.

在 VDNS 算法中, 搜索空间包含了点集 D 的所有情况. 例如, 对于一个 k-CDS 问题, VDNS 搜索空间的大小不超过 $C^k_{|V|}$. 在实际求解过程中, 算法可以通过一系列特殊的机制来缩小搜索空间. 在后续的章节中, 会详细介绍这些机制.

给定一个 k-CDS 问题的格局记为 X, 其可由 D, D^+ 以及 D^- 这 3 个集合唯一确定, 即 $X = (D, D^+, D^-)$. k-CDS 问题的评价函数可以定义为

$$f(X) = |D^-|,$$

即目标函数值为没有被 D 所支配的节点的个数. 因此, 当 $f(X) = 0$ 时, D 即为所求解问题的解.

3.3 邻域结构

在局部搜索中, 将一个操作 p 作用于某个格局 X, 可以将此格局变成其邻居格局, 记为 $X \oplus p$. 若 $P(X)$ 为所有格局 X 的可选操作集合, 则 X 的邻域结构可以定义为 $N(X) = \{X \oplus p : p \in P(X)\}$.

在变深度邻域搜索算法的设计中, 最重要的一环就是如何设计出一个足够简单, 并且经过组合可以形成足够复杂动作的基础操作. 在本文中, 定义交换为 VDNS 算法的基础操作. 所谓交换操作即从 D 中取出一个节点 n_i, 同时选取 D^+ 中的一个节点 n_j 加入到 D 中, 记为 $p = (n_i, n_j)$. 因此, 对于某个格局, 其基础邻域结构的大小不超过 $O(|D||D^+|)$. 由此, 可定义本文所述的 VDNS 的邻域结构如下:

$$N_1 = \{X \oplus p : p = (n_i, n_{i'}), n_i \in D, n_{i'} \in D^+\},$$

$$N_2 = \{X \oplus p : p = ((n_i, n_{i'}), (n_j, n_{j'})), n_i \in D, n_{i'} \in D^+, n_j \in D, n_{j'} \in D^+\},$$

$$N_3 = \{X \oplus p : p = ((n_i, n_{i'}), (n_j, n_{j'}), (n_s, n_{s'})), n_i \in D, n_{i'} \in D^+,$$
$$n_j \in D, n_{j'} \in D^+, n_s \in D, n_{s'} \in D^+\}.$$

其中 N_1 表示交换一对节点所得到的邻域解集合; N_2 表示连续交换两对节点所得到的邻域解集合; N_3 表示连续交换三对节点所得到的邻域解集合. 在每次迭代中 VDNS 算法优先搜索邻域 N_1; 若不能在

N_1 中找到改进操作, 则搜索邻域 N_2; 若在 N_2 中也不能找到改进操作, 则搜索邻域 N_3. 若 3 个邻域中均没有改进操作, 则随机选取一个使目标函数增加最少的操作. 对于 N_2 和 N_3, 应先交换第一对节点, 然后再交换第 2 对 (和第 3 对) 节点.

本文所述算法在搜索时保证 $G(D)$ 的连通性不发生改变. 也就是说, 操作 $p=(n_i,n_j)$ 中的节点应从 $G(D)$ 中的非关键节点 (记为 $\dot{D}$) 中选取. 因此, 在本算法中, 对于一个格局 X, 其基础操作的邻域大小为 $O(|\dot{D}||D^+|)$. 如果在图 G 中去掉一个节点后, 该图的连通分支数量增加, 则称该节点为 G 的关键节点, 关键节点又称作割点. 其中连通分支为图 G 中的一个极大连通子图. 所谓图 G 的非关键节点, 即为图 G 中除关键节点外的其他所有节点. 由于 n_i 不是 $G(D)$ 的关键节点, 那么从 D 中去除 n_i 不会影响 $G(D)$ 的连通性; 而 n_j 与 D 相邻, 那么在 D 中加入 n_j 也不会影响 $G(D)$ 的连通性. 若操作 (n_i,n_j) 中 n_i 是 n_j 连接到 D 的唯一节点, 那么不考虑这个操作. 因为此操作执行之后会破坏图 $G(D)$ 的连通性. 综上所述, 操作 $p=(n_i,n_j)$ 可以确保得到的 $G(D)$ 仍然是 G 的连通子图. 寻找关键节点的算法参见文献 [30], 关键节点的确定算法是一个线性时间算法, 其时间复杂度为 $O(|V|+|E|)$. 每次局部搜索迭代中, 只需执行一次该算法即可把当前格局中所有的关键节点找出来.

3.4 变深度邻域搜索算法

使用大邻域搜索通常能够得到更好的结果, 但是也会消耗更多的计算时间. 因此, 比较常见的做法是优先使用小邻域进行搜索, 当搜索遇到局部极值时扩大邻域范围继续搜索.

算法 2 描述了变深度邻域搜索算法的主要过程. 在每一次迭代中, 子过程 find_moves 根据搜索深度返回数个基础操作. 若找到的操作集能够改进当前格局, 则执行这些动作, 否则增加搜索深度直到达到最大深度限制. 在单次迭代中, 依次执行最优操作集中的各个操作. 在算法 2 中, M 表示算法能够搜索的最大深度, 在本文中 $M=3$.

