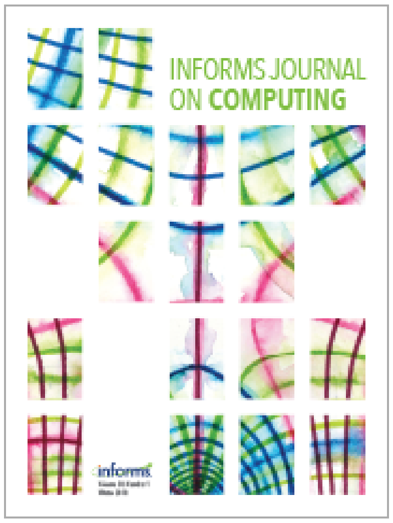


INFORMS Journal on Computing

A Fast Vertex Weighting-Based Local Search for Finding Minimum Connected Dominating Sets

Xinyun Wu, Zhipeng Lü, Fred Glover

To cite this article:
Xinyun Wu, Zhipeng Lü, Fred Glover (2022) A Fast Vertex Weighting-Based Local Search for Finding Minimum Connected Dominating Sets. INFORMS Journal on Computing 34(2):817-833. https://doi.org/10.1287/ijoc.2021.1106

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 34, No. 2, March–April 2022, pp. 817–833
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

A Fast Vertex Weighting-Based Local Search for Finding Minimum Connected Dominating Sets

Xinyun Wu,[a] Zhipeng Lü,[b,*] Fred Glover[c]

[a] School of Computer Science, Hubei University of Technology, Wuhan 430068, P.R. China; [b] SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P.R. China; [c] Department of Electrical, Computer and Energy Engineering, College of Engineering & Applied Science, University of Colorado Boulder, Boulder, Colorado 80309
* Corresponding author

Contact: xinyun@hbut.edu.cn, https://orcid.org/0000-0002-7525-0114 (XW); zhipeng.lv@hust.edu.cn, https://orcid.org/0000-0001-9185-3233 (ZL); fredwglover@yahoo.com, https://orcid.org/0000-0001-6945-0438 (FG)

Received: August 8, 2020
Revised: February 15, 2021; May 28, 2021
Accepted: June 2, 2021
Published Online in Articles in Advance: November 12, 2021

https://doi.org/10.1287/ijoc.2021.1106

Copyright: © 2021 INFORMS

Abstract. The minimum connected dominating set (MCDS) problem consists of selecting a minimum set of vertices from an undirected graph, such that each vertex not in this set is adjacent to at least one of the vertices in it, and the subgraph induced by this vertex set is connected. This paper presents a fast vertex weighting (FVW) algorithm for solving the MCDS problem, which integrates several distinguishing features, such as a vertex weighting-based local search with tabu and perturbation strategies to help the search to jump out of the local optima, as well as a search space reduction strategy to improve the search efficiency. Computational experiments on four sets of 112 commonly used public benchmark instances, as well as 15 newly introduced sparse instances, show that FVW is highly competitive compared with the state-of-the-art algorithms in the literature despite its simplicity. FVW improves the previous best-known results for 20 large public benchmark instances while matching the best-known results for all but 2 of the remaining ones. Several ingredients of FVW are investigated to demonstrate the importance of the proposed ideas and techniques.

Summary of Contribution: As a challenging classical NP-hard problem, the minimum connected dominating set (MCDS) problem has been studied for decades in the areas of both operations research and computer science, although there does not exist an exact polynomial algorithm for solving it. Thus, the new breakthrough on this classical NP-hard problem in terms of the computational results on classical benchmark instances is significant. This paper presents a new fast vertex weighting local search for solving the MCDS problem. Computational experiments on four sets of 112 commonly used public benchmark instances show that fast vertex weighting (FVW) is able to improve the previous best-known results for 20 large instances while matching the best-known results for all but 2 of the remaining instances. Several ingredients of FVW are also investigated to demonstrate the importance of the proposed ideas and techniques.

History: Accepted by Erwin Pesch, Area Editor for Heuristic Search & Approximation Algorithms.
Funding: The research was supported by the National Natural Science Foundation of China [Grants 61902116 and 72101094]. It was also supported by the Natural Science Foundation of Hubei Province [Grant 2020CFB798].

Keywords: metaheuristic • minimum dominating set • vertex weighting • local search • search space reduction

1. Introduction

The minimum connected dominating set (MCDS) problem is a classical combinatorial optimization problem that consists of selecting a minimum set X of vertices from an undirected graph $G = (V, E)$ such that each vertex not in X is adjacent to at least one vertex of X and the subgraph of G induced by X must be a connected graph.

The MCDS problem has also been shown to be equivalent to the maximum leaf spanning tree problem (MLSTP), another well-known classical problem (Fujie 2004). The MLSTP aims to find a spanning tree in an undirected simple graph such that the number of leaves of the tree is maximized. The vertices in the tree apart from the leaves provide a connected dominating set of the graph as a result of the connectivity of the tree. Therefore, maximizing the number of leaves of the spanning tree is equivalent to minimizing the size of the connected dominating set. In the following, we treat these two problems as one.

The MCDS problem is a variant of the minimum dominating set (MDS) problem arising in ad hoc networks (Balasundaram and Butenko 2006). It plays an important role in broadcast routing (Tseng et al. 2002, Wu and Dai 2003, Cheng et al. 2006), power management (Chen et al. 2002, Deb et al. 2003, Ding et al. 2003), and the regenerator location problem in fiber-optic networks (Chen et al. 2010). As a challenging NP-hard problem, the MCDS problem has attracted considerable attention from the academic community over the past few decades. Solution methods for the MCDS problem can be mainly categorized into exact algorithms, approximation algorithms, and heuristic/metaheuristic methods.

Exact algorithms proposed for the MCDS problem notably include the work of Simonetti et al. (2011), who presented an integer programming formulation and new valid inequalities accompanied by a branch-and-cut algorithm based on a reinforced formulation. This work was extended by Gendron et al. (2014), who proposed a decomposition algorithm for the MCDS. Buchanan et al. (2015) proposed three integer programming formulations based on vertex cuts for the k-connected d-dominating set problem. Their proposed lazy constraints branch-and-cut (LCBC) algorithm competes favorably with existing exact solution approaches for the MCDS problem (the case $k = d = 1$). In parallel, Ahn and Park (2015) also studied the k-connected d-dominating set problem, by proposing an integer programming formulation and an optimal algorithm. Other exact solution approaches based on MLSTP can be found in Fujie (2004), Gouveia and Simonetti (2017), and Reis et al. (2015). In addition, approximation algorithms, which provide a guaranteed approximation factor, have been introduced in Guha and Khuller (1998), Wan et al. (2004), Misra and Mandal (2010), and Khuller and Yang (2019).

Metaheuristics, the domain in which our present method lies, have been demonstrated to be highly effective for many hard real-world problems (Burke and Bykov 2016, Wang et al. 2016, Ghaznavi et al. 2017, Subramanyam et al. 2020). Various state-of-the-art heuristic and metaheuristic algorithms have also been proposed for tackling the MCDS. A few of the more significant entries include a greedy heuristic proposed by Butenko et al. (2004), which starts from a connecting dominated set (CDS) consisting of all vertices and gradually reduces the size of the current CDS by excluding some vertices; a simulated annealing and a tabu search algorithm proposed by Morgan and Grout (2007); and a competitive one-step ant colony algorithm proposed by Jovanovic and Tuba (2013). More recently, Wu et al. (2017) proposed a restricted swap-based neighborhood structure embedded in a tabu search framework (RSN-TS) for solving the MCDS problem, which outperforms the previous algorithms on the standard benchmark instances. Li et al. (2019) additionally proposed a multistart local search that produces results competitive with those obtained by the RSN-TS. Their algorithm adapts a configuration checking mechanism to avoid the cycling problem. Another recent innovation is an ant colony optimization algorithm (ACO-RVNS) proposed by Bouamama et al. (2019), which produces results comparable to those of RSN-TS on the previously tested instances while outperforming RSN-TS on some newly generated large-scale benchmark instances. In parallel with this paper, Li et al. (2020) presented a new MCDS algorithm specially designed for tackling massive graphs that outperforms previous algorithms. Their algorithm takes advantage of a hybrid dynamic connectivity maintenance method switching alternately between a novel fast connectivity maintenance method based on a spanning tree and its previous counterpart, therefore being very effective on massive graphs. Additional metaheuristic approaches for the MCDS have been presented in Li et al. (2017), Chinnasamy et al. (2019), and Hedar et al. (2019).

This paper presents a local search algorithm based on fast vertex weighting (FVW) for the MCDS problem by incorporating a vertex weighting technique and a search space reduction technique. Similar to the idea of RNS-TS from Wu et al. (2017), FVW transforms the MCDS problem into a series of subproblems with a fixed number of vertices in the dominating set. Specifically, it checks whether a CDS exists for each size k in a relevant range. In addition to this problem transformation mechanism, we introduce several new ideas to solve the MCDS problem. Our contributions can be summarized as follows:

1. A vertex weighting technique and an incremental neighborhood evaluation that enable FVW to achieve a better balance between exploitation and exploration
2. A search space reduction technique that further improves the performance of FVW and a perturbation operator that helps FVW to produce more stable solutions
3. Tests conducted on four sets of 112 most commonly used instances of the MCDS problem, disclosing that the FVW algorithm improves the best-known results on 20 instances and matches the best records for all but 2 remaining cases; these tests also show that our computational time is much shorter than that of the state-of-the-art algorithms in the literature for most tested instances
4. Tests on 15 newly introduced instances showing the high performance of the FVW on large sparse graphs; these instances also provide a benchmark for future comparison
5. Extensive experiments to analyze the contributions of the key components of our algorithm, which include the vertex weighting technique and the search space reduction technique
6. Two fundamental propositions that elucidate the reasons for the efficacy of the proposed neighborhood structure

The remainder of the paper is organized as follows. Section 2 describes the MCDS problem and our transformation method. Section 3 presents the FVW algorithm and detailed descriptions of its main components. Experimental results of the FVW algorithm on public benchmark instances are reported in Section 4. Finally, the analysis of several important features of the FVW algorithm is given in Section 5, followed by the conclusion in Section 6. In addition, the appendix presents the two fundamental propositions that elucidate the reasons for the efficacy of the proposed neighborhood structure.

