Expert Systems With Applications 215 (2023) 119338

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A greedy randomized adaptive search procedure (GRASP) for minimum weakly connected dominating set problem

Dangdang Niu [a,b,d,e], Xiaolin Nie [a], Lilin Zhang [a], Hongming Zhang [a,*], Minghao Yin [c]

[a] College of Information Engineering, Northwest A&F University, Yangling, Shaanxi 712100, China
[b] Guangxi Key Lab of Multi-source Information Mining & Security, Guangxi Normal University, Guilin 541004, China
[c] School of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China
[d] Shaanxi Key Laboratory of Agricultural Information Perception and Intelligent Service, Northwest A&F University, Yangling 712100, China
[e] Key Laboratory of Agricultural Internet of Things, Ministry of Agriculture and Rural Affairs, Northwest A&F University, Yangling, Shaanxi 712100, China

ARTICLE INFO

Keywords:
Combinatorial optimization
Weakly connected dominating set
Greedy randomized adaptive search
Heuristics
Local search

ABSTRACT

The minimum weakly connected dominating set problem (MWCDSP), a variant of the classical minimum dominating set problem, aims to find a minimum weakly connected dominating set on a given connected and undirected graph. To address this problem, a 0–1 integer linear programming (ILP) model and a framework of greedy randomized adaptive search procedure (GRASP) for MWCDSP are proposed. Specially, two novel local search procedures are introduced to improve the initial candidate solution in GRASP based on two greedy functions and tabu strategy. First, two greedy functions *Dscore* and *Nscore* are proposed to define the dominating score and neighbor score of each vertex respectively. Second, a tabu strategy is introduced to avoid the cycling problems. Third, a new local search procedure for searching a (k-1)-size solution after obtaining a k-size solution based on the *Dscore* function and tabu strategy is proposed, which is further improved by integrating *Nscore* into the selection process of moving vertices in an improved local search algorithm. Experimental results on four kinds of graph instances show that our GRASPs outperforms the famous CPLEX, the self-stabilizing algorithm MWCDS and the memetic algorithm MA. Specially, the GRASP with improved local search procedure outperforms all of the competitors.

1. Introduction

Consider an undirected, connected graph $G = (V, E)$, where V denotes the set of vertices, E denotes the set of edges. A Dominating Set (DS) is a subset $S \subseteq V$ such that for every vertex $v \in V$, either $v \in S$, or there exist an edge $(u, v) \in E$ for some $u \in S$. If the subgraph induced by a DS is connected, we call it Connected Dominating Set (CDS). The subgraph weakly induced by W ($W \subseteq V$) is defined as $S_w = (N[W], E \cap (W \times N[W]))$, where the set $N[W]$ collects one-hop neighbor of W and itself. The dominating set W is a Weakly Connected Dominating set (WCDS) if the subgraph weakly induced by W is connected. The minimum weakly connected dominating set problem (MWCDSP) is to identify a WCDS of minimum size.

The minimum weakly connected dominating set problem was introduced in 1997 by Dunbar et al. (1997). The concept of WCDS has been widely used for clustering in mobile ad hoc networks and distributed sensor network (Du et al., 2012; Han & Jia, 2007; Pathan & Hong, 2006), which is a better method for clustering than the CDS in general, since a WCDS can be smaller than a CDS and computing a small WCDS is not more complicated than computing a small CDS (Yu et al., 2012). The WCDS is very suitable for cluster formation (Han & Jia, 2007), which was used to reduce the number of cluster heads in the network (Pathan & Hong, 2006) and solve secure clustering problem which plays an important role in distributed sensor networks (Du et al., 2012; Pathan & Hong, 2006). It has been proved that finding the minimum WCDS in an arbitrary graph is a NP-Hard problem (Dunbar et al., 1997). In addition to the computational challenge of NP-Hard complexity for the general graphs, many researches also focus on the computational complexity of MWCDSP for some special kinds of graphs (Domke et al., 2005; Dunbar et al., 1997; Raczek & Cyman, 2019; Sandueta, 2020). These researches further show the theoretical significance of studying MWCDSP.