算法 2 变深度邻域搜索

```
Input X
while f(X) > 0 do
    moves ← Φ
    best_delta ← ∞
    for depth = 1 to M do
        current_moves ← find_moves(depth)
        if Δf(current_moves) < 0 then
            moves ← current_moves
            best_delta ← Δf(current_moves)
            break
        else if Δf(current_moves) < best_delta then
            moves ← current_moves
            best_delta ← Δf(current_moves)
        end if
    end for
    for ∀(n_i, n_j) ∈ moves do
        设置 n_i 的禁忌步长
        执行操作 (n_i, n_j)
    end for
end while
return X
```

本算法引用了禁忌机制 [31], 在一定程度上保证了算法的疏散性. 禁忌算法被认为是一种高效的用于求解组合优化问题的算法策略 [32]. 禁忌搜索与普通局部搜索的差异在于: 在每次进行邻域操作之后, 将相应的变量设为禁忌状态, 并在后续的动作中不考虑被禁忌的变量. 通常被禁忌的变量在一定时长之后会被解禁, 这个时长称为禁忌步长. 禁忌机制通常使用禁忌表来实现, 禁忌表是一个用于记录每个禁忌对象的禁忌步长的数据结构. 禁忌机制保证了算法在一定跨度内不会重复访问某格局, 避免算法陷入局部极值. 针对 k-CDS 问题, 本文所述算法使用的禁忌策略为: 从 D 中移出的节点在一定迭代次数内不会被再次移入 D, 其中禁忌长度设置为 50.

算法 3 描述了本文所述 VDNS 算法的邻域评估算法 (即算法 2 中的 find_moves 子过程), 它返回一个由一连串基础动作组成的复合操作. find_moves 子过程使用递归的方式实现, 在每次递归中找出使目标函数值 $f(X)$ 下降最大的基础操作, 最后返回所有找到的基础操作集合. 它接受的参数 m 为当前搜索的最大深度. 算法 3 中第 5 行, 子过程 compare_moves 管理可选操作集 candidate_moves, 保证存在此集合中的元素为拥有最小 Δf 值的操作. 如果此次递归深度小于最大递归深度, 算法会分别执行这些操作, 并找出下一个深度的最优操作. 从下层递归中返回时, 需要回退之前所作的操作, 以保证格局的一致性. 若当前递归已达到深度限制, 那么从集合 candidate_moves 中随机选取一个元素返回到上层.

算法 3 邻域评估 find_moves

```
1: Input m
2: candidate_moves ← Φ
3: best_chain ← Φ
4: for ∀n_i ∈ Ḋ and ∀n_j ∈ D^+ do
5:     compare_moves(candidate_moves, (n_i, n_j))
6: end for
7: if m > 1 then
8:     for (n_i, n_j) ∈ candidate_moves do
9:         执行操作 (n_i, n_j)
10:        moves ← {(n_i, n_j)} ∪ find_moves(m − 1)
11:        执行操作 (n_j, n_i)
12:        if Δf(best_chain) > Δf(moves) then
13:            best_chain ← moves
14:        end if
15:    end for
16: else
17:    best_chain ← 从 candidate_moves 中随机选取一个元素
18: end if
19: return best_chain
```

为了使算法的搜索过程更加快速, 可以使用一些技巧来缩小算法的搜索空间. 首先, 限定集合 candidate_moves 的大小不超过 3, 若拥有最小 Δf 值的操作多于 3 个, 算法随机保留其中 3 个进入下层递归搜索. 其次, 在搜索中引入了剪枝策略, 若当前分支的动作 Δf 值之和大于 2, 则对当前分支不再继续搜索. 通常情况下, 一个基础操作的 Δf 值在 -1 到 1 之间, 若当前 Δf 的总和超过 2, 根据实验发现, 很难在后续的搜索中找到将 Δf 转变为负值的操作, 因此不再搜索 Δf 总值超过 2 的分支.

3.5 增量评估方法

对于邻域搜索算法, 是否能够快速地评估邻域解的质量是至关重要的. 为了达到这个目的, 算法中设计了一套增量更新机制来快速地确定每一个基础操作的 Δf 值. 本文所述算法通过管理表 L 来达到快速更新的目的. 表中的元素 $L[i]$ $(1 \leqslant i \leqslant n)$ 记录了 D 中连接到节点 i 的节点的个数,

$$L[i] = |j : \{i, j\} \in E, j \in D|.$$

很容易看出, 若 $L[i] = 0$ 则说明节点 i 没有被 D 所支配, 即 $i \in D^-$. 那么当前格局的目标函数值只需要通过统计表 L 中值为 0 的元素的个数即可求得. 实际上, 由于每个动作仅影响部分节点, 并不会使表 L 整体发生改变, 所以并不需要对整个表 L 进行扫描来得到目标函数值的变化量. 对于某基础动作 (n_i, n_j), 其对于整体目标函数值的改变量可以按照如下方式快速计算:

$$\Delta f(X)_{(n_i,n_j)} = |\{n_p : n_p \in C_{n_i}, L[n_p] = 1\}| - |\{n_q : n_q \in C_{n_j}, L[n_q] = 0\}|,$$

其中, C_{n_i} 与 C_{n_j} 分别表示节点 n_i 和 n_j 的邻居节点集. 若 n_i 的某邻居节点 n_p 在表 L 中的值为 1, 说明其与 D 的连接点只有 n_i. 若将 n_i 移出 D, 则 n_p 必将不再被 D 所支配. 若 n_j 的邻居节点 n_q 在表 L 中的值为 0, 说明其暂时不被 D 所支配. 若将 n_j 移入 D, 则 n_q 必将处于被支配的状态. 而此过程中其他节点与 D 的连接状态并不发生变化, 所以目标函数的变化值即为这两类节点个数的差值.