2. Problem Description and Transformation

To formally describe the MCDS problem, we use the following notation: Variable $x_i \in \{0,1\}(i \in V)$ identifies whether vertex i is included in the dominating set, where $x_i = 1$ for inclusion and $x_i = 0$ otherwise. Variable $y_e \in \{0,1\}(e \in E)$ similarly identifies whether edge e is selected to ensure the dominating set is connected. The term Γ_i denotes the set of vertices that includes vertex i and all its neighbors, and D represents any subset of V. The MCDS problem can then be formally stated as follows:

$$\text{MCDS:} \quad \min \sum_{i\in V} x_i \tag{1}$$

$$\text{s.t.} \sum_{j\in\Gamma_i} x_j \geq 1 \quad i \in V, \tag{2}$$

$$\sum_{e\in E} y_e = \sum_{i\in V} x_i - 1, \tag{3}$$

$$\sum_{e\in E(D)} y_e \leq \textstyle\sum_{i\in D\setminus\{j\}} x_i \quad D \subset V, j \in D, \tag{4}$$

$$x_i, y_e \in \{0,1\} \quad i \in V, e \in E. \tag{5}$$

This formulation minimizes the number of vertices to be included in the dominating set while using variable y to select edges to guarantee that a spanning tree can be found in the subgraph induced by this set. Constraints (2) guarantee that each vertex connects at least one vertex in the selected set (i.e., the selected vertex set can dominate all the vertices). Constraints (3) and (4) guarantee that there exists a spanning tree in the selected dominating set (i.e., the dominating set is connected). For more details of this formulation, the reader is referred to Simonetti et al. (2011).

As previously mentioned, the proposed algorithm in this paper transforms the MCDS problem into a series of subproblems of finding the connected dominating set with a fixed size. The formulation of the subproblem can be obtained from the original MCDS model by adding the constraint

$$\sum_{i\in V} x_i = k \tag{6}$$

and replacing the objective function by

$$\min g(X) = \textstyle\sum_{i\in V} c_i x_i.$$

By this means, we can define the *k-CDS* problem as $\min g(X), \text{s.t.}: (2)\text{–}(6)$. Here, the form of the objective function g is inconsequential to make the formulation standard (i.e., the c_i coefficients can be any constants). The k-CDS problem would be infeasible if no CDS of size k exists for the graph G and any feasible solution produces a *yes* answer (i.e., it is a decision problem). We then transform the original MCDS problem into a series of k-CDS problems where $k \in \{1, 2, \ldots, |V|\}$. The k-CDS problem with a feasible solution with the minimum value of k is considered the best solution for the original MCDS problem.

We introduce an auxiliary optimization problem k-CC where the goal is to find a k-vertex connected component dominating as many vertices as possible in graph G. The solution of k-CC that dominates the whole set V is a feasible solution of k-CDS. This new auxiliary problem is obtained from k-CDS by relaxing Constraints (2) to minimize the number of nondominated vertices. For this purpose, we introduce another binary variable, z_i, to denote whether vertex i is dominated ($z_i = 1$ if vertex i is dominated) by the selected vertices (where $z_i = 1$ if vertex i is dominated and 0 otherwise). Thus, the following constraints must be satisfied:

$$0 \leq |\Gamma_i| \cdot z_i - \sum_{j\in\Gamma_i} x_j \leq |\Gamma_i| - 1 \quad i \in V, \tag{7}$$

$$z_i \in \{0,1\} \quad i \in V. \tag{8}$$

The number of vertices that do not connect any of the selected vertices can be calculated as

$$f(X) = \sum_{i\in V} (1 - z_i). \tag{9}$$

By minimizing the objective given by Equation (9), the original k-CDS problem can be transformed into the following auxiliary problem:

$$k\text{-CC}: \min\ f(X), \text{s.t.}: (3)\text{–}(8).$$

Note that a dominating set of size k exists only if the best solution for model k-CC produces a solution with objective value 0. In the following, k-CC is actually used to solve the k-CDS problem.

3. Fast Vertex Weighting-Based Local Search

The basic idea of our FVW is to tackle the MCDS problem based on a reduction to the decision version of the problem (the k-CDS problem). Specifically, in order to solve an MCDS instance for network G, our algorithm tries iteratively to find in G a CDS with size k (k-CC with objective value 0) for successively smaller values of k. If a solution with objective value 0 can be found for the k-CC, the algorithm decreases k to $k-1$. Otherwise, the algorithm stops, and $k+1$ is the best solution for the original MCDS problem. The pseudocode of the FVW is given in Algorithm 1.

Algorithm 1 (Main Framework of FVW)

```
Input: $G = (V, E)$
Output: A feasible CDS configuration $X_{\text{best}}$
1: procedure FVW($G$)
2:     $X_{\text{best}} \leftarrow V$
3:     repeat
4:         $X \leftarrow \textsc{Shrink}(X_{\text{best}})$
5:         $X \leftarrow$ LS_κCC(X) ▷ Section 3.2
6:         if $f(X) = 0$ then
7:             $X_{\text{best}} \leftarrow X$
8:         else
9:             return $X_{\text{best}}$
10:        end if
11:    until The termination condition is met
12:    return $X_{\text{best}}$
13: end procedure
```

FVW starts with the set of all vertices V (line 2). Because V is always a feasible CDS for a connected graph, this guarantees the algorithm will produce a feasible solution. In Algorithm 1, X_{best} records the minimum feasible CDS found so far. At each iteration, a nonarticulation vertex is selected and removed from X_{best} if removing it is the best choice according to the objective function (line 4). Thus, an initial k-CC configuration X is produced. The local search procedure LS_κCC starts from the initial configuration X in an attempt to find a feasible CDS with size $k = |X_{\text{best}}| - 1$. Then, LS_κCC tries to improve the current configuration, (i.e., minimizing the number of nondominated vertices) by trying to solve the model k-CC. If the LS_κCC procedure produces a feasible CDS with size k, X_{best} is replaced by the newly obtained CDS (line 7), and the process is repeated until the LS_κCC procedure fails to produce a feasible solution (line 9) or a predefined termination condition is met.

3.1. Vertex Weighting Technique

The vertex weighting technique of FVW helps the search to escape from the local optima by dynamically changing the objective function. It can be regarded as a variant of guided local search (Voudouris et al. 2010) and has been successfully applied to various challenging optimization problems, such as the unicost set cover problem (Gao et al. 2015), the minimum vertex cover problem (Cai et al. 2011), and the satisfiability problem (Luo et al. 2012).

In Wu et al. (2017) the objective function for a k-CDS problem is defined the same as the one in model k-CC, where the configuration X is considered as a feasible connected dominating set if $f(X) = 0$. In this paper, by contrast, FVW adopts another additional objective function:

$$f^*(X) = \sum_{i \in V} w_i(1 - z_i), \tag{10}$$

where w_i represents the weight for vertex i. Note that w_i varies as the search proceeds. If one vertex keeps failing to be dominated, we tag it as hard to be dominated and treat it with higher priority. Specifically, when the local search is trapped in a local optimal trap, the FVW algorithm increases the weight w_v by 1 for each nondominated vertex v with $z_v = 0$. This process changes the landscape of the solution space such that the previous local optimum is no longer the valley bottom, and the algorithm will be able to continue to explore other search areas. The more frequently a vertex fails to be dominated during the stagnating status, the greater its weight will be. On one hand, the vertex weighting technique is able to prevent vertices from repeatedly failing from being dominated and diversifies the search in an adaptive manner. On the other hand, it modifies the solution space in a smooth way and guides the search to more promising regions. Using frequency memory to place higher emphases on variables that repeatedly fail to achieve a feasible value is similarly proposed in parametric tabu search (Glover 2006).

3.2. Main Framework of LS_κCC for Solving the *k*-CC

In this section, we describe the subprocedure LS_κCC, which solves the k-CC problem formulated as k-CC with additional modified objective function f^*. It follows the common procedure of a local search algorithm by incorporating tabu and perturbation strategies. From an initial configuration, it gradually improves the current configuration X until X becomes a CDS for the graph G. Defined by the neighborhood structure, a move is a small alteration to the current configuration. At each iteration, a best nontabu move is chosen from the neighborhood and applied to the current configuration to produce a new neighboring configuration. Algorithm 2 describes the basic procedure of the subprocedure LS_κCC.

Algorithm 2 (LS_κCC Procedure)

```
Input: $X_0$
Output: A feasible $k$-CDS configuration $X$ or NULL
1: procedure LS_κCC($X_0$)
2:     set the initial *status* to be promising
3:     $X \leftarrow X_0$; $f^*_{\text{best}} \leftarrow f^*(X_0)$; $\mathcal{X} \leftarrow \{X_0\}$; $\iota \leftarrow 0$; $\Upsilon[u] \leftarrow 0, \forall u \in V$
4:     while $f^*(X) \neq 0$ and the termination condition is not met do
5:         $mv \leftarrow \textsc{FindMove}(X, \iota, \Upsilon)$          ▷ Section 3.3
6:         if the current *status* is promising but $mv$ does not improve $X$ then
7:             the current *status* becomes stagnating
8:         else if the *status* is stagnating and $f^*(X \oplus mv) < f^*_{\text{best}}$ then
```

```
9:          the status becomes promising, $f^*_{\text{best}} \leftarrow f^*(X \oplus mv)$
10:     end if
11:     $X, \Upsilon \leftarrow$ MakeMove$(X, mv, \iota, \Upsilon)$    ▷ Section 3.6
12:     if the status is stagnating then
13:         for all $v$ with $z_v = 0$ do
14:             $w_v \leftarrow w_v + 1$
15:         end for
16:     end if
17:     $X, \mathcal{X} \leftarrow$ PerturbDiv$(X, \mathcal{X})$    ▷ Section 3.7
18:     $\iota \leftarrow \iota + 1$
19: end while
20: if $f^*(X) = 0$ then
21:     return $X$
22: else
23:     return NULL
24: end if
25: end procedure
```

In Algorithm 2, the search process operates on the configuration X while maintaining the best objective found so far (denoted as f^*_{best}) and a pool $\mathcal{X}$ of best solutions. Their values are initialized according to the input configuration X_0 (line 3). The procedure also maintains an iteration counter ι indicating the number of iterations that have been elapsed and a tabu table Υ indicating whether it is the tabu state (tabu or not tabu) of each vertex. At each iteration, a move *mv* is chosen from the neighborhood (line 5) and applied to the current configuration (line 11). This process is repeated until the current configuration becomes a CDS or a termination condition is met. In addition, we use *status* to designate whether the current search is promising or stagnating, initialized as promising. Then, the *status* becomes stagnating once the chosen *mv* fails to improve the current configuration for the first time (lines 6 and 7) and returns to be promising again if the chosen *mv* improves the current configuration sufficiently to be better than the best configuration found so far (lines 8 and 9). For each iteration in a stagnating status, the weights for all the nondominated vertices are increased by 1 (lines 12–16). A diversification mechanism is also employed at each iteration to help the search to jump out of local optima (PerturbDiv in line 17), which may possibly update $\mathcal{X}$ too.

The following subsections describe in detail how the subprocedure LS_kCC tackles the k-CC problem by the neighborhood structure, the fast evaluation technique, and the diversification mechanism.

3.3. Neighborhood Structure

We use the following notations to describe our neighborhood structure:

$\mathcal{S}$: The search space (set of configurations) explored by our local search procedure.

X: A configuration $X \in \mathcal{S}$ is any connected set of k vertices.

Art$(G(X))$: The set of articulation vertices of the subgraph of G induced by X.