In order to obtain high-quality WCDS, some self-stabilizing algorithms are specially proposed to constructed minimal WCDS in connected graphs (Ding et al., 2016; Kamei & Kakugawa, 2007; Srimani

* Corresponding author.
E-mail addresses: niudd@nwafu.edu.cn (D. Niu), niexiaolin@nwafu.edu.cn (X. Nie), zhanglilinmg@163.com (L. Zhang), zhm@nwsuaf.edu.cn (H. Zhang), ymh@nenu.edu.cn (M. Yin).

https://doi.org/10.1016/j.eswa.2022.119338
Received 2 April 2022; Received in revised form 28 September 2022; Accepted 20 November 2022
Available online 23 November 2022

& Xu, 2007). Specially, Ding et al. proposed a linear time self-stabilizing algorithm MWCDS for minimal weakly connected dominating sets, which terminates in $O(n)$ steps using a synchronous daemon for an arbitrary connected graph with n nodes (Ding et al., 2016). These self-stabilizing algorithms can quickly construct a minimal WCDS, but it is still difficult to apply them in solving MWCDSP for complex graphs which usually contains many local optimal solutions. Recently, the self-stabilizing memetic algorithm MA has been designed for MWCDSP (Niu & Yin, 2022), in which the self-stabilizing algorithm MWCDS is used to transform the infeasible solutions into minimal feasible solutions in the searching process.

Compared to the minimum dominating set problem and some minimum dominating set variants, where extensive heuristic procedures (Hu et al., 2021; Jovanovic & Tuba, 2013; Li et al., 2017; Pan et al., 2022) and exact algorithms (Abu-Khzam, 2022; Nakkala et al., 2022) have been proposed for efficiently solving graphs in the real world, fewer heuristics or exact algorithms have been developed for MWCDSP. In this work, we aim to fill the gap by proposing the ILP model and designing effective heuristics that find high-quality approximate solutions for MWCDSP. The greedy randomized adaptive search procedure (GRASP) is a meta-heuristic for combinatorial optimization, which was first proposed by Feo and Resende (1989) for set covering problem. From then on, GRASP is continued to study (Feo & Resende, 1995; Pitsoulis & Resende, 2002; Resende & Ribeiro, 2010) and is widely applied to obtaining near-optimal solutions in many other problems (Cravo & Amaral, 2019; Ferdi & Layeb, 2018; Li et al., 2017; Uribe et al., 2021). The above discussions motivate us to design efficient GRASP for MWCDSP, which is a theoretically feasible way to efficiently solve MWCDSP.

In this paper, we present an ILP model which is an exact solution for MWCDSP, and a framework of GRASP for MWCDSP. In our framework of GRASP, the *ConstructionGreedyRandomized* procedure is used to generate an initial WCDS, which will be proposed to the local search procedure as initial candidate in the framework of GRASP. Two greedy functions *Dscore* and *Nscore* are introduced, which provide the selection basis of moving vertices. We also introduce two local search procedures based on the above two greedy functions and the tabu strategy. The first local search procedure searches (k-1)-size WCDS after a k-size WCDS is found. The above local search procedure is easy to fall into local optimum, so we proposed an improved local search procedure based on different strategies for adding vertices to and removing vertices from candidate solution. Experimentally, our GRASPs are competitive with the famous CPLEX, the self-stabilizing algorithm MWCDS and the memetic algorithm MA, and the GRASP with the improved local search procedure is the best solver.

We summarize the main contributions of this work as follows. First, an ILP model is designed for MWCDSP, which can provide exact MWCDSP solutions for the input small graphs. Second, a GRASP framework is introduced for MWCDSP, and two special local search algorithms used in GRASP are designed for MWCDSP. Extensive experiments have verified the great efficiency of our designed GRASPs, which can be used to provide reference for more other connected dominating set problems. Third, we assess the GRASP algorithms on 24 randomly generated small graphs and 74 conventional benchmark instances from the literatures. The solutions of the above benchmarks are presented, which can be used to assess future MWCDSP algorithms.