当执行某个操作之后, 表 L 的值也会发生改变. 同样地, 并不需要将表 L 进行整体的重新计算, 而是对其中个别相关元素进行更新. 计算方法如下:

对于所有 n_i 的邻居节点 n_p, 执行 $L[n_p] \leftarrow L[n_p] - 1$;

对于所有 n_j 的邻居节点 n_q, 执行 $L[n_q] \leftarrow L[n_q] + 1$.

由此, 对操作 (n_i, n_j) 的评估以及对动作执行后对表 L 更新的算法的时间复杂度均为 $O(|C_{n_i}| + |C_{n_j}|)$.

需要注意的是, 表 L 在算法的最开始时是需要整体计算初始值的, 在之后的局部搜索中只需要使用上述方法更新即可.

4 算例与实验结果

使用文献 [20, 22] 中的 3 组公共算例对本文提出的 VDNS 算法进行了测试, 这些算例的规模 $|V|$ 在 $30 \sim 400$ 之间, 是较大规模的最小连通支配集问题算例. 将本文所介绍的 VDNS 算法与文献 [20] 中的两种蚁群算法以及文献 [22] 中提到的 6 个精确算法进行了对比. 除此之外, 还使用了随机生成的大规模算例, 对所提出的 VDNS 算法的伸缩性进行了测试. 本文所述 VDNS 算法使用 JAVA 语言实现, 测试运行的 PC 机运行环境为 Core i3 3.1 GHz CPU 和 4.0 GB RAM, 操作系统为 Windows 7, JVM 为 Oracle JRE 1.8. 采用随机种子对这些基准测试算例测试 20 次, 每次计算时间 300 s.

4.1 算法参数调节

本小节采用一组对比试验来确定 VDNS 算法中的两个重要参数.

- 邻域深度 M, VDNS 所搜索的邻域深度. 在本试验中, 测试了 3 个 M 值: 1, 3, 5. 当值设为 1 时, 表示算法仅使用单一交换动作作为邻域动作.
- 禁忌步长 tt, 动作执行后节点被禁忌的步长. 在本试验中, 测试了 3 个 tt 值: 10, 50, 100.

表 1 参数性能对比

Table 1 Parameter setting performance

Parameters		Average	
M	tt	Gap (%)	Time (s)
1	10	2.32	12
1	50	1.57	17
1	100	1.57	17
3	10	4.49	82
3	50	0.62	78
3	100	0.62	79
5	10	5.76	182
5	50	2.33	167
5	100	2.51	141

这两个参数都与算法疏散性相关 (算法的集中性由邻域搜索体现). 通常来说, 这两个参数取值越大, 算法的疏散性越好. 过大或者过小的疏散性都不利于问题的求解. 疏散性与集中性之间的平衡对于算法是至关重要的.

该实验中, 使用了文献 [22] 中节点个数为 200 的算例来进行参数测试. 这些算例属于较难的算例, 图密度从 5% ∼ 70%. 试验中记录 VDNS 算法能够在规定时间内达到的最优解, 以及达到最优解的最短时间. 每组算例对于参数 M 和 tt 的不同组合使用不同的随机种子各计算 10 次.

由表 1 可以看出, 搜索深度越深, 算法耗时越长. 在规定的时间内, 更深的搜索深度并不能帮助算法得到更好的解. 当搜索深度为 1 时, 算法求解速度最快, 但结果并不如深度为 3 的情况好. 由此可知, 变深度的邻域有助于算法跳出局部极值. 禁忌步长 50 与 100 的差距不大, 但从表 1 中可知, 过短的禁忌步长不足以使搜索跳出局部极值. 从表 1 中可以发现当 $M = 3$, $tt = 50$ 时, 算法能够给出相对较好的结果. 因此在以下的章节中, 使用这组参数设置进行实验.

4.2 LPNMR 算例测试结果

本节使用了第十届逻辑编程与非单调推理国际会议 (LPNMR) 上提出的基准算例对 VDNS 算法进行测试. 将本文所述算法与文献 [20] 中的两个蚁群算法进行了对比, 对比结果如表 2. 表 2 中第 2 列为各算例当前已知的最优解, 第 3 列为 VDNS 算法得出的解的平均值, 后两列为文献 [20] 中的两个蚁群算法得出的解的平均值. 括号中的数值为平均计算时间. 两个蚁群算法均使用 C# 语言实现, 测试平台为 Dell OptiPlex 755, Intel Core2 CPU E8500 3.16 GHz 3 GB RAM.

由表 2 可以看出, VDNS 算法的结果十分稳定. 对于所有的测试算例, VDNS 都能够在极短的时间内找出历史最优解. 按照文献 [20] 的描述, 两个蚁群算法虽然也能够算出所有的历史最优值, 但结果并不十分稳定, 且耗时相对较长. 由此可以看出, 在这些算例上, VDNS 算法相比文献中介绍的蚁群算法具有较明显的优势.