$\Gamma_{G(X)}$: Relative to an undirected graph $G(V, E)$ and a set $X \subseteq V$ of vertices, $\Gamma_{G(X)} = X \cup \{v \in V : u \in X, \{u, v\} \in E\}$ denotes the set of vertices that are in X or adjacent to any vertex in X. If $\Gamma_{G(X)} = V$, then X is said to be a *dominating set* of G.

X^+: For a given configuration $X \in \mathcal{S}$, $X^+ = \Gamma_{G(X)} \backslash X$ denotes the set of vertices that do not belong to X but are dominated by X.

X^-: For a given configuration $X \in \mathcal{S}$, $X^- = V \backslash \Gamma_{G(X)}$ denotes the set of vertices that are not dominated by X; $X^- = \emptyset$ indicates that X is a *connected dominating set* of G.

Our FVW employs a similar restricted swap-based neighborhood structure used in RSN-TS (Wu et al. 2017) but with more problem-specific restrictions. Denoted by swap(u, v), a swap move produces a neighboring configuration $X \oplus \text{swap}(u, v) = X \cup \{u\} \backslash \{v\}$ by adding vertex u to X and removing vertex v from X. To keep the configuration connected, swap(u, v) must satisfy the following properties.

Property 1. *Vertex* $u \in X^+$.

Property 2. *Vertex* $v \in X \backslash \text{Art}(G(X))$.

Property 3. *There exists* $s \in X \backslash \{v\}$, $(u, s) \in E$.

Property 4. *There exists* $(u, s) \in E$ *and* $s \in X^-$.

According to Property 1, a vertex u to be added to the configuration is any vertex in $V \backslash X$ that connects at least one vertex in X. According to Property 2, a vertex v to be removed from the configuration is any vertex of X that is not an articulation vertex of the subgraph of G induced by X. According to Property 3, v is not the unique vertex in X to which u is connected. Notice that Property 1 requires that $X \cup \{u\}$ is connected, Property 2 ensures that $X \backslash \{v\}$ is connected, and Property 3 further guarantees that $X \oplus \text{swap}(u, v) = X \cup \{u\} \backslash \{v\}$ is also connected. Property 4 is an additional property to require that vertex u must connect to X^-.

The main difference between our proposed move and the one used by Wu et al. (2017) derives from Property 4. The move swap(u, v) applies more restrictions for the vertex to be added to the current configuration by requiring that u must connect to both X and X^-. This helps to bypass evaluating a lot of nonpromising candidate moves, thereby significantly reducing the search space.

3.4. Neighborhood Evaluation

The fast incremental evaluation mechanism of FVW works as follows. At each iteration, the neighborhood moves are evaluated, and one of the highest-quality neighborhood moves is chosen to be applied to the current configuration. The move quality is evaluated by two functions, δ and δ^*, defined as

$$\delta(mv) = f(X \oplus mv) - f(X), \tag{11}$$
$$\delta^*(mv) = f^*(X \oplus mv) - f^*(X). \tag{12}$$

Although both objective functions f and f^* are used, it is time consuming to directly apply the foregoing equations to evaluate the neighborhood. Our algorithm instead employs an incremental evaluation method, where δ is calculated for move swap(u, v) as

$$\delta(\text{swap}(u,v)) = \delta^+(u) + \delta_u^-(v), \tag{13}$$

and δ^* is calculated as

$$\delta^*(\text{swap}(u,v)) = \delta^{*+}(u) + \delta_u^{*-}(v), \tag{14}$$

where $\delta^+(u)$, $\delta^{*+}(u)$ represent the change values incurred by adding vertex u, and $\delta_u^-(v)$, $\delta_u^{*-}(v)$ represent the sequential change values incurred by removing vertex v after adding vertex u.

The calculation of $\delta^+(u)$ and $\delta_u^-(v)$ is the same as in Wu et al. (2017). That is, $\delta^{*+}(u)$ can be calculated as the negative of the accumulated weights of the neighbors of u in X^-. Formally,

$$\delta^{*+}(u) = -\sum_{q\in Q} w_q, \quad Q = \{q : q \in X^-, \exists(u,q) \in E\}; \tag{15}$$

$\delta_u^{*-}(v)$ can be calculated as the accumulated weights of the neighbors of v, which will become nondominated if v is removed. Formally,

$$\delta_u^{*-}(v) = \sum_{q\in Q} w_q, \quad Q = \{q : L[q] = 1, \exists(v,q) \in E, \nexists(u,q) \in E\}. \tag{16}$$

Note that the values of δ^{*+} are always nonpositive, whereas the values of δ_u^{*-} are always nonnegative. The elements $L[i] (1 \le i \le n)$ in list L measure the number of vertices in X connected to vertex i. The maintenance of list L is the same as in Wu et al. (2017).

Our neighborhood move selection strategy is based on the fast incremental neighborhood evaluation method. Algorithm 3 describes how FVW chooses the best move at each iteration of the local search with the tabu strategy. Procedure FindMove first randomly chooses one nondominated vertex y. Then, the nontabu neighbors of y with a small δ^{*+} value (among the top ζ) in X^+ are considered to be candidates for vertices to be added (lines 4 and 5). If all the neighbors of y are tabu vertices ($\Upsilon[u] > \iota$), lines 4 and 5 are repeated to try another y until a nonempty *ivs* is found. All vertices v in X that yield feasible swap(u, v) moves are considered as candidates to be removed. A move is considered feasible if it satisfies all the properties described in Section 3.3. During this process, the best move found so far is stored in *bestMv*. If multiple moves with the same best δ^* value are found, ties are broken randomly. After all the candidate moves are evaluated, the move with the minimum δ^* value is selected. However, the evaluation terminates once an improving negative δ move is found, which implies that the original objective value f is improved.

Algorithm 3 (Neighborhood Move Selection Procedure FindMove)

Input: The current configuration X, the iteration counter ι, and the tabu table Υ
Output: The best move *bestMv*

```
1: procedure FindMove($X, \iota, \Upsilon$)
2:     $bestMv \leftarrow$ NULL
3:     $\delta_{\text{best}} \leftarrow \infty$
4:     Randomly choose one vertex $y \in X^-$
5:     $ivs \leftarrow \{u : u \in X^+, \Upsilon[u] < \iota, \exists(u,y) \in E\}$
6:     Calculate $\delta^+(u)$ and $\delta^{*+}(u)$ for $\forall\ u \in ivs$.
7:     Sort $ivs$ by $\delta^{*+}(u)$ in an increasing order and the age information in a decreasing order.
8:     for all $u \in$ {top $\zeta$ elements in $ivs$} do
9:         for all $v \in X$ do
10:            if swap$(u, v)$ is feasible then
11:                Calculate $\delta_u^-(v)$ and $\delta_u^{*-}(v)$.
12:                if $\delta^+(u) + \delta_u^-(v) < 0$ then
13:                    return swap$(u, v)$
14:                end if
15:                if $\delta^{*+}(u) + \delta_u^{*-}(v) < \delta_{\text{best}}$ then
16:                    $bestMv \leftarrow$ swap$(u,v)$
17:                    $\delta_{\text{best}} \leftarrow \delta^{*+}(u) + \delta_u^{*-}(v)$
18:                end if
19:            end if
20:        end for
21:    end for
22:    return $bestMv$
23: end procedure
```

For each vertex, FVW maintains an age value indicating how many iterations have elapsed since the state of this vertex has been changed either from X to X^- or from X^- to X. This age value serves as a secondary objective when the first objective is tied, as shown in line 7 of Algorithm 3. More specifically, if two vertices have the same δ^{*+} value, the older one has the priority to be selected. This preference for selecting elements based on their age is a common approach in tabu search. In this way, FVW evaluates promising moves with higher priority. Note that the age information is only used in line 7.

It is worth noting that the two objective functions f and f^* are used for different purposes in the FVW algorithm. Function f is used to evaluate the quality of configurations throughout the whole algorithm (e.g., the neighborhood evaluation and the best solutions pool) (described in Section 3.7). The purpose of function f^* is to differentiate configurations with the same quality such that promising moves can be selected among the equal solutions in terms of the objective function f. Therefore, function f^* is used in the neighborhood evaluation phase (procedure FindMove) and to judge whether the algorithm is in a stagnating state (procedure LS_κCC). In the neighborhood evaluation phase, f^* is the main function to evaluate each neighborhood move, whereas f is used to judge only whether the move improves the current configuration in line 12 of Algorithm 3. We will further explain the reason why f is also used here.

3.5. Neighborhood Reduction Technique

As described in Algorithm 3, our FVW employs several important strategies to reduce the neighborhood size in order to efficiently perform the search without sacrificing the search quality. These strategies allow our neighborhood to be quite different from those used in previous state-of-the-art methods, such as RNS-TS (Wu et al. 2017), multi-start local search (Li et al. 2019), and ACO-RVNS (Bouamama et al. 2019).

These FVW strategies are as follows.

1. The neighbors of only one random nondominated vertex are considered to be candidates for vertices to be added. Rather than evaluating all vertices in X^+, the FVW algorithm selects only a small number of vertices as candidates to be added (lines 4 and 5). There is always at least one swap(u, v) satisfying all the properties in a non-CDS configuration (this point is discussed further in the appendix), and adding a vertex connected to X^- is usually a more promising approach for making a nondominated vertex dominated. Thus, FVW focuses only on the vertices connected to X^-. At each iteration, one vertex in X^- is randomly chosen, and its neighbors in X^+ are considered to be the candidates to be added. In this case, it is reasonable to choose only one qualifying vertex rather than to consider all vertices in X^-, because the transformed problem is a decision problem, which means that the optimal objective value of k-CC is 0. Thus, it does not matter which vertex in X^- is first dominated.

2. Only the first ζ vertices are selected for further evaluation as candidates to be added. The FVW algorithm reduces the search space by only considering those candidates that have small δ^{*+} values (lines 6–8). After the candidate-adding vertices are sorted in increasing order of their δ^{*+} values, only the top ζ are considered. The quality of the move largely depends on the δ^{*+} value of an added vertex, because δ^{*+} is always nonpositive, whereas δ_u^{*-} is always nonnegative. The δ^{*+} value must be negative if the move is improving. The smaller δ^{*+} is, the more likely it is an improving move. By only considering a small number of candidate vertices with small δ^{*+} values, the search space is significantly reduced. This technique is very effective, especially for large-scale dense graphs as shown in Section 5.2.

3. The first improvement strategy is used for the original objective function f. The procedure searches the move with the best δ^* value. However, the procedure terminates and returns the current move swap(u, v) if $\delta^+(u) + \delta_u^-(v) < 0$; that is, swap($u$, v) improves the original objective function f. Our experiments demonstrate that this first improvement strategy reduces the computational time without sacrificing the quality of the final solution, compared with the RSN-TS, which employs the best improvement strategy. We note that little knowledge is available about what types of problem and neighborhood structures can be exploited more effectively with a first improvement strategy or, alternatively, with a best improvement strategy. Our investigation shows that for the present application and our chosen neighborhood (implemented within a tabu search framework) that a first improvement strategy proves superior.