The reminder of this paper is organized as follows: Section 2 presents some preliminary knowledge and the ILP model for MWCDSP. Section 3 presents the framework and the concrete details of GRASP for the MWCDSP, in which two novel local search procedures are proposed. The experimental results are presented in Section 4. The statistical results and the discussion corresponding to them are presented in Section 5. Finally, Section 6 gives our conclusions and future works.

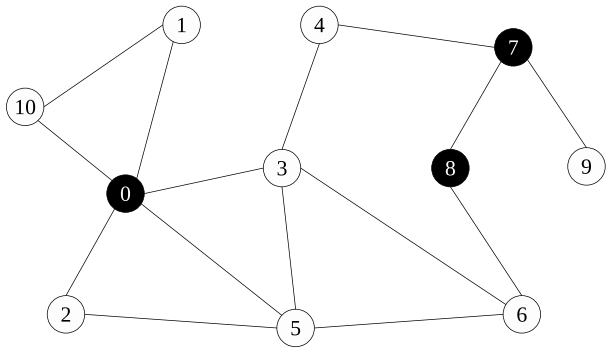

Fig. 1. An example of dominating set on the graph G.

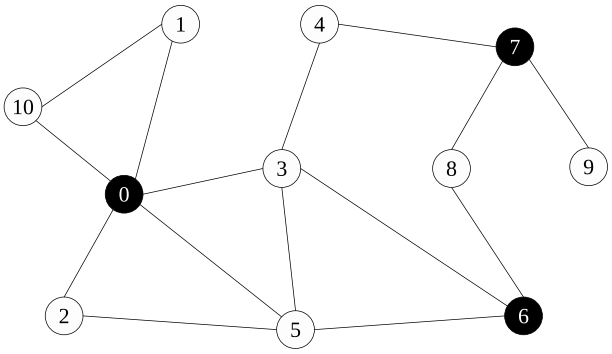

Fig. 2. An example of weakly connected dominating set on the graph G.

2. Preliminaries

2.1. Basic definitions

Consider an undirected, connected graph $G = (V, E)$, where V denotes the set of vertices, E denotes the set of edges. We shall use $N(v)$ to represent the neighbor vertex set of v. We also denote $N[v] = N(v) \cup \{v\}$. The shortest hop path from u to v is denoted as $hop(u, v)$. Then, the neighboring vertices with hop paths in [1, i] of the vertex v are collected in $N_i(v) = \{u \mid 1 \leq hop(u, v) \leq i\}$. In addition, we also denote $N_i[v] = N_i(v) \cup v$. We shall extend the above concepts to the vertex set. Given a vertex set $VS \subseteq V$, the neighboring vertices with hop paths in $[1, i]$ of the vertex set VS is presented as $N_i(VS) = \{u \mid 1 \leq hop(u, v) \leq i, v \in VS\}$. In addition, we denote $N_i[VS] = N_i(VS) \cup VS$. The dominating vertex set of a given vertex set S is denoted as $dom(S) = N[S]$, which collects the vertices dominated by VS. Some important definitions are described as follows.

Definition 1 (*Weakly Induced Subgraph, WIS*)**.** Given an undirected graph $G = (V, E)$, $WIS_G(W) = (N[W], E_1)$, $W \subseteq V$, $E_1 = \{E \cap (W \times N[W])\}$, subgraph $WIS_G(W)$ is called the weakly induced subgraph of graph G based on W.

Definition 2 (*Weakly Connected Dominating Set, WCDS*)**.** Given an undirected, connected graph $G = (V, E)$, the weakly connected dominating set W of G is a dominating set such that $WIS_G(W)$ is connected.

Given an undirected, connected graph $G = (V, E)$, the Minimum Weakly Connected Dominating Set Problem (MWCDSP) is to find a weakly connected dominating set W of G with minimum cardinality.