4.3 随机算例测试结果

第 2 组算例为文献 [20] 中作者随机生成的大规模算例. 算例生成方法如下: 在一个 $N \times N$ 的固定区域中随机选取一定数量的节点作为算例的图节点; 若两节点 i, j 之间的距离小于某常数 R, 则 $\{i, j\}$

表 2 LPNMR 算例的计算结果

Table 2 Result for the LPNMR instances

Instance	Best	VDNS	ACO	ACO + PCS
40*200	5	5.0(<1)	5.8(3)	5.3(4)
45*200	5	5.0(<1)	5.8(3)	5.5(4)
50*250(1)	8	8.0(<1)	8.1(4)	8.0(6)
50*250(2)	7	7.0(<1)	7.5(5)	7.1(6)
55*250	8	8.0(<1)	8.8(5)	8.3(7)
60*400	7	7.0(<1)	7.0(6)	7.0(9)
70*250	13	13.0(<1)	14.2(11)	13.9(13)
80*500	9	9.0(<1)	10.0(12)	9.8(16)
90*600	10	10.0(<1)	10.9(14)	10.6(17)

表 3 随机算例计算结果 I

Table 3 Result for the random instances I

Instance	LB	VDNS			ACO		ACO + PCS	
		Best (300 s)	Best (30 s)	Gap (%)	Best	Gap (%)	Best	Gap (%)
n400_80_r60	15	**18**	**18**	20	20	33.3	19	26.7
n400_80_r70	12	**14**	**14**	16.7	16	33.3	15	25
n400_80_r80	10	12	12	20	12	20	12	20
n400_80_r90	8	**10**	**10**	25	11	37.5	11	37.5
n400_80_r100	7	8	8	14.3	8	14.3	8	14.3
n400_80_r110	6	**7**	**7**	16.7	8	33.3	8	33.3
n400_80_r120	6	**6**	**6**	0	7	16.7	7	16.7
n600_100_r80	18	**21**	**21**	16.7	23	27.8	22	22.2
n600_100_r90	16	**19**	**19**	18.8	22	37.5	21	31.3
n600_100_r100	14	**16**	**16**	14.3	17	21.4	17	21.4
n600_100_r110	12	15	15	25	15	25	15	25
n600_100_r120	10	**13**	**13**	30	15	50	14	40
n700_200_r70	31	**39**	**39**	25.8	46	48.4	46	48.4
n700_200_r80	26	**33**	**33**	26.9	41	57.7	41	57.7
n700_200_r90	21	**27**	**27**	28.6	34	62.0	33	57.2
n700_200_r100	18	**23**	**23**	27.8	28	55.6	28	55.6
n700_200_r110	16	**20**	**20**	25	23	43.8	22	37.5
n700_200_r120	13	**17**	**17**	30.8	21	61.5	21	61.5

是算例图中的一条边. 该组随机算例共有 41 个, 节点数由 80 至 400 不等. 计算结果如表 3 和 4 所示. 其中 “LB” 代表该算例最优解的最小界限, 其计算方法参见文献 [22]. Gap 值的计算方式为 Gap =(最小值 − 下界)/下界 ×100%. “Best (300 s)” 列记录 VDNS 算法在 300s 内能够达到的最好结果, “Best (30 s)” 列记录 VDNS 算法在 30 s 内能够达到的最好结果, 其中粗体的结果表示相应算法取得的结果优于其他算法.

表 4 随机算例计算结果 II

Table 4 Result for the random instances II

Instance	LB	VDNS			ACO MMAS		ACO + PCS	
		Best (300 s)	Best (30 s)	Gap (%)	Best	Gap (%)	Best	Gap (%)
n1000_200_r100	31	**39**	**39**	25.8	46	48.4	46	48.4
n1000_200_r110	27	**34**	**34**	25.9	43	59.3	42	55.6
n1000_200_r120	24	**30**	**30**	25	37	54.2	37	54.2
n1000_200_r130	21	**26**	**26**	23.8	32	52.4	32	52.4
n1000_200_r140	18	**23**	**23**	27.8	30	66.7	29	61.1
n1000_200_r150	17	**21**	**21**	23.5	28	64.7	26	53.0
n1000_200_r160	15	**20**	**20**	33.3	24	60	25	66.7
n1500_250_r130	40	**49**	**49**	22.5	60	50	60	50
n1500_250_r140	36	**44**	**44**	22.2	53	47.2	52	44.4
n1500_250_r150	32	**41**	**41**	28.1	51	59.4	51	59.4
n1500_250_r160	29	**38**	**38**	31.0	47	62.0	45	55.2
n2000_300_r200	33	**42**	43	27.3	55	66.7	52	57.6
n2000_300_r210	30	**39**	**39**	30	51	70	50	66.7
n2000_300_r220	28	**36**	**36**	28.6	47	67.9	45	60.7
n2000_300_r230	26	**36**	**36**	38.5	44	69.2	44	69.2
n2500_350_r200	49	**61**	**61**	24.5	79	61.2	79	61.2
n2500_350_r210	45	**58**	**58**	28.9	75	66.7	74	64.4
n2500_350_r220	42	**56**	**56**	33.3	68	61.9	69	64.3
n2500_350_r230	39	**50**	**50**	28.2	66	69.2	66	69.2
n3000_400_r210	61	**76**	77	24.6	99	62.3	98	60.7
n3000_400_r220	56	**72**	73	28.6	88	57.1	91	62.5
n3000_400_r230	53	**67**	68	26.4	86	62.3	86	62.3
n3000_400_r240	49	**63**	65	28.6	82	67.3	80	63.3

由表 3 和 4 可以看到, 本文所述 VDNS 算法的结果全面超越了蚁群算法 ACO 和 ACO+PCS. 除 n400 _80_r80, n400_80_r100 和 n600_100_r110 这 3 个算例结果相同外, VDNS 算法对其他算例的结果都优于文献中蚁群算法的结果. 观察算例属性得知, 这 3 个算例属于规模相对较小的算例. 且随着算例规模增大, 蚁群算法结果与 VDNS 算法结果的差距也越来越大. 表中两种蚁群算法 ACO 和 ACO+PCS 的计算结果的数据来源于文献 [20]. 由于该文献没有报道各算例的具体计算时间, 且没有提供相应的算法源程序, 因此表中没有给出蚁群算法的计算时间. 但从计算结果可以看出, 在计算时间为 30 s 的情况下, VDNS 算法的结果仍明显优于蚁群算法所得结果. 从表 2 也可以看出, 对于小规模的算例, VDNS 算法找到最优解的速度要优于蚁群算法. 该计算结果表明, VDNS 算法是求解最小连通支配集问题的一种有效算法.