3.6. Applying the swap(u, v) Move

The procedure MakeMove described in Algorithm 4 applies the move selected by the procedure FindMove to the current configuration. It also updates the tabu table Υ accordingly at each iteration. More specifically, if we remove vertex v from X, its corresponding value in the tabu table $\Upsilon[v]$ is reset to be the current iteration count plus the tabu tenure tt. Therefore, v will be excluded for the next tt iterations from the candidate vertices to be added in the procedure FindMove. We fix the parameter of the tabu tenure to be a random integer value in the range $[0,10)$ (i.e., $tt \in \{1,2,...,9\}$.

Algorithm 4 (Applying Move Procedure MakeMove)

```
Input: The current configuration $X$, the move swap($u$, $v$), the iteration counter $\iota$, and the tabu table $\Upsilon$
Output: The updated $X$ and $\Upsilon$
1: procedure MakeMove($(X, \text{swap}(u,v), \iota, \Upsilon)$)
2:     $X \leftarrow X \cup \{u\}$
3:     $X \leftarrow X \backslash v$
4:     $\Upsilon[v] \leftarrow \iota + tt$
5:     return $X, \Upsilon$
6: end procedure
```

3.7. Perturbation Operator

The random perturbation operator has been widely used in metaheuristics as a diversification technique. Applying the local search again on the perturbed configuration offers a possibility that a better local optimum may be obtained. Perturbation can be considered as a jump in the search space, and performing a perturbation when trapped in a local optimum may afford a means to jump out of the trap. Inspired by the breakout local search algorithm (Benlic and Hao 2013a, b, c; Peng et al. 2020), our perturbation consists of a series of random moves. The strength of the perturbation is measured by the number of random moves to be performed. Algorithm 5 describes how the perturbation mechanism is performed in the FVW. Our perturbation operator is controlled by three parameters, *minStr*, *maxStr*, and β, which are fixed during the procedure PerturbDiv. The parameters *minStr* and *maxStr* represent the minimum and maximum perturbation strength, respectively. The parameter β represents the number of stagnation iterations allowed before triggering the perturbation operator. The procedure operates by manipulating two global quantities, *g_str* and *g_stagIters*, initialized to equal *minStr* and 0, respectively, at the beginning of FVW. The quantity *g_str* records the current perturbation strength, and *g_stagIters* records the number of stagnation iterations since the last improvement for the original objective value f.

Algorithm 5 (The Procedure of the Perturbation Operator PerturbDiv)

```
Input: The current configuration $X$, the best solution pool $\mathcal{X}$
Output: The updated $X$ and $\mathcal{X}$
1: procedure PerturbDiv($X$, $\mathcal{X}$)
2:     if $f(X) = f_{best}$ and $X \notin \mathcal{X}$ then
3:         $\mathcal{X} \leftarrow \mathcal{X} \cup \{X\}$, $h_X \leftarrow 1$
4:         $g\_str \leftarrow minStr$
5:     else if $f(X) < f(\mathcal{X}_0)$ then    ▷ $\mathcal{X}_0$ represents the first element in $\mathcal{X}$
6:         $\mathcal{X} \leftarrow \{X\}$
7:         $g\_str \leftarrow minStr, g\_stagIters \leftarrow 0$
8:     else if $X \in \mathcal{X}$ then
9:         $g\_str \leftarrow \min\{g\_str + 1, maxStr\}$
10:        $h_X \leftarrow h_X + 1$
11:    end if
12:    $g\_stagIters \leftarrow g\_stagIters + 1$
13:    if $g\_stagIters > \beta$ then
14:        $X \leftarrow$ element with the smallest $h_X$ in $\mathcal{X}$
15:        repeat
16:            $mv \leftarrow$ random swap$^*(u,v)$
17:            $X \leftarrow$ MakeMove($X, mv$)
18:        until $g\_str$ times
19:        $g\_stagIters \leftarrow 0$
20:    end if
21:    return $X, f_{best}, \mathcal{X}$
22: end procedure
```

Procedure PerturbDiv is performed at the end of each iteration in procedure LS_kCDS (line 17 of Algorithm 2). It maintains a pool of configurations $\mathcal{X}$ consisting of all solutions with the same minimum size of X^-. If the current solution X has the same objective value f as the elements in $\mathcal{X}$ but with different configurations, it will be added to $\mathcal{X}$ (lines 2 and 3). Once solution X is better than the configurations in $\mathcal{X}$, $\mathcal{X}$ is initialized as $\{X\}$ (lines 5–7). In both cases, *g_str* is set to be *minStr*. However, *g_str* increases by 1 if X coincides one of the configurations in $\mathcal{X}$ but is bounded by *maxStr* (line 9). When the number of stagnation iterations *g_stagIters* reaches the threshold β, a perturbation is triggered by performing random swap$^*(u,v)$ moves for *g_str* times on one of the configurations in $\mathcal{X}$, where swap$^*(u,v)$ represents the move satisfying Properties 1, 2, and 3 (lines 13–20). For each configuration X in $\mathcal{X}$, there is an associated value h_X indicating how many times the search procedure visits this configuration. When X is added to $\mathcal{X}$, h_X is initialized with value 1 (line 3) and increases by 1 if X is revisited in the search (line 10). To make the algorithm as diversifying as possible, the perturbation is performed on the configuration with the minimum h_X value in $\mathcal{X}$ (line 14).

We fix *minStr* to be $|V|/10$ and *maxStr* to be $|V|$. For some large dense graphs, where the CDS is only a small portion of the set of vertices, stronger perturbations may produce better results.

4. Computational Results

In this section, we report the experimental results of FVW on the widely used benchmark instances in the literature compared with several state-of-the-art reference algorithms.

4.1. Problem Instances and Experimental Protocol

Our experiments are carried out on the following four sets of public benchmark instances:

• LMS consists of 41 problem instances proposed by Lucena et al. (2010) whose optima are known.

• BPFTC consists of 6 problem instances whose optima are known. These graphs belong to the Bus Power Flow Test Case (BPFTC), which is a subset of the Power Systems Test Case.[1] Each graph represents a portion of the American Electric Power System. These instances were used by Wu et al. (2017) and Gendron et al. (2014).

• RGG consists of 41 large random geometric graphs (RGGs) proposed by Jovanovic and Tuba (2013) whose optima are unknown. A random geometric graph is obtained by randomly placing n vertices in an $N \times N$ square and connecting two vertices with an edge if their distance is smaller than a given threshold R (radius).

• BOBL consists of 24 large difficult instances ranging from 1,000 to 5,000 vertices introduced by Bouamama et al. (2019). The optima for this data set are unknown.

The data set newly introduced in this paper, Sparse, consists of 15 large random sparse graphs. The optima for this data set are unknown.

Our FVW algorithm was programmed in Java, and the experiments were conducted on a PC with an Intel i7 CPU with 2.9 GHz and 16 GB of RAM. The reference algorithm RSN-TS (Wu et al. 2017) was retested along with the newly introduced instances on the same platform as FVW, whereas ACO-RVNS (Bouamama et al. 2019) was tested on a cluster of computers with an Intel Xeon CPU with 2.933 GHz with a time limit of 600 seconds. In order to make fair comparison by considering the processor difference, the time limit of our algorithm (and RSN-TS) is 600 seconds (the total time from obtaining the initial k to the final k). Each instance is independently run 10 times. The exact approach LCBC (Buchanan et al. 2015) was tested using Gurobi Optimizer, version 5.5, on a Dell Precision WorkStation T7500 machine with two Intel Xeon E5620 2.40 GHz quad-core processors and 12 GB RAM. All compared algorithms are run in single-thread mode, except for LCBC.

4.2. Calibration

Experiments were conducted to fix the values of the two key parameters of the FVW algorithm:

- Parameter ζ for the search space reduction
- Parameter β for triggering the perturbation

Several representative instances were chosen as the testbed. The algorithm was run 10 times on each problem instance and each parameter setting with a time limit of 300 seconds for each run. Table 1 reports the results of this experiment, where the column "Best" reports the minimum CDS size obtained over the 10 runs, the column "Count" reports the number of times that the algorithm obtains the best solution, and the column "Time" reports the average time consumed if all the 10 runs produce the same results. The best value is indicated in bold if it is equal to or better than the others. The time and the count values are also indicated in bold if both its corresponding best objective value (the "Best" column) and itself are equal to or better than the others. The "Time" column in the following tables always reports the average time consumed when the algorithm reaches the best objective. The value is marked with an em dash (—) if the average objective is not equal to the best one.

Table 1 discloses that the performance differences between the different settings are negligible for small- and medium-scale instances. This indicates the stability of the FVW algorithm to some extent. However, we observe that the parameter setting $\{\zeta = 10, \beta = 320,000\}$ leads to better objective values for the large difficult instances compared with other settings.

In our experiments, the parameters of the algorithm are fixed to $\{\zeta = 10, \beta = 320,000\}$ for all problem instances based on the preceding calibration experiment.

4.3. Computational Results on the LMS and BPFTC Instances

Both LMS and BPFTC are instances with known optima proven by the exact algorithm introduced by Gendron et al. (2014) and are widely used in other references (Lucena et al. 2010, Buchanan et al. 2015, Wu et al. 2017, Bouamama et al. 2019). Although the scale of these instances is not very large, the instance ieee-bus-300 is considered one of the most difficult instances and is typically used to test the effectiveness of MCDS algorithms. The computational results on these two data sets are presented in Table 2. For each algorithm except for LCBC, we report the best and average objective values as well as the average time (seconds) to reach the best objective value. For LCBC, we report the objective value obtained by the algorithm and the time (seconds) to reach this value. Because there is no obvious difference among different algorithms on these small instances, we only present results for those instances with more than 120 vertices in LMS and more than 70 vertices in BPFTC. Our FVW is compared with RSN-TS, ACO-RVNS, and LCBC. All three heuristic algorithms can stably obtain the optimal results within less than one second for the instances of LMS and BPFTC not reported in Table 2 (LCBC takes about one second or less on these instances).

Table 2 shows that all four algorithms can obtain optimal solutions for all instances in these two data sets, but ACO-RVNS is not able to obtain the best results every time for instance v150-d10. In terms of efficiency, FVW spends much less computational time than does RSN-TS for all the instances. When compared with ACO-RVNS, FVW can obtain the best results in less time or nearly the same computational time as ACO-RVNS for all but two instances. All three heuristics obtain optimal solutions faster than does LCBC. However, LCBC is competitive with RSN-TS on most small-scale instances. From the table, we observe that instances in BPFTC seem to be harder for the proposed FVW compared with other instances. FVW takes 17.9 seconds to obtain an optimal solution for the instance ieee-300-bus, much slower than other instances. One reason may be that its scale is much larger (300 vertices) than other instances in these two data sets. However, the instance rts96 with only 73 vertices still takes much longer time than instances with more vertices in LMS. From this experiment, we find that the ratios of the number of edges to vertices for these two instances are 1.36 and 1.48, whereas the ratios for other larger instances in LMS data sets are much larger. It indicates, to some extent, that sparser graphs are more difficult than denser ones for the proposed FVW algorithm.