Figs. 1 and 2 show an undirected graph $G = (V, E)$ with 11 vertices and 14 edges. Obviously, The black vertices in the set $D = \{0, 7, 8\}$ is a dominating set of G in Fig. 1, but the weakly induced graph of G based on D are not connected. In Fig. 2, the black vertices in the set $W =$

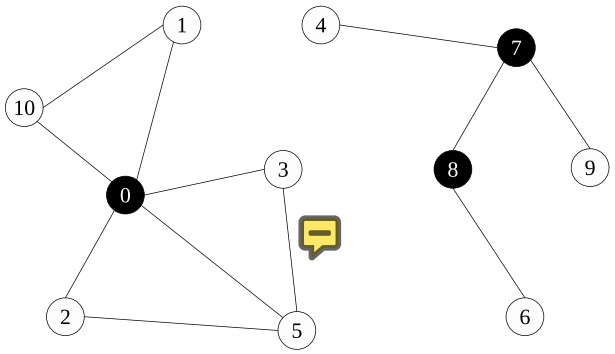

Fig. 3. The weakly induced graph $(N[D], E \cap (D \times N[D]))$ of G by $D = \{0, 7, 8\}$.

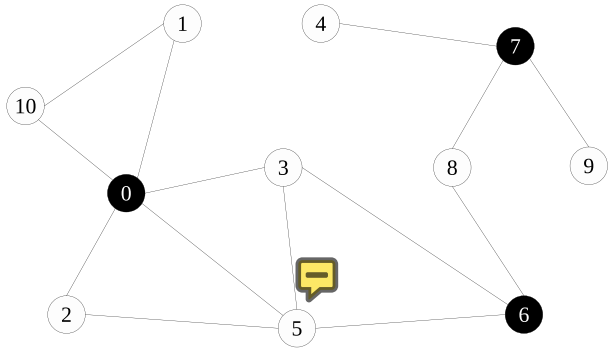

Fig. 4. The weakly induced graph $(N[W], E \cap (W \times N[W]))$ of G by $W = \{0, 6, 7\}$.

{0, 6, 7} are in a weakly connected dominating set, which means that the weakly induced graph of G based on W is connected. The weakly induced graphs of G based on D and W are shown as Figs. 3 and 4 respectively.

According to Definition 1, the weakly induced graph $(N[D], E \cap (D \times N[D]))$ by D is show in Fig. 3, and the weakly induced graph $(N[W], E \cap (W \times N[W]))$ by W is show in Fig. 4. Obviously, the graph in Fig. 3 is not connected, but the graph in Fig. 4 is connected. All vertices not in W are dominated by W, so W is a weakly connected dominating set according to Definition 2.

Now, we introduce two greedy functions for selecting moving vertices in the local search procedure, which can provide comparison standards when algorithms intend to select some vertices.

Dscore: For a vertex v, $Dscore(v)$ indicates how many vertices will become non-dominated from dominated once v's status is changed (added to or removed from the solution). That is to say, $Dscore(v) = |N[S_v]| - |N[S]|$, in which S is the current solution and S_v is the solution after changing v's status. From the above definition, $Dscore(v)$ is a non-negative number when v is added to the current solution S, and $Dscore(v)$ is a non-positive number when v is removed from the current solution S.

Nscore: For a vertex v, $Nscore(v)$ indicates how many neighbors of v are in the current solution S. That is to say, $Nscore(v) = |N[v] \cap S|$. $Nscore(v)$ can potentially reflect the influence on connectivity of S when removing v from the current solution.