由表 3 可以看出, 对于较小规模的算例, VDNS 算法的计算结果在 30 s 和 300 s 两种计算时间内没有明显的区别. 而表 4 中的计算结果表明, 对于大规模算例, VDNS 算法可以在 300 s 的计算时间内得到比 30 s 的计算时间内更好的结果. 由此可知, VDNS 算法对于中小规模的算例能够在较短的计算时间内找到最优解或近似最优解; 而对于大规模的算例, VDNS 算法需要相对较长的计算时间.

表 5 VDNS 与精确算法对比结果 I

Table 5 Comparison with the exact algorithm I

Instance	LB	VDNS	SAHY	IPHY	SABE	IPBE	SABC	IPBC
v30_d10	15	**15(7)**	15(12)	15(8)	15(41)	15(24)	15(<1)	15(<1)
v30_d20	7	**7(<1)**	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)
v30_d30	4	**4(<1)**	4(<1)	4(<1)	4(<1)	4(<1)	4(<1)	4(<1)
v30_d50	3	**3(<1)**	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v30_d70	2	**2(<1)**	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v50_d5	31	**31(<1)**	31(22)	31(9)	*31(>3600)*	*31(>3600)*	31(<1)	31(<1)
v50_d10	12	**12(<1)**	12(2)	12(1)	12(12)	12(1)	12(<1)	12(<1)
v50_d20	7	**7(<1)**	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)	7(<1)
v50_d30	5	**5(<1)**	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)	5(<1)
v50_d50	3	**3(<1)**	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v50_d70	2	**2(<1)**	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v70_d5	27	**27(<1)**	*29(>3600)*	27(290)	*29(>3600)*	*29(>3600)*	27(1)	27(<1)
v70_d10	13	**13(<1)**	13(25)	13(1)	13(1)	13(1)	13(14)	13(5)
v70_d20	7	**7(<1)**	7(1)	7(<1)	7(<1)	7(<1)	7(2)	7(1)
v70_d30	5	**5(<1)**	5(<1)	5(<1)	5(<1)	5(<1)	5(1)	5(<1)
v70_d50	3	**3(<1)**	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)	3(<1)
v70_d70	2	**2(<1)**	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)	2(<1)
v100_d5	24	**24(18)**	24(970)	24(35)	24(2542)	24(1963)	24(38)	24(65)
v100_d10	13	**13(1)**	13(2)	13(1)	13(<1)	13(<1)	13(17)	13(35)
v100_d20	8	**8(1)**	8(6)	8(2)	8(1)	8(1)	8(205)	8(534)
v100_d30	6	**6(1)**	6(11)	6(4)	6(3)	6(2)	6(209)	6(275)
v100_d50	4	**4(1)**	4(3)	4(1)	4(1)	4(<1)	4(40)	4(27)
v100_d70	3	**3(1)**	3(1)	3(<1)	3(1)	3(<1)	3(12)	3(12)
v120_d5	25	**25(3)**	25(1118)	25(27)	25(3)	25(10)	25(705)	25(105)
v120_d10	13	**13(1)**	13(56)	13(18)	13(3)	13(3)	*13(>3600)*	*15(>3600)*
v120_d20	8	**8(2)**	8(16)	8(8)	8(5)	8(3)	8(828)	*8(>3600)*
v120_d30	6	**6(3)**	6(14)	6(7)	6(5)	6(4)	6(496)	6(1039)
v120_d50	4	**4(1)**	4(8)	4(4)	4(4)	4(2)	4(161)	4(153)
v120_d70	3	**3(1)**	3(2)	3(2)	3(2)	3(<1)	3(34)	3(26)
v150_d5	26	**26(15)**	*27(>3600)*	*26(>3600)*	26(1047)	26(259)	*27(>3600)*	*27(>3600)*
v150_d10	14	**14(17)**	14(651)	14(195)	14(51)	14(28)	*15(>3600)*	*15(>3600)*
v150_d20	9	**9(195)**	9(2117)	9(902)	9(366)	9(273)	*9(>3600)*	*9(>3600)*
v150_d30	6	**6(8)**	6(34)	6(24)	6(21)	6(11)	6(2077)	*6(>3600)*
v150_d50	4	**4(3)**	4(17)	4(10)	4(7)	4(5)	4(535)	4(342)
v150_d70	3	**3(3)**	3(5)	3(2)	3(4)	3(<1)	3(51)	3(45)
v200_d5	27	28(18)	*29(>3600)*	*28(>3600)*	*29(>3600)*	27(1658)	*29(>3600)*	*29(>3600)*
v200_d10	16	**16(247)**	*16(>3600)*	*16(>3600)*	*16(>3600)*	*16(>3600)*	*16(>3600)*	*16(>3600)*

表 5 续

Instance	LB	VDNS	SAHY	IPHY	SABE	IPBE	SABC	IPBC
v200_d20	9	**9(163)**	*9(>3600)*	*9(>3600)*	9(1687)	9(1943)	*9(>3600)*	*9(>3600)*
v200_d30	7	**7(59)**	*7(>3600)*	*7(>3600)*	7(3208)	7(1848)	*7(>3600)*	*7(>3600)*
v200_d50	4	**4(9)**	4(41)	4(28)	4(24)	4(19)	4(2247)	4(1279)
v200_d70	3	**3(2)**	3(9)	3(5)	3(10)	3(<1)	3(334)	3(261)