4.4. Computational Results on the RGG Instances

Computational results comparing the FVW algorithm with RSN-TS and ACO-RVNS for RGG instances are presented in Table 3. Instances where our FVW algorithm is able to improve the previous best-known results are indicated with an asterisk (*).

Table 3 shows that FVW outperforms RSN-TS and ACO-RVNS on this data set in all three categories of

Table 1. Calibration Experiment for Parameter Settings

ζ, β	1, 80,000			10, 80,000			20, 80,000			10, 320,000		
Instance	Best	Time	Count	Best	Time	Count	Best	Time	Count	Best	Time	Count
v200-d5	**27**	9.3	**10**	**27**	0.2	**10**	**27**	0.2	**10**	**27**	0.2	**10**
ieee-300-bus	**129**	243.3	**10**	**129**	**6.7**	**10**	**129**	10.3	**10**	**129**	11.7	**10**
n1000-200-r100	**38**	40.1	**10**	**38**	0.6	**10**	**38**	**0.5**	**10**	**38**	1.0	**10**
n1500-250-r130	**49**	13.0	**10**	**49**	0.3	**10**	**49**	**0.2**	**10**	**49**	**0.2**	**10**
n2000-300-r200	**41**	21.7	**10**	**41**	2.4	**10**	**41**	**1.2**	**10**	**41**	2.3	**10**
n2500-350-r200	**60**	11.8	**10**	**60**	0.4	**10**	**60**	**0.3**	**10**	**60**	0.4	**10**
n2500-350-r230	**48**	—	8	**48**	20.0	**10**	**48**	19.6	**10**	**48**	**16.4**	**10**
n3000-400-r210	**74**	26.9	**10**	**74**	1.4	**10**	**74**	1.6	**10**	**74**	2.1	**10**
n3000-400-r230	**64**	—	8	**64**	**6.8**	**10**	**64**	10.9	**10**	**64**	9.6	**10**
n1000-ep0014	**98**	—	3	**98**	72.4	**10**	**98**	31.8	**10**	**98**	68.6	**10**
n1000-r0048	273	—	1	**271**	—	**3**	**271**	—	2	**271**	—	1
n5000-ep0014	170	—	1	164	—	1	165	—	1	**163**	—	1
n5000-r0070	**124**	—	2	**124**	—	1	125	—	7	**124**	—	**3**
ζ, β	1, 160,000			10, 160,000			20, 160,000			20, 320,000		
v200-d5	**27**	15.9	**10**	**27**	**0.1**	**10**	**27**	0.2	**10**	**27**	**0.1**	**10**
ieee-300-bus	**129**	—	9	**129**	13.1	**10**	**129**	17.4	**10**	**129**	19.4	**10**
n1000-200-r100	**38**	73.1	**10**	**38**	0.6	**10**	**38**	0.7	**10**	**38**	0.7	**10**
n1500-250-r130	**49**	37.1	**10**	**49**	0.3	**10**	**49**	**0.2**	**10**	**49**	0.4	**10**
n2000-300-r200	**41**	121.8	**10**	**41**	**1.2**	**10**	**41**	1.9	**10**	**41**	4.4	**10**
n2500-350-r200	**60**	27.6	**10**	**60**	**0.3**	**10**	**60**	0.5	**10**	**60**	**0.3**	**10**
n2500-350-r230	**48**	—	7	**48**	22	**10**	**48**	16.6	**10**	**48**	29.0	**10**
n3000-400-r210	**74**	75.7	**10**	**74**	1.6	**10**	**74**	**0.7**	**10**	**74**	1.4	**10**
n3000-400-r230	**64**	128.4	**10**	**64**	10.4	**10**	**64**	9.9	**10**	**64**	10.3	**10**
n1000-ep0014	99	—	9	**98**	**25.8**	**10**	**98**	53.7	**10**	**98**	46.7	**10**
n1000-r0048	274	—	1	**271**	—	**5**	**271**	—	1	**271**	—	**3**
n5000-ep0014	172	—	2	164	—	**5**	165	—	5	164	—	2
n5000-r0070	**124**	—	**3**	**124**	—	1	**124**	—	2	**124**	—	**3**

Note. The value is indicated in bold if it is equal to or better than others.

Table 2. Computational Results of FVW and Comparisons with RSN-TS, ACO-RVNS, and LCBC on the LMS and BPFTC Data Sets

	FVW			RSN-TS			ACO-RVNS			LCBC	
Instance	Best	Average	Time	Best	Average	Time	Best	Average	Time	Obj.	Time
v120-d5	25	25.0	<0.1	25	25.0	0.5	25	25.0	0.2	25	0.3
v120-d10	13	13.0	<0.1	13	13.0	0.5	13	13.0	0.5	13	0.3
v120-d20	8	8.0	<0.1	8	8.0	0.3	8	8.0	0.2	8	1.8
v120-d30	6	6.0	<0.1	6	6.0	0.1	6	6.0	<0.1	6	2.3
v120-d50	4	4.0	<0.1	4	4.0	0.1	4	4.0	<0.1	4	1.6
v120-d70	3	3.0	<0.1	3	3.0	0.1	3	3.0	<0.1	3	2.4
v150-d5	26	26.0	<0.1	26	26.0	1.1	26	26.0	0.1	26	3.4
v150-d10	14	14.0	<0.1	14	14.0	2.6	14	14.5	—	14	4.7
v150-d20	9	9.0	<0.1	9	9.0	1.2	9	9.0	<0.1	9	9.3
v150-d30	6	6.0	<0.1	6	6.0	2.6	6	6.0	<0.1	6	6.5
v150-d50	4	4.0	<0.1	4	4.0	0.9	4	4.0	<0.1	4	2.4
v150-d70	3	3.0	<0.1	3	3.0	2.7	3	3.0	<0.1	3	4.7
v200-d5	27	27.0	<0.1	27	27.0	7.5	27	27.0	7.2	27	32.9
v200-d10	16	16.0	<0.1	16	16.0	1.5	16	16.0	0.2	16	496.4
v200-d20	9	9.0	<0.1	9	9.0	1.5	9	9.0	0.4	9	243.2
v200-d30	7	7.0	<0.1	7	7.0	1.6	7	7.0	<0.1	7	172.5
v200-d50	4	4.0	<0.1	4	4.0	1.3	4	4.0	0.2	4	8.1
v200-d70	3	3.0	<0.1	3	3.0	1.1	3	3.0	<0.1	3	9.45
rts96	32	32.0	2.9	32	32.0	1.4	32	32.0	<0.1	32	0.6
ieee-118-bus	43	43.0	0.1	43	43.0	0.6	43	43.0	<0.1	43	<0.1
ieee-300-bus	129	129.0	17.9	129	129.0	56.5	129	129.0	4.9	129	52.8

Table 3. Computational Results of FVW and Comparisons with RSN-TS and ACO-RVNS on the RGG Data Set

	FVW				RSN-TS				ACO-RVNS		
Instance	Best	Average	Time	s-dev	Best	Average	Time	s-dev	Best	Average	Time
n400-80-r60	**18**	**18.0**	<0.1	0.00	**18**	**18.0**	0.1	0.00	**18**	**18.0**	<0.1
n400-80-r70	**14**	**14.0**	<0.1	0.00	**14**	**14.0**	0.2	0.00	**14**	**14.0**	0.3
n400-80-r80	**12**	**12.0**	<0.1	0.00	**12**	**12.0**	0.1	0.00	**12**	**12.0**	<0.1
n400-80-r90	**10**	**10.0**	<0.1	0.00	**10**	**10.0**	0.4	0.00	**10**	**10.0**	0.2
n400-80-r100	**8**	**8.0**	<0.1	0.00	**8**	**8.0**	0.2	0.00	**8**	**8.0**	0.4
n400-80-r110	**7**	**7.0**	0.1	0.00	**7**	**7.0**	1.9	0.00	**7**	**7.0**	<0.1
n400-80-r120	**6**	**6.0**	<0.1	0.00	**6**	**6.0**	1.3	0.00	**6**	**6.0**	0.4
n600-100-r80	**21**	**21.0**	<0.1	0.00	**21**	**21.0**	0.2	0.00	**21**	**21.0**	<0.1
n600-100-r90	**19**	**19.0**	<0.1	0.00	**19**	**19.0**	1.3	0.00	**19**	**19.0**	<0.1
n600-100-r100	**16**	**16.0**	<0.1	0.00	**16**	**16.0**	1.4	0.00	**16**	**16.0**	0.2
n600-100-r110	**14**	**14.0**	0.1	0.00	15	15.0	0.3	0.00	**14**	**14.0**	3.5
n600-100-r120	**13**	**13.0**	<0.1	0.00	**13**	**13.0**	0.3	0.00	**13**	**13.0**	0.1
n700-200-r70	**38**	**38.0**	0.4	0.00	39	39.0	17.3	0.00	**38**	38.1	—
n700-200-r80	**32**	**32.0**	0.1	0.00	**32**	**32.0**	15.2	0.00	**32**	**32.0**	28.9
n700-200-r90	**26**	**26.0**	0.1	0.00	**26**	**26.0**	5.3	0.00	**26**	**26.0**	7.9
n700-200-r100	**22**	**22.0**	0.2	0.00	**22**	**22.0**	61.6	0.00	**22**	**22.0**	26.0
n700-200-r110	**20**	**20.0**	<0.1	0.00	**20**	**20.0**	3.5	0.00	**20**	**20.0**	1.3
n700-200-r120	**17**	**17.0**	<0.1	0.00	**17**	**17.0**	11.7	0.00	**17**	**17.0**	3.4
n1000-200-r100	**38**	**38.0**	0.6	0.00	**38**	38.4	—	0.63	**38**	38.2	—
n1000-200-r110	**34**	**34.0**	0.1	0.00	**34**	**34.0**	9.2	0.00	**34**	**34.0**	2.1
n1000-200-r120	**29**	**29.0**	0.3	0.00	**29**	**29.0**	86.5	0.00	**29**	**29.0**	48.1
n1000-200-r130	**26**	**26.0**	<0.1	0.00	**26**	**26.0**	12.9	0.00	**26**	**26.0**	6.9
n1000-200-r140	**23**	**23.0**	<0.1	0.00	**23**	**23.0**	5.9	0.00	**23**	**23.0**	2.2
n1000-200-r150	**21**	**21.0**	0.1	0.00	**21**	**21.0**	9.7	0.00	**21**	**21.0**	10.0
n1000-200-r160	**19**	**19.0**	0.5	0.00	**19**	**19.0**	71.5	0.00	**19**	**19.0**	27.8
n1500-250-r130	**49**	**49.0**	0.5	0.00	**49**	**49.0**	111.3	0.00	**49**	49.1	—
n1500-250-r140	**43**	**43.0**	2.7	0.00	44	44.0	19.4	0.00	**43**	43.9	—
n1500-250-r150	**40**	**40.0**	0.5	0.00	**40**	**40.0**	91.7	0.00	**40**	40.7	—
n1500-250-r160	**36**	**36.0**	0.1	0.00	**36**	**36.0**	31.8	0.00	**36**	**36.0**	4.4
n2000-300-r200	***41**	**41.0**	1.7	0.00	42	42.0	135.9	0.00	42	42.1	—
n2000-300-r210	**38**	**38.0**	0.2	0.00	**38**	38.3	—	0.55	**38**	**38.0**	39.5
n2000-300-r220	**35**	**35.0**	0.2	0.00	**35**	**35.0**	134.0	0.00	**35**	35.1	—
n2000-300-r230	**33**	**33.0**	0.6	0.00	**33**	**33.0**	284.5	0.00	**33**	**33.0**	58.4
n2500-350-r200	***59**	**59.8**	—	0.40	61	61.1	—	0.32	60	60.7	—
n2500-350-r210	***54**	**54.0**	3.6	0.00	56	56.0	119.2	0.00	55	56.1	—
n2500-350-r220	**51**	**51.0**	1.7	0.00	52	52.0	123.3	0.00	**51**	52.7	—
n2500-350-r230	**48**	**48.0**	33.4	0.00	50	50.0	133.9	0.00	**48**	49.5	—
n3000-400-r210	**74**	**74.0**	1.1	0.00	75	75.3	—	0.55	**74**	75.5	—
n3000-400-r220	***69**	**69.0**	51.2	0.00	71	71.0	129.6	0.00	70	70.8	—
n3000-400-r230	***64**	**64.0**	11.9	0.00	65	65.6	—	0.77	65	65.9	—
n3000-400-r240	***60**	**60.0**	3.5	0.00	61	61.5	—	0.71	61	61.7	—