2.2. ILP model for the MWCDSP

We also describe a 0-1 integer linear programming (ILP) model for the MWCDSP in this section. Our constructing method of ILP model for MWCDSP is mainly derived from the ILP model for minimum dominating tree problem (MDTP) in Shin et al. (2010). Obviously, if all weights of edge are unified, the solutions of MDTP are the spanning trees of the minimum connected dominating sets. In order to construct ILP model for MWCDSP, we add a 2-dimensional adjacency matrix E_{weak} to collect all edges which connect V and $N_2(V)$, which means that for any vertex $v \in V$, there are edges between v and each vertex of $N_2(v)$ in E_{weak}. For any vertex pair $(u, v)(u, v \in V)$, if there exists an edge between them in E_{weak}, $E_{weak}[u, v] = 1$, $E_{weak}[u, v] = 0$ otherwise. Now, we define the following decision variables:

- Let $s = \{E_1, E_2, \ldots, E_k\}$ $(\forall E_i \subseteq E_{weak})$ be a candidate solution.
- $\forall x_{uv} \in \{0, 1\}$ $(u, v \in V)$ is a binary decision variable that is equal to 1 if and only if (u, v) is selected in the optimal solution, 0 otherwise.
- $\forall y_t \in \{0, 1\}$ $(t \in V)$ is a binary decision variable that is equal to 1 if and only if t is selected in the optimal solution, 0 otherwise.

We obtain the following ILP model for the WSCP:

$$f(s) = 1 + \sum_{u,v \in V} x_{uv} \tag{1}$$

$$\begin{cases} x_{uv} \le E_{weak}[u, v], & \forall u, v \in V \quad \text{(a)} \\ x_{uv} = x_{vu}, & \forall u, v \in V \quad \text{(b)} \\ y_u + y_v \ge 2 \times x_{uv}, & \forall u, v \in V \quad \text{(c)} \\ \sum_{u,v \in S} x_{uv} \le |S| - 1, & S \subset V \,\&\, \sum_{u,v \in S} E_{weak}[u, v] \\ & > |S| - 1 \,\&\, |S| \le UB \quad \text{(d)} \\ \sum_{u,v \in V} x_{uv} = \sum_{t \in V} y_t - 1 & \quad \text{(e)} \\ \sum_{t \in N[v]} y_t \ge 1, & \forall v \in V \quad \text{(f)} \end{cases} \tag{2}$$

The objective function of formula (1) is to minimize the sum of vertices in the weakly connected dominating set. In formula (1), the spanning tree corresponding to the weakly connected dominating set consists of all edges with $x_{uv} = 1$. Since the sum of vertices in a connected graph is the sum of edges in its spanning tree plus 1, we add 1 to $\sum_{u,v \in V} x_{uv}$ in formula (1). In formula (2), the first two constraints (2)(a) and (b) ensure that the set of edges in any solution is a subset of the edges in E_{weak}. The third constraint (2)(c) shows the relation between vertices and edges in optimal solution, that is, for each edge selected in optimal solution, the end vertices have to be selected. The fourth constraint (2)(d) and fifth constraint (2)(e) are similar to the formulation in Minimum Spanning Tree problem to guarantee that the solution is a tree. In order to reduce the number of constraints, we add two additional conditions to select subsets of V in the (2)(d), which are different from Shin et al. (2010). The first condition $\sum_{u,v \in S} E_{weak}[u, v] > |S| - 1$ can delete some subsets of V, in which the fourth constraint (2)(d) has been satisfied. The second condition $|S| \le UB$ indicates that the scale of each selected subset of V should not be greater than the upper bound (UB in this condition) of this problem. The parameter UB will be set as the best solutions found by our proposed GRASP for each graph in the experiments. The last constraint (2)(f) shows the constraint of the dominating set. Basically, for each vertex v, at least one of its neighbors or itself has to be selected to the optimal solution.

We also summarize notations frequently used in this paper in Table 1.

3. GRASP for minimum weakly connected dominating set problem

3.1. Framework of GRASP

The Greedy randomized adaptive search procedure (GRASP) is a multi-start metaheuristic approach, which includes two procedures: a *ConstructionGreedyRandomized* procedure to build a feasible solution and a local search procedure to address some combinatorial optimization problems. In each iteration of GRASP, a greedy randomized candidate solution is constructed firstly, and then the solution is regarded as a starting current candidate solution for local search procedure. The local search procedure investigates the neighborhood of current

Table 1
Notations and meanings.