表 6 VDNS 与精确算法对比结果 III

Table 6 Comparison with the exact algorithm III

	VDNS	SAHY	IPHY	SABE	IBPE	SABC	IPBC
Succeed	40	35	36	37	38	33	31
Fail	1	6	5	4	3	8	10
Success rate	97.6%	85.4%	87.8%	90.2%	92.7%	80.5%	75.6%

4.4 与精确算法的对比

本节将 VDNS 算法与文献 [22] 中提出的 6 个精确算法进行对比. 这 6 个精确算法分别为独立分支限界算法 (SABC)、迭代探测分支限界算法 (IPBC)、独立分解算法 (SABE)、迭代探测分解算法 (IPBE)、独立混合算法 (SAHY) 以及迭代探测混合算法 (IPHY). 其中两个混合算法与两个分解算法是现有的求解最小连通支配集问题的最佳算法, 是该领域的里程碑算法. 6 个精确算法均使用 c 语言实现, 其实验平台为 Intel XEON E5405 2.0 GHz 8 GB RAM.

本节的对比试验使用了与文献 [22] 中相同的算例. 该组图密度从 5% ∼ 70%, 节点个数由 30 ∼ 200, 最早由文献 [13] 引入.

实验对比结果见表 5, 其中 "VDNS" 列表示由 VDNS 算法得到的平均结果, 后 6 列的数据来自文献 [22], 括号中的数字代表算法得到相应结果所需的平均计算时间. 粗体字代表结果与下界相同, 即算法算出该算例的最优解. 斜体表示得到结果的时间超过 1 h. 表 6 是表 5 的汇总, 显示了各算法在 1 h 内成功求出最优解的算例的个数.

可以看出, 除算例 v200_d5 外, VDNS 算法对其余 40 个算例均能找到最优解. 对比其他 6 个算法, VDNS 算法在 1 h 的计算时间内成功率达到 97.6%. 并且, 对于某些复杂算例, VDNS 算法求解速度更快. 这些数据表明, 本文所述 VDNS 算法是求解最小连通支配集求解问题的一种高效算法.

4.5 算法的伸缩性测试

本小节使用一些更大规模的算例图来检测 VDNS 算法的伸缩性. 在本试验中, 随机生成 5 个节点数为 1000, 两点连接概率从 1% 至 5% 的平面图[1)]. 实验结果见表 7. 由于采用的算例规模较大, 在本实验中, 将算法运行时间增长为 10 min, 其他运行环境不变.

由表 7 可以看出, 本文所述 VDNS 算法对于大规模算例也能够很好地进行求解. 从以上所有实验可以看出, VDNS 算法能够求解的算例规模比较广泛 ($|V| = 30 \sim 1000$), 且在计算大规模算例时, 也能保持较低的内存资源消耗. 由此说明, VDNS 算法的伸缩性是较好的.

1) 生成器可在 http://web.cs.ualberta.ca/ joe/Coloring/index.html 下载.

表 7 大规模算例计算结果

Table 7 Result for the large instances

Instance	Connection probability (%)	LB	VDNS	Memory (MB)
flat1000_1	1	100	142	54.2
flat1000_2	2	51	85	47.8
flat1000_3	3	34	61	77.8
flat1000_4	4	25	49	70.0
flat1000_5	5	21	42	68.2