Notes. The asterisk mark indicates that FVW algorithm is able to improve the previous best-known result for the instance. The value is indicated in bold if it is equal to or better than others.

best results, average results, and average computational time. For FVW and RSN-TS, we also present the standard deviation (the column "s-dev") of the objective values obtained in 10 runs. FVW outperforms ACO-RVNS on 6 instances with smaller best objective values and outperforms RSN-TS on 12 instances. For 9 instances, FVW obtains better average objective values than does ACO-RVNS even where their best results are the same, and FVW improves the previous best-known results for 6 instances in this data set. FVW shows remarkable robustness in this data set. It solves all the instances except for only one to the same best objective values, which indicates the robustness of the algorithm. For the instance n2500-350-r200, the average result is not equal to the best result, where the standard deviation is 0.40.

4.5. Computational Results on the BOBL Instances

The benchmark BOBL consists of 24 random instances with vertices numbering from 1,000 to 5,000. These are considered to be the most difficult instances for most existing MCDS algorithms. Table 4 presents the computational results of comparing FVW with RSN-TS and ACO-RVNS.

Table 4 shows that FVW outperforms RSN-TS in terms of all the criteria. The performance of RSN-TS deteriorates dramatically for the instances with 5,000

Table 4. Computational Results of FVW and Comparisons with RSN-TS and ACO-RVNS on the BOBL Data Set

	FVW				RSN-TS				ACO-RVNS		
Instance	Best	Average	Time	s-dev	Best	Average	Time	s-dev	Best	Average	Time
n1000-ep0007	***179**	**179.0**	21.5	0.00	188	190.7	—	1.35	185	186.6	—
n1000-ep0014	***98**	**98.0**	46.9	0.00	103	104.3	—	0.78	101	103.1	—
n1000-ep0028	***59**	**59.0**	42.1	0.00	62	62.6	—	0.66	61	62.8	—
n1000-ep0056	**37**	**37.0**	2.2	0.00	**37**	37.5	—	0.50	**37**	37.8	—
n1000-ep0112	**22**	**22.0**	1.1	0.00	**22**	**22.0**	178.5	0.00	**22**	**22.0**	77.5
n1000-ep0224	**12**	**12.0**	3.2	0.00	**12**	**12.0**	159.9	0.00	**12**	12.1	—
n1000-r0048	***271**	**271.9**	—	0.70	277	281.4	—	1.74	275	275.8	—
n1000-r0070	***123**	**123.7**	—	0.67	129	134.2	—	2.82	127	128.4	—
n1000-r0100	***60**	**60.2**	—	0.40	63	65.8	—	1.40	64	65.4	
n1000-r0140	***31**	**31.0**	340.0	0.00	32	33.7	—	0.90	32	32.9	—
n1000-r0207	**15**	**15.0**	27.2	0.00	**15**	15.5	—	0.50	**15**	15.7	—
n1000-r0308	**7**	**7.0**	27.8	0.00	**7**	7.3	—	0.90	**7**	**7.0**	0.5
n5000-ep0007	***273**	**273.2**	—	0.40	444	578.0	—	52.92	281	283.7	—
n5000-ep0014	***163**	**164.1**	—	0.83	270	293.8	—	16.53	165	166.2	—
n5000-ep0028	96	97.2	—	0.75	147	167.7	—	11.02	**95**	**95.8**	—
n5000-ep0056	**56**	56.9	—	0.54	119	808.5	—	676.59	**56**	**56.1**	—
n5000-ep0112	32	32.2	—	0.40	3,418	3,504.2	—	87.30	**31**	**31.7**	—
n5000-ep0224[a]	18	18	188.4	0.00	4,165	4,210.5	—	25.66	**17**	**17.1**	—
n5000-r0048	***263**	**264.2**	—	0.60	345	364.0	—	11.38	294	298.5	—
n5000-r0070	***124**	**124.9**	—	0.70	158	163.0	—	3.44	141	143.2	—
n5000-r0100	***62**	**62.2**	—	0.40	77	81.1	—	3.30	72	72.5	—
n5000-r0140	***32**	**32.6**	—	0.49	43	189.2	—	216.31	36	36.9	—
n5000-r0207	**16**	**16.0**	287.6	0.00	3,236	3,431.0	—	136.96	**16**	16.8	—
n5000-r0308	***7**	7.9	—	0.70	4,043	4,148.6	—	72.61	8	8.0	28.9

Note. The value is indicated in bold if it is equal to or better than others.
[a]FVW can obtain an objective value 17 for n5000-ep0224 if the time limit is extended to 1,000 seconds.
*FVW algorithm is able to improve the previous best-known result for the instance.

vertices, perhaps because of the inefficiency of its neighborhood evaluations. We also observe that except for three instances (n5000-ep0028, n5000-ep0112, and n5000-ep0224), FVW obtains results that match or are better than those of the other algorithms within the time limit. (The objective value 17 can be obtained for instance n5000-ep0224 if the time limit is extended to 1,000 seconds.) For all the instances, the standard deviations are less than 1, demonstrating the robustness of the proposed FVW. For four instances, FVW obtains better average objective values than ACO-RVNS even where their best results are the same. For two instances, both algorithms reach the same best and average results. For one instance, FVW obtains the same best result but worse average objective value than ACO-RVNS. In particular, FVW is able to improve the best-known results for 14 instances in this data set.

4.6. Computational Results on the Sparse Instances

From Table 2, we observe that sparse graphs are relatively difficult for our proposed FVW algorithm. We thus generate 15 large sparse instances to test the performance of FVW as well as providing a benchmark for future comparison. The Sparse data set can be obtained from the website https://github.com/xavierwoo/Sparse-MCDS-Benchmark.git (accessed September 29, 2021). The instances in this Sparse data set are generated in the following way:

1. Generate a vertex set V with size N.
2. Generate a random spanning tree (V, E') on V.
3. Select N' vertices randomly as set V' such that $V' \subset V$.
4. Generate a random spanning tree (V', E'') on V'.
5. The graph $(V, E' \cup E'')$ is the instance.

The instance generated in this way is always sparse and connected, and the ratio of its edge set size to its vertex set size is no more than $(N + N' - 2)/N$. The instances in this data set are named *sparse-nN-npN′*.

Table 5 presents the computational results of comparing FVW with RSN-TS. From the table, we observe that FVW produces much better results on most of the instances than RSN-TS, indicating the effectiveness of FVW. For two instances with 1,000 vertices, RSN-TS slightly outperforms FVW in terms of the best results. However, FVW still obtains better average results than RSN-TS for these two instances. The reason might lie in the fact that the neighborhood of RSN-TS is larger than that of FVW. FVW may drop certain promising moves, thus producing a worse result. However, for larger or denser instances, FVW can produce much better results than RSN-TS. For example, FVW produces better results than RSN-TS for instance sparse-n1000-np700, which is denser than the other two with 1,000 vertices.

Table 5. Computational Results of FVW and Comparisons with RSN-TS on the Sparse Data Set

	FVW				RSN-TS			
Instance	Best	Average	Time	s-dev	Best	Average	Time	s-dev
sparse-n1000-np300	429	429.0	1.2	0.00	**428**	429.4	—	2.29
sparse-n1000-np500	377	377.0	1.0	0.00	**376**	377.1	—	0.70
sparse-n1000-np700	**337**	337.0	5.0	0.00	338	340.8	—	1.47
sparse-n1500-np500	**628**	628.2	—	0.40	632	634.9	—	2.43
sparse-n1500-np700	**579**	579.0	5.9	0.00	583	593.3	—	5.95
sparse-n1500-np900	**534**	534.0	43.0	0.00	545	550.8	—	3.16
sparse-n2000-np600	**864**	864.0	10.6	0.00	865	869.1	—	2.17
sparse-n2000-np800	**803**	803.0	50.1	0.00	815	820.3	—	3.93
sparse-n2000-np1000	**755**	755.0	10.7	0.00	762	769.3	—	5.18
sparse-n2500-np800	**1,048**	1,049.4	—	1.11	1,055	1,061.0	—	3.52
sparse-n2500-np1000	**1,003**	1,003.0	23.4	0.00	1,009	1,018.6	—	7.94
sparse-n2500-np1200	**937**	937.0	7.1	0.00	948	954.5	—	5.55
sparse-n3000-np1000	**1,239**	1,239.0	71.6	0.00	1,247	1,254.9	—	5.72
sparse-n3000-np1300	**1,179**	1,179.0	73.4	0.00	1,193	1,207.2	—	7.40
sparse-n3000-np1600	**1,106**	1,106.6	—	0.66	1,128	1,138.9	—	5.75

Note. The value is indicated in bold if it is equal to or better than others.

5. Discussion and Analysis

Now we turn our attention to evaluating the primary strategies of the FVW algorithm, which are the vertex weighting technique and the search space reduction technique. In this section, the experiments are performed on selected representative instances from Section 4.2, but it is to be noted that similar results can be observed for other instances as well.

5.1. Importance of FVW Ingredients

To evaluate the merits of the primary FVW strategies, we conducted experiments on the representative instances to compare FVW with its simplified versions obtained by

• deactivating the vertex weighting technique (FLS) and

• disabling the perturbation phase (FVW-S).