Notation	Meaning
$G = (V, E)$	An undirected, connected graph
DS	Dominating set
CDS	Connected dominating set
$WCDS$	Weakly connected dominating set
$MWCDSP$	Minimum weakly connected dominating set problem
$N(v)(resp.\ N(VS))$	The neighbor vertex set of v $(resp.\ VS, VS \subseteq V)$
$N[v](resp.\ N[VS])$	$N(v) \cup \{v\}$ $(resp.\ N(VS) \cup VS)$
$N_i(v)(resp.\ N_i(VS))$	$\{u \mid 1 \le hop(u,v) \le i\}(resp.\ \{u \mid 1 \le hop(u,v) \le i, v \in VS\})$
$N_i[v](resp.\ N_i[VS])$	$N_i(v) \cup \{v\}$ $(resp.\ N_i(VS) \cup VS)$
$Dscore(v)$	$\lvert N[S_v]\rvert - \lvert N[S]\rvert$, S_v is a solution obtained by changing the status of the vertex v in S
$Nscore(v)$	$\lvert N[v] \cap S\rvert$, S is the current solution
E_{weak}	A adjacency matrix which collects all edges which connect V and $N_2(V)$
SWN	$SWN = N_2(W)$ if W is non-empty, $SWN = V$ otherwise, in which W is the current solution
$SWCE$	Collects the vertices with the most times appear in the two level neighbors of each weakly connected component
RCL	$\{v \mid Dscore(v) \ge \alpha \times (Dscore_{max} - Dscore_{min}), v \in SWN\}$, $0 \le \alpha \le 1$

candidate solution and replaced it with a better solution until the stop criterion is met. The above two procedures are repeated many times either independently or following some learning mechanisms, and the best optimal solution found by all iterations is then chosen as the final solution. For further understanding and references, the readers could read the paper (Resende & Ribeiro, 2010) to be useful.

Algorithm 1 GRASP

Input: an undirected, connected graph $G = (V, E)$
Output: The best weakly connected dominating set W^*

```
1: Initialize the best solution W*;
2: while elapsed time < cutoff do
3:     W ← ConstructionGreedyRandomized();
4:     W ← Localsearch(W, ITER_NUM);
5:     if |W| < |W*| then
6:         W* ← W;
7:     end if
8: end while
9: return W*;
```

The main framework of GRASP, which is used for solving the MWCDSP, is illustrated in Algorithm 1. The main loop of the GRASP consists of lines 2–8. An initial WCDS is generated by the *ConstructionGreedyRandomized*() with some greedy and randomized strategies in line 3, and improved by the local search procedure in line 4. The *ITER_NUM* is the iteration number of local search.

In this section, we design efficient GRASP to solve the MWCDSP. Two procedures are employed in GRASP. In the first procedure, an initial WCDS is constructed based on some greedy and randomized strategies. After that, a local search procedure is called to improve the initial solution.

3.2. ConstructionGreedyRandomized procedure

In order to make a balance between the random strategy and greedy strategy, we build the restricted candidate list (*RCL*) in this paper, from which the vertices of initial solution are selected. Our *ConstructionGreedyRandomized* procedure builds the *RCL* based on a threshold parameter α which was first proposed by Feo and Resende (1989). Given an undirected, connected graph $G = (V, E)$, W is the current solution, $Dscore_{max}$ and $Dscore_{min}$ are the maximum value and minimum value in $\{Dscore(v) \mid v \in V\text{-}W\}$ respectively. The building method of *RCL* in our *ConstructionGreedyRandomized* procedure is shown as formula (3).

$$RCL \leftarrow \{v \mid Dscore(v) \ge \alpha \times (Dscore_{max} - Dscore_{min}), v \in SWN\} \quad (3)$$

In Formula (3), if W is not empty, SWN collects the vertices which are in the two level neighbor of W, i.e., $SWN = N_2(W)$. If W is empty, SWN collects all vertices in V. Obviously, the weakly connectedness of the candidate solution is not destroyed by adding any vertex in SWN. Formula (3) makes a balance between the random strategy and greedy strategy. Specially, the *ConstructionGreedyRandomized* procedure falls to a pure random procedure if α is set to 0; on the other hand, the *ConstructionGreedyRandomized* procedure falls to a pure greedy procedure if α is set to 1 (Feo & Resende, 1989). In our algorithm, α is set to be 0.8.