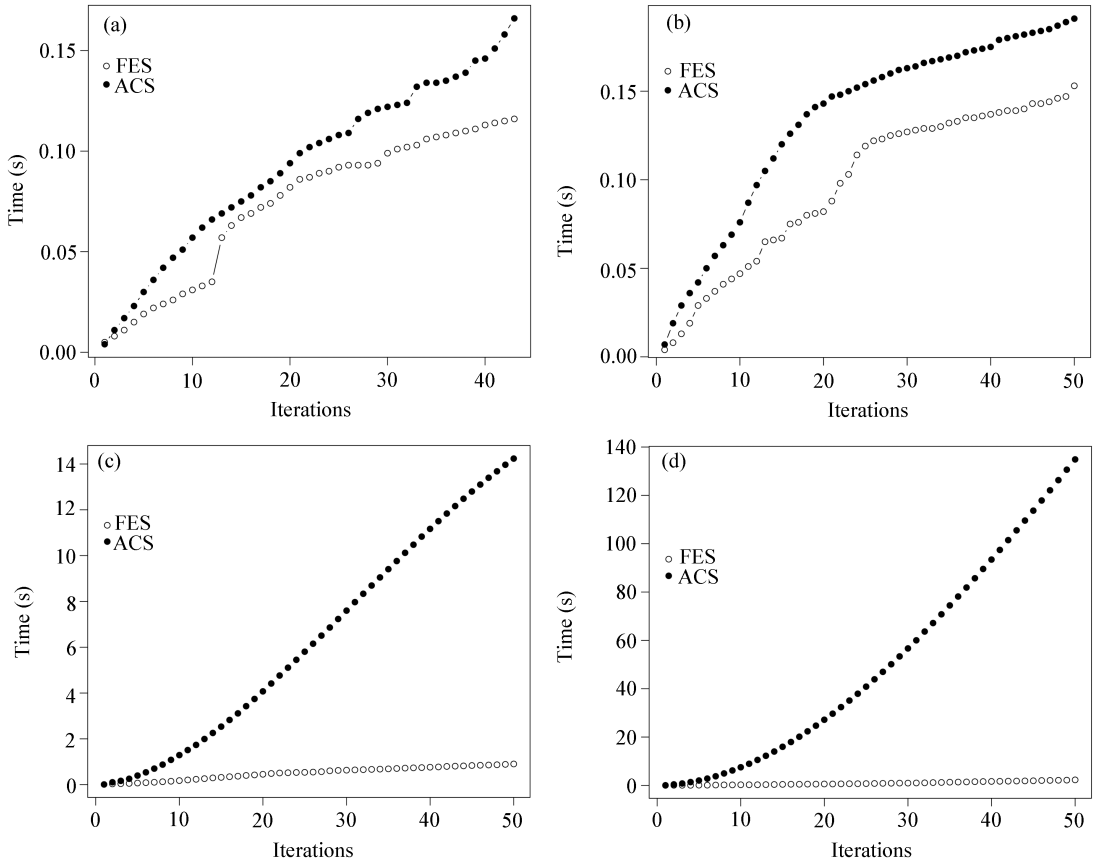


图 1 增量更新算法对比

Figure 1 Comparison between the evaluation methods. (a) 40*200; (b) 55*250; (c) n600_100_r90; (d) n1000_200_r140

5 分析与讨论

对于局部搜索算法, 如何进行有效的邻域评估是至关重要的. 一种好的邻域评估策略可以使算法在更短的时间内搜索到更多的邻域空间, 从而得到更好的结果. 在第 3.5 小节中, 本文介绍了一种增量评估方法, 该方法使用表 L 来快速计算某动作的目标函数值的改变量. 为了检验该策略是否有效, 设计了如下对比实验.

本试验对比了以下两个版本的 VDNS 算法: (1) 本文所述的使用快速增量更新策略的 VDNS 算

法 (FES); (2) 与本文所述 VDNS 算法求解思想相同, 但不使用快速增量更新方法而每次从头计算动作改变量的算法 (ACS). 对每一个算例分别使用这两种算法进行求解, 并记录每次迭代所使用的累计时长. 从两组算例中分别选取两个算例的对比结果进行分析, 见图 1. 值得注意的是, 其他算例也能得到类似的结果.

由图 1 可以看出, 采用增量更新策略的算法在时间上具有明显的优势. 对于小规模的算例, 增量更新的时间优势并不特别明显. 但对于大规模算例, 增量更新策略节约的时间是非常显著的, 并且算例规模越大, 增量更新效果越明显. 由此说明, 本文所采用的增量更新方法是有效的.

6 结论

本文提出了一种求解最小连通支配集问题的变深度邻域搜索算法 VDNS. 在 VDNS 算法中, 本文提出了一种新的邻域结构来更好地适应最小连通支配集问题. 这种邻域结构在扩大搜索空间的同时, 应用限制技术合理缩小了搜索范围, 提高了搜索效率, 减少了计算时间. 用本算法对当前文献中最小连通支配集问题的共 91 个公共算例进行了测试, 改进了其中 38 个算例的当前最优解, 其他 52 个算例与当前最优解持平. 在与其他文献中介绍的算法进行对比后发现, 本算法在计算时间上有明显优势. 上述结果表明了本文所提出的 VDNS 算法的有效性.

参考文献

1 Cheng X, Ding M, Du D H, et al. Virtual backbone construction in multihop ad hoc wireless networks. Wirel Commun Mobile Comput, 2006, 6: 183–190
2 Berge C, Minieka E. Graphs and Hypergraphs. Amsterdam: North-Holland publishing company, 1973. 175–181
3 Liu C L. Introduction to Combinatorial Mathematics. New York: McGraw-Hill, 1968
4 Rogers E M. Diffusion of Innovations. New York: Simon and Schuster, 2010. 23–31
5 Valente T W. Network Models of the Diffusion of Innovations. New York: Hampton Press Cresskill 1995. 163–164
6 Domingos P, Richardson M. Mining the network value of customers. In: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM, 2001. 57–66
7 Asavathiratham C, Roy S, Lesieutre B, et al. The influence model. IEEE Control Syst, 2001, 21: 52–64
8 Cooper C, Klasing R, Zito M. Lower bounds and algorithms for dominating sets in web graphs. Internet Math, 2005, 2: 275–300
9 Eubank S, Guclu H, Kumar V A, et al. Modelling disease outbreaks in realistic urban social networks. Nature, 2004, 429: 180–184
10 Stanley E A. Social networks and mathematical modeling. Connections, 2006, 27: 43–49
11 Wang F, Du H, Camacho E, et al. On positive influence dominating sets in social networks. Theor Comput Sci, 2011, 412: 265–269
12 Balasundaram B, Butenko S. Graph domination, coloring and cliques in telecommunications. In: Handbook of Optimization in Telecommunications. New York: Springer, 2006. 865–890
13 Lucena A, Maculan N, Simonetti L. Reformulations and solution algorithms for the maximum leaf spanning tree problem. Comput Manag Sci, 2010, 7: 289–311
14 Garey M R, Johnson D S. Computers and Intractability: a Guide to NP-Completeness. New York: WH Freeman, 1979
15 Marathe M V, Breu H, Hunt H B, et al. Simple heuristics for unit disk graphs. Networks, 1995, 25: 59–68
16 Guha S, Khuller S. Approximation algorithms for connected dominating sets. Algorithmica, 1998, 20: 274–387
17 Ruan L, Du H, Jia X, et al. A greedy approximation for minimum connected dominating sets. Theor Comput Sci, 2004, 329: 325–330