Table 6 presents the computational results of comparing FVW with its two simplified versions. From Table 6, we can observe that both simplified versions fail to obtain results as good as the full-featured FVW algorithm. In particular, FLS can only obtain the same best results found by FVW on two instances. This indicates that the vertex weighting technique, missing in FLS, is a critical ingredient of the FVW. The perturbation operator is also important because FVW-S fails to produce the best-known results every time for instance ieee-300-bus, and it obtains worse results than FVW for the last three large-scale instances.

5.2. The Importance of the Search Space Reduction

In Section 3.5, we introduced the search space reduction technique to boost the performance of our FVW algorithm. To evaluate how it improves the search efficiency, we carry out experiments to investigate the performance of FVW with and without some component strategies of this technique. The following six versions of the algorithm are considered: (1) the full-featured FVW algorithm, (2) the FVWn1 algorithm where the first strategy is disabled, (3) the FVWn2 algorithm where the second strategy is disabled, (4) the FVWn3 algorithm where the third strategy is disabled, (5) the FVWn12 algorithm where the first and second strategies are disabled, and (6) the FVWn23 algorithm where the second and third strategies are disabled.

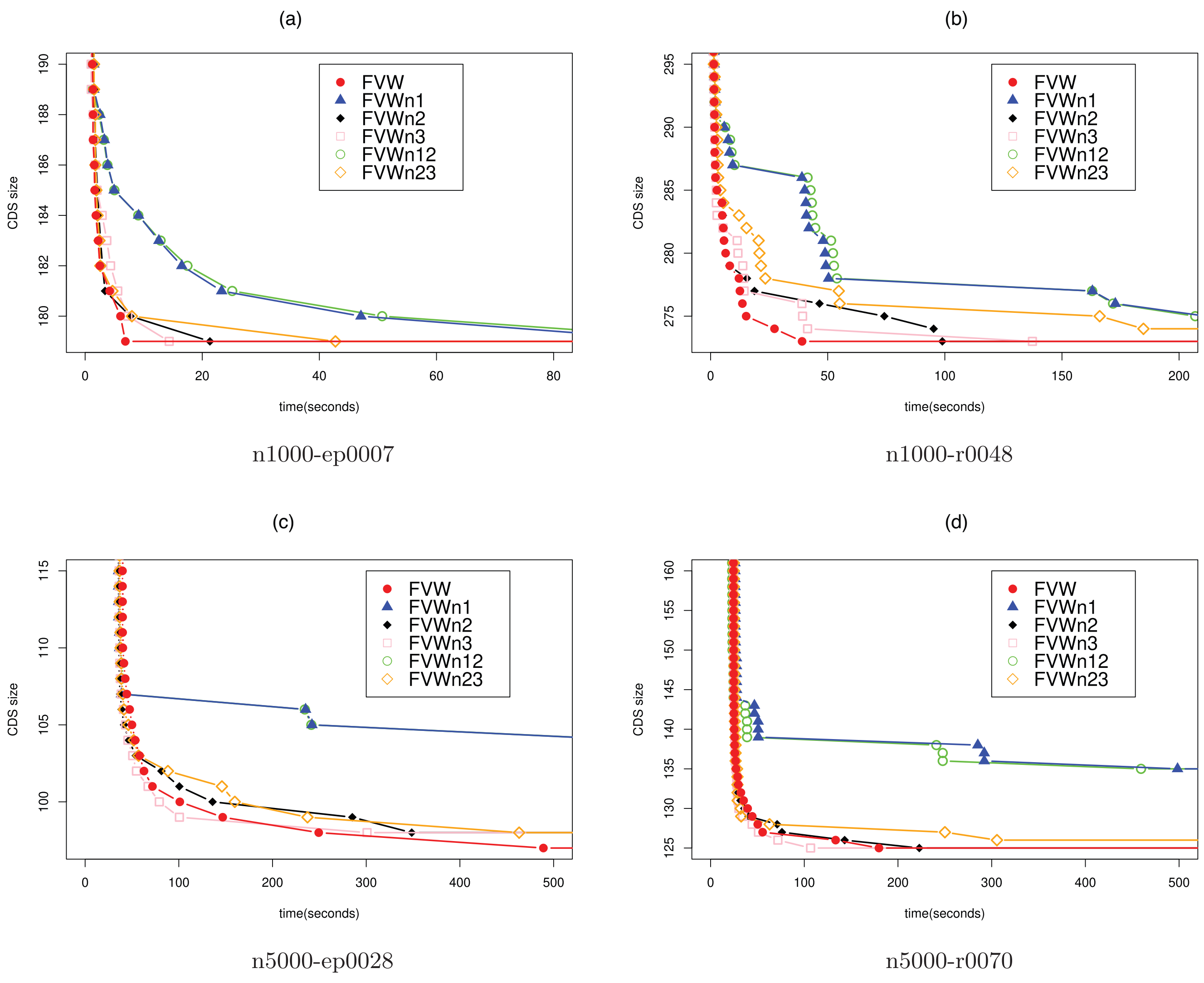
We conduct experiments on four representative instances: n1000-ep0007, n1000-r0048, n5000-ep0028, and n5000-r0070. Similar results can be observed for other instances, especially for large-scale problems. The evolution of the minimum size of the CDS found so far with the computational time is shown in Figure 1.

Table 6. Comparison Among FVW and Its Simplified Versions

	FVW		FLS		FVW-S	
Instance	Best	Average	Best	Average	Best	Average
v200-d5	**27**	**27.0**	**27**	27.5	**27**	**27.0**
ieee-300-bus	**129**	**129.0**	133	135.6	**129**	129.9
n1000-200-r100	**38**	**38.0**	40	42.6	**38**	**38.0**
n1500-250-r130	**49**	**49.0**	52	56.6	**49**	**49.0**
n2000-300-r200	**41**	**41.0**	43	45.8	**41**	**41.0**
n2500-350-r200	**60**	**60.0**	65	69.6	**60**	**60.0**
n2500-350-r230	**48**	**48.0**	50	55.3	**48**	**48.0**
n3000-400-r210	**74**	**74.0**	79	86.9	**74**	**74.0**
n3000-400-r230	**64**	**64.0**	70	74.7	**64**	**64.0**
n1000-ep0014	**98**	**98.0**	99	102.4	**98**	**98.0**
n1000-r0048	**271**	**271.9**	285	290.4	272	273.7
n5000-ep0014	**163**	**163.9**	**163**	164.5	164	165.1
n5000-r0070	**124**	**125.0**	143	149.1	125	125.5

Note. The value is indicated in bold if it is equal to or better than others.

Figure 1. (Color online) Computational Performance Comparison with and Without the Search Space Reduction Strategies



One finds that algorithms with the first strategy converge much faster than those without it, although all the versions show little difference among each other at the early stage. This indicates that the search space reduction strategy (1) helps to improve the efficiency of the proposed algorithm significantly. FVWn2 performs slightly worse than the full-featured FVW. When the third strategy is disabled, the algorithm converges a little slower than the full-featured FVW for some instances. However, for other instances, disabling the third strategy may make the algorithm perform slightly better (instance n5000-r0070). When both the second and third strategies are disabled, the algorithm performs slower than the full-featured FVW, especially when k becomes close to the optimal. This indicates that these strategies help to reduce the search space by tailoring low-quality solutions. In summary, the search space reduction technique is vital for the proposed FVW to obtain promising results.

6. Conclusion

The vertex weighting-based local search algorithm FVW for the minimum connected dominating set problem operates by decomposing the optimization problem into a series of decision problems and solving each of them by incorporating a special vertex weighting technique in a tabu search process. The FVW algorithm is demonstrated to be highly effective in tests on a collection of widely used benchmark instances where FVW is compared with the two best-performing state-of-the-art algorithms RNS-TS and ACO-RVNS. Of 112 public instances, FVW improves the best-known results on 20 problems and matches the best records in the literature for most of the remaining instances (90 problems) while consuming less computational time. For the newly generated 15 sparse instances, FVW outperforms RNS-TS on 13 instances in terms of the best solution found and on all the instances in terms of the average objective value.

These outcomes suggest the relevance of future studies to investigate the potential merit of combining the weighting technique and its associated strategies with the tabu search for solving other optimization problems. Although the techniques proposed in this paper are specific to the minimum connected dominating set problem, most of these ideas can be applied to other combinatorial optimization problems. For example, the vertex weighting technique can be used in local search-based heuristics for other constraint satisfaction problems where most of the neighborhood moves neither improve nor worsen the configuration. The neighborhood reduction technique can be used in problems with large-scale instances or large neighborhoods. However, the implementation of this technique should be treated carefully because it has the potential to impair the completeness of the neighborhood structure, which may cause the algorithm to skip promising moves and thus produce unsatisfying results. Finally, it is interesting to test the proposed ideas in other metaheuristic frameworks with other optimization problems.

Acknowledgments

The authors appreciate Dr. Zhouxing Su and Dr. Yucheng Zhang for discussing the ideas of the paper and their comments and suggestions to improve the paper. The authors are also thankful to the referees for their insightful remarks that have improved the paper significantly.

Appendix. Characteristics of the swap(u, v) Move

In this section, we discuss two characteristics of the swap(u, v) move that are important to ensure FVW will explore the whole solution space.

Proposition A.1. *If there is no feasible candidate* swap(u, v) *operator, the current configuration X is itself a CDS.*

Proof. The proposition is equivalent to its contrapositive, which can be stated as follows: There must exist a swap(u, v) move satisfying all the four properties if the configuration X is not a CDS. In the following, we prove this contrapositive to be true. For the purpose of clarity, we repeat the four properties: Property 1, $u \in X^+$; Property 2, $v \in X \backslash \mathrm{Art}(G(X))$; Property 3, $\exists s \in X \backslash \{v\}, (u,s) \in E$; and Property 4, $\exists (u,s) \in E$ and $s \in X^-$.

The set V of vertices is divided into three subsets X, X^+, and X^-, as shown in Figure A.1. We have the following

Figure A.1. The Set of Vertices Is Divided into X, X^+, and X^-

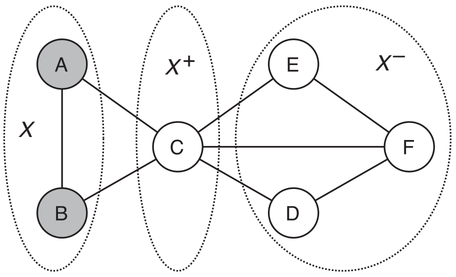
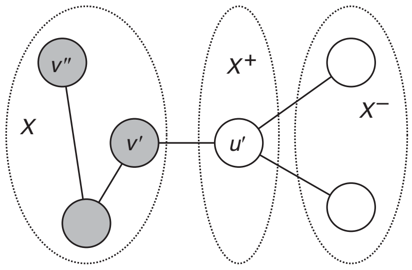


Figure A.2. One Can Always Find Another Nonarticulation Vertex v'' to Replace v'



statements that Properties 1, 4, 2, and 3 can be sequentially satisfied:

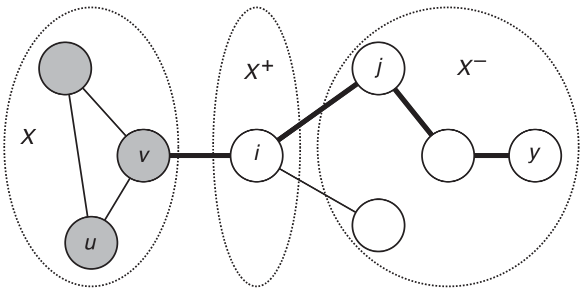
1. If the configuration X is not a CDS, X^+ must be nonempty too. Otherwise, $X^+ = \emptyset$ implies that $X = V$, where the vertex set V is a CDS. Thus, Property 1 can be satisfied.