Based on above discussion, the *ConstructionGreedyRandomized* procedure is introduced in Algorithm 2.

Algorithm 2 *ConstructionGreedyRandomized*

Input: an undirected, connected graph $G = (V, E)$
Output: an initial weakly connected dominating set W

```
1: Initialize Dscore(v) = |N(v)|, for each v∈V;
2: W←∅;
3: RCL←∅;
4: SWN←V;
5: while W is not a WCDS do
6:     Dscore_max←max{Dscore(v) | v∈SWN};
7:     Dscore_min←min{Dscore(v) | v∈SWN};
8:     RCL←{v | Dscore(v) ≥α×(Dscore_max - Dscore_min), v ∈ SWN};
9:     w←select vertex w randomly from RCL;
10:    W←W∪{w};
11:    SWN←N2(W);
12: end while
13: return W;
```

At the beginning of *ConstructionGreedyRandomized* procedure, the *Dscore* function, the solution W, the *RCL* and the *SWN* are initialized respectively in lines 1–4. In the main loop of *ConstructionGreedyRandomized* procedure, the maximum *Dscore* and minimum *Dscore* of the vertices are first obtained in lines 6–7. Based on that, the *RCL* is constructed according to Formula (3) in line 8. Then, a vertex is selected randomly from the *RCL* to build the initial weakly connected dominating set in line 9. The *SWN* is updated after adding a new vertex to the solution in lines 10–11. The iteration will continue until W is a dominating set, and Algorithm 2 returns the result in line 13.

3.3. Local search procedure in GRASP for MWCDSP

According to Algorithm 1, after an initial solution is constructed by *ConstructionGreedyRandomized* procedure, a local search procedure is exploited to further improve it. In this paper, we integrate a tabu strategy to the local search framework for searching possible better solution. Besides, so as to make sure that the candidate solution maintains weakly connectedness during the whole search, a vertex set of solution weakly connected elements (*SWCE*) is presented, which is used to provide selected vertices for adding to the candidate solution.

3.3.1. Tabu strategy

In order to deal with the cycling problem of local search, many literatures have proposed some effective strategies. Tabu strategy is one of the above strategies and used to forbid reversing the recent changes. Tabu strategy has been used to significantly improve the corresponding local search procedures in many combinatorial optimization problems solving, such as minimum weight vertex cover problem (Cai et al., 2019), minimum connected dominating set problem (Li et al.,

2017), minimum weight dominating set (Wang, Cai et al., 2018; Wang et al., 2017), maximum weight clique problem (Wang et al., 2020) etc. Usually, a tabu list is introduced, and the statuses of elements in the tabu list are not allowed to be changed within the tabu tenure. In our algorithm, to be more efficient, the tabu list consists of only one vertex, and the tabu tenure is equal to 1. In other words, during the local search procedure, the tabu strategy only forbids removing the added vertex immediately. The above method is also used in Li et al. (2017).

3.3.2. Solution weakly connect elements (SWCE)

If the candidate solution is not weakly connected, we maintain a vertex set of solution weakly connected elements (*SWCE*) to make a candidate solution weakly connected. Different from connected component, a weakly connected component of a non-weakly-connected graph G is a maximal weakly connected subgraph of G. *SWCE* maintains the vertices with the most times appear in the two level neighbors of each weakly connected component. Assuming that the candidate $W = W_1 \cup \cdots \cup W_k$, in which $W_1, \ldots, W_k$ are k weakly connected components, then $SWCE = \{v | sum\_A(v) = max\{sum\_A(n) = \sum_{i=1\ldots k} A(n,i), n \in V - W\}, v \in V - W\}$, in which $A(n,i) = 1$ if n appears in the $N_2(W_i)$, $A(n,i) = 0$ otherwise.

Specifically, if the number of times that a vertex appears in the two level neighbors of each weakly connected component is equal to the number of weakly connected components, the candidate solution will be weakly connected after adding this vertex into it. If no the above vertex exists, we add a vertex v with most times that appearing in the two level neighbors of each weakly connected component, i.e. $v \in SWCE$, to the candidate solution. The above operation can quickly reduce the number of weakly connected components of the candidate solution.

3.3.3. Local search procedure

The local search procedure works as Algorithm 3. After obtaining a WCDS W with k vertices, the local search procedure removes one vertex from the candidate solution and goes on to search a weakly connected dominating set with k-1 vertices. Based on the above process, the core of the local search procedures is searching for a WCDS with fixed number of vertices. To find a WCDS with k vertices, the procedure iteratively exchanges two vertices in k-sized candidate solution and *SWCE* respectively until the candidate solution is a WCDS. It first selects a vertex form candidate solution and removes it. After that, it selects a vertex from *SWCE* and adds this vertex to the candidate solution.

The local search procedure executes the main loop in lines 2–15 until the given maximum iteration number *ITER_NUM* is reached. During the search procedure, once all vertices are dominated by W and W is weakly connected, which means that W is a WCDS, the procedure tries to search a better solution by removing one vertex from the candidate solution and updates the best solution R with W in lines 3–7. If the best solution is not updated, an infeasible solution with $|R| - 1$ vertices is obtained after executing the loop in lines 3–7. In order to maintain the vertex number of candidate solution, local search procedure in Algorithm 3 swaps two vertices in W and *SWCE* respectively in lines 8–14. It first selects a vertex w with greatest *Dscore* in W, which is not in the tabu list, to remove, breaking ties randomly in lines 8–9. After removing w, the *SWCE* is updated in line 10. Then the procedure randomly selects a vertex $v \in SWCE$ with the greatest score value, breaking ties in favor of vertices with small *time_stamp*, and adds it into W in lines 11–12. In Algorithm 3, *time_stamp* of any vertex v is the number of the last iteration of the loop in line 2, in which v' status is changed. Along with exchanging the two selected vertices, the tabu vertex and *SWCE* set are updated accordingly in lines 13 and 14. When the given maximum iteration number *ITER_NUM* in line 2 is reached, the local search procedure in Algorithm 3 returns the best solution of MWCDSP found in this execution (line 16).

For each step in the local search phase (Line 3–14) in Algorithm 3, the complexity is $\max\{|W|^2 \times \triangle(G), |W| \times \triangle(G)^2\}$, where $\triangle(G)$ is the maximum vertex degree of G and $O(\triangle(G)^2)$ is the worst time complexity of finding the two level neighbor vertices of w.

Algorithm 3 *Local search*

Input: an initial weakly connected dominating set W, and the maximum iteration number *ITEM_NUM*
Output: a searched better weakly connected dominating set W

1: $R \leftarrow W$;
2: **for** $it = 0$; $it < ITER\_NUM$; $it{+}{+}$ **do**
3: **while** W is a WCDS **do**
4: $R \leftarrow W$;
5: $w \leftarrow$select w from W with the greatest *Dscore*, breaking ties randomly;
6: $W \leftarrow W \backslash w$;
7: **end while**
8: $w \leftarrow$select w from W with the greatest *Dscore* and w is not the *tabu_vertex*, breaking ties randomly;
9: $W \leftarrow W \backslash w$;
10: update *SWCE*;
11: $v \leftarrow$randomly select v from *SWCE* with the greatest *Dscore* value, breaking ties in favor of vertices with small *time_stamp*;
12: $W \leftarrow