18 Butenko S, Cheng X, Oliveira C, et al. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. In: Recent Developments in Cooperative Control and Optimization. Berlin: Springer, 2004. 61–73
19 Butenko S, Oliveira C, Pardolos P. A new algorithm for the minimum connected dominating set problem on ad hoc wireless networks. In: Proceedings of CCCT'03 International Institute of Informatics and Systematics, Orlando, 2003. 39–44
20 Jovanovic R, Tuba M. Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Comput Sci Inf Syst, 2013, 10: 133–149
21 Simonetti L, da Cunha A S, Lucena A. The minimum connected dominating set problem: formulation, valid inequalities and a branch-and-cut algorithm. In: Network Optimization. Berlin: Springer, 2011. 162–169
22 Gendron B, Lucena A, Cunha A S, et al. Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. INFORMS J Comput, 2014, 26: 645–657
23 Cai S, Su K, Sattar A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif Intell, 2011, 175: 1672–1696
24 Cai S, Su K, Luo C, et al. NuMVC: an efficient local search algorithm for minimum vertex cover. J Artif Intell Res, 2013, 64 : 687–716
25 Lin S, Kernighan B W. An effective heuristic algorithm for the traveling-salesman problem. Oper Res, 1973, 21: 498–516
26 Pisinger D, Ropke S. Large neighborhood search. In: Handbook of Metaheuristics. New York: Springer, 2010. 399–419
27 Sontrop H, van Der Horn P, Uetz M. Hybrid Metaheuristics. Berlin: Springer, 2005. 78–89
28 Yagiura M, Ibaraki T, Glover F. A path relinking approach with ejection chains for the generalized assignment problem. Eur J Oper Res, 2006, 169: 548–569
29 Dowsland K A. Nurse scheduling with tabu search and strategic oscillation. Eur J Oper Res, 1998, 106: 393–407
30 Hopcroft J E, Tarjan R E. Algorithm 447: efficient algorithms for graph manipulation. Commun ACM, 1973, 16: 372–378
31 Glover F. Tabu search-part I. ORSA J Comput, 1989, 1: 190–206
32 Wang Z, Lv Z P, Ye T. Local search algorithms for large-scale load balancing problems in cloud computing. Sci Sin Inform, 2015, 45: 587–604 [王卓, 吕志鹏, 叶涛. 求解大规模云计算负载均衡问题的局部搜索算法. 中国科学: 信息科学, 2015, 45: 587–604]

Variable-depth neighborhood search algorithm for the minimum-connected dominating-set problem

Lingmin WANG[1], Taoqing ZHOU[2,3], Xinyun WU[3*] & Zhipeng LÜ[3]

1 *Systems Engineering Research Institute, Beijing* 100036*, China*;
2 *Department of Computer Science, School of Information Engineering, Zhejiang A & F University, Hangzhou,* 311300*, China*;
3 *Laboratory of Smart Computing and Optimization, Department of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan* 430074*, China*
*E-mail: xavier@hust.edu.cn

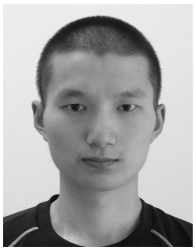
Abstract This paper presents a variable-depth neighborhood search (VDNS) algorithm for solving the minimum-connected dominating-set problem. By considering the problem structure of the minimum-connected dominating-set problem, this paper introduces an effective partition-based neighborhood structure, which consists of a series of basic neighborhood moves, restricts the search space to traverse towards more promising search regions, and generates better solutions during the search. This paper also presents two techniques to further improve the search efficiency of the algorithm: pruning the search branch, and the incremental evaluation technique. Applying the proposed VDNS algorithm to solve the 91 public instances used in the literature, VDNS outperforms the reference

algorithms in the literature by improving 38 of the best-known results, demonstrating the efficacy of the proposed VDNS algorithm.

Keywords meta-heuristics, variable-depth neighborhood search, neighborhood structure, minimum-connected dominating set, incremental evaluation

Lingmin WANG was born in 1979. She received a M.S. degree in computer science from Northwestern Polytechnical University, Xi'an in 2005. Currently, she is a senior engineer at the China State Shipbuilding System Engineering Research Institute in Beijing. Her research interests include system engineering, network planning, routing and scheduling.

Taoqing ZHOU was born in 1972. He received a master's degree in computer application technology from South-Central University for Nationalities, Wuhan in 2003. Currently, he is a doctoral candidate at Huazhong University of Science & Technology, as well as a senior lecturer at Zhejiang A&F University. His research interests include computational intelligence, solving large-scale combinatorial search problems, designing heuristic algorithms for practical applications, the traveling-salesman problem and the vehicle-routing problem.



Xinyun WU was born in 1987. He received a B.S. degree in computer science and technology from Naval University of Engineering, PLA, China in 2009. He is a Ph.D. candidate at the School of Computer Science & Technology of Huazhong University of Science and Technology. His research interests include computational intelligence, traffic grooming routing and the wavelength assignment algorithm in the WDM network, and the multi-commodity network design problem.

Zhipeng LÜ was born in 1979. He received a B.S. degree in applied mathematics from Jilin University, China in 2001 and a Ph.D. degree in Computer Software and Theory from Huazhong University of Science & Technology, China in 2007. He was a postdoctoral research fellow at LERIA, Department of Computer Science, University of Angers, France, from 2007 to 2011. He is a professor at the School of Computer Science and Technology of Huazhong University of Science and Technology and the Director of the Laboratory of Smart Computing and Optimization (SMART). His research interests include artificial intelligence, computational intelligence, green computing, and adaptive meta-heuristics for solving large-scale real-world and theoretical combinatorial optimization and constrained satisfaction problems.