2. If the configuration X is not a CDS, X^- must be nonempty. Given the condition $X^+ \neq \emptyset$ and $X^- \neq \emptyset$, there must exist $u \in X^+$ connecting to some vertices in X^- because the vertex set V is divided into X, X^+, and X^-, whereas X is not connected to X^-. Thus, Properties 1 and 4 can be satisfied at the same time.

3. Property 2 is independent of Properties 1 and 4. One can always find nonarticulation vertices in a connected configuration X. Thus, Property 2 can be satisfied along with Properties 1 and 4.

4. Finally, we prove that Property 3 can be satisfied along with Properties 1, 2, and 4. Suppose that all pairs of vertices (u, v) satisfying Properties 1, 2, and 4 cannot satisfy Property 3. Then, for each pair (u, v) satisfying Properties 1, 2, and 4, v is the only vertex in X connecting to u. Consider a pair (u', v'), where v' is the only vertex in X connecting to u', which satisfies Properties 1, 2, and 4 but not Property 3. Then, to make the assumption true, it should be impossible to find another nonarticulation vertex $v'' \in X$ that can replace v' to form a pair (u', v'') satisfying Properties 1, 2, and 4 while v'' does not connect to u'. Because Properties 1 and 4 are unrelated to v'', and v' is the only vertex in X connecting to u', Property 2 is the only property to be broken. This indicates that v' is the only nonarticulation vertex in X, which is impossible for any connected graph with more than one vertex, as shown in Figure A.2. Thus, the assumption is not true; that is, there exists a swap(u, v) move satisfying all the four properties in a non-CDS configuration.

Figure A.3. Path $\mathcal{P}_{vy}$, Performing swap(i, u) Decreases the Distance Between y and X



In summary, this establishes that the proposition is true. □

Proposition A.2. *Any vertex* $y \in X^-$ *can be dominated by performing a finite number of* swap(u, v) *moves on any connected configuration* X.

Proof. If $\exists z \in X^+$ for which $(z, y) \in E$, it is obvious that the move swap(z, v) will make y dominated, where v is any nonarticulation vertex in X.

In the following, we demonstrate that the proposition is still true if such a z does not exist for the current configuration. Because G is connected, there exists a path $\mathcal{P}_{vy}$ for any vertex $v \in X$. Suppose that v is the nearest vertex to y in X. Because $v \in X$ and $y \in X^-$, there must exist an edge $(i, j) \in \mathcal{P}_{vy}$ such that $i \in X^+$ and $j \in X^-$, as shown in Figure A.3. By performing the move swap(i, u) where u is any nonarticulation vertex apart from v in X, the distance from vertex y to X must decrease. If we repeatedly perform such moves along the path, y must be finally dominated because the length of the path is finite. □

Endnote

[1] See http://www.ee.washington.edu/research/pstca/ accessed September 29, 2021.

References

Ahn N, Park S (2015) An optimization algorithm for the minimum k-connected m-dominating set problem in wireless sensor networks. *Wireless Networks* 21(3):783–792.

Balasundaram B, Butenko S (2006) Graph domination, coloring and cliques in telecommunications. Resende MGC, Pardalos P, eds. *Handbook of Optimization in Telecommunications* (Springer, New York), 865–890.

Benlic U, Hao JK (2013a) Breakout local search for maximum clique problems. *Comput. Oper. Res.* 40(1):192–206.

Benlic U, Hao JK (2013b) Breakout local search for the max-cut problem. *Engrg. Appl. Artificial Intelligence* 26(3):1162–1173.

Benlic U, Hao JK (2013c) Breakout local search for the quadratic assignment problem. *Appl. Math. Comput.* 219(9):4800–4815.

Bouamama S, Blum C, Fages JG (2019) An algorithm based on ant colony optimization for the minimum connected dominating set problem. *Appl. Soft Comput.* 80(July):672–686.

Buchanan A, Sung J, Butenko S, Pasiliao E (2015) An integer programming approach for fault-tolerant connected dominating sets. *INFORMS J. Comput.* 27(1):178–188.

Burke EK, Bykov Y (2016) An adaptive flex-deluge approach to university exam timetabling. *INFORMS J. Comput.* 28(4):781–794.

Butenko S, Cheng X, Oliveira C, Pardalos P (2004) A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. Butenko S, Murphey R, Pardalos P, eds. *Recent Developments in Cooperative Control and Optimization* (Springer, New York), 61–73.

Cai S, Su K, Sattar A (2011) Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence* 175(9):1672–1696.

Chen S, Ljubić I, Raghavan S (2010) The regenerator location problem. *Networks* 55(3):205–220.

Chen B, Jamieson K, Balakrishnan H, Morris R (2002) Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks* 8(5): 481–494.

Cheng X, Ding M, Du DH, Jia X (2006) Virtual backbone construction in multihop ad hoc wireless networks. *Wireless Comm. Mobile Comput.* 6(2):183–190.

Chinnasamy A, Sivakumar B, Selvakumari P, Suresh A (2019) Minimum connected dominating set based RSU allocation for smartCloud vehicles in VANET. *Cluster Comput.* 22(5):12795–12804.

Deb B, Bhatnagar S, Nath B (2003) Multi-resolution state retrieval in sensor networks. Cayirci E, Znati T, Ekici E, eds. *Proc. First IEEE Internat. Workshop Sensor Network Protocols Appl.* (IEEE, Piscataway, NJ), 19–29.

Ding M, Cheng X, Xue G (2003) Aggregation tree construction in sensor networks. Guizani M, ed. *IEEE 58th Vehicular Tech. Conf.*, vol. 4 (IEEE, Piscataway, NJ), 2168–2172.

Fujie T (2004) The maximum-leaf spanning tree problem: Formulations and facets. *Networks* 43(4):212–223.

Gao C, Yao X, Weise T, Li J (2015) An efficient local search heuristic with row weighting for the unicost set covering problem. *Eur. J. Oper. Res.* 246(3):750–761.

Gendron B, Lucena A, da Cunha AS, Simonetti L (2014) Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. *INFORMS J. Comput.* 26(4):645–657.

Ghaznavi M, Shahriar N, Kamali S, Ahmed R, Boutaba R (2017) Distributed service function chaining. *IEEE J. Selected Areas Comm.* 35(11):2479–2489.

Glover F (2006) Parametric tabu search for mixed integer programs. *Comput. Oper. Res.* 33(9):2449–2494.

Gouveia L, Simonetti L (2017) Spanning trees with a constraint on the number of leaves. a new formulation. *Comput. Oper. Res.* 81(May):257–268.

Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. *Algorithmica* 20(4):374–387.

Hedar A-R, Ismail R, El-Sayed GA, Khayyat KMJ (2019) Two metaheuristics designed to solve the minimum connected dominating set problem for wireless networks design and management. *J. Network Systems Management* 27(3):647–687.

Jovanovic R, Tuba M (2013) Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. *Comput. Sci. Inform. Systems* 10(1):133–149.

Khuller S, Yang S (2019) Revisiting connected dominating sets: An almost optimal local information algorithm. Jansen K, Matthieu C, Rolim J, Umans C, eds. *Algorithmica* 81:2592–2605.

Li B, Cai S, Lin J, Wang Y, Blum C (2020) NuCDS: An efficient local search algorithm for minimum connected dominating set. Bessiere C, ed. *Proc. 29th Internat. Joint Conf. Artificial Intelligence* (International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan), 1503–1510.

Li R, Hu S, Gao J, Zhou Y, Wang Y, Yin M (2017) Grasp for connected dominating set problems. *Neural Comput. Appl.* 28(1):1059–1067.

Li R, Hu S, Liu H, Li R, Ouyang D, Yin M (2019) Multi-start local search algorithm for the minimum connected dominating set problems. *Mathematics* 7(12): Article 1173.

Lucena A, Maculan N, Simonetti L (2010) Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Comput. Management Sci.* 7(3):289–311.

Luo C, Su K, Cai S (2012) Improving local search for random 3-SAT using quantitative configuration checking. De Raedt L, Bessiere C, Dubois D, Doherty P, Frasconi P, eds. *Proc. 20th Eur. Conf. Artificial Intelligence* (IOS Press, Amsterdam), 570–575.

Misra R, Mandal C (2010) Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks. *IEEE Trans. Parallel Distributed Systems* 21(3):292–302.

Morgan M, Grout V (2007) Metaheuristics for wireless network optimisation. Meghanathan N, Collange D, Takasaki Y (co-chairs), eds. *Third Adv. Internat. Conf. Telecomm.* (IEEE, Piscataway, NJ), 1–7.

Peng B, Liu D, Lü Z, Martí R, Ding J (2020) Adaptive memory programming for the dynamic bipartite drawing problem. *Inform. Sci.* 517(May):183–197.

Reis M, Lee O, Usberti F (2015) Flow-based formulation for the maximum leaf spanning tree problem. *Electronic Notes Discrete Math.* 50(December):205–210.

Simonetti L, da Cunha AS, Lucena A (2011) The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm. Pahl J, Reiners T, Voß S, eds. *Network Optimization*, Lecture Notes in Computer Science, vol. 6701 (Springer, Berlin), 162–169.

Subramanyam A, Repoussis PP, Gounaris CE (2020) Robust optimization of a broad class of heterogeneous vehicle routing problems under demand uncertainty. *INFORMS J. Comput.* 32 (3): 661–681.

Tseng YC, Ni SY, Chen YS, Sheu JP (2002) The broadcast storm problem in a mobile ad hoc network. *Wireless Networks* 8(2-3):153–167.

Voudouris C, Tsang EP, Alsheddy A (2010) *Guided Local Search* (Springer, Boston), 321–361.

Wan PJ, Alzoubi KM, Frieder O (2004) Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks Appl.* 9(2):141–149.

Wang L, Wu H, Ding Y, Chen W, Poor HV (2016) Hypergraph-based wireless distributed storage optimization for cellular D2D underlays. *IEEE J. Selected Areas Comm.* 34(10):2650–2666.

Wu J, Dai F (2003) Broadcasting in ad hoc networks based on self-pruning. *Internat. J. Foundations Comput. Sci.* 14(2):201–221.

Wu X, Lü Z, Galinier P (2017) Restricted swap-based neighborhood search for the minimum connected dominating set problem. *Networks* 69(2):222–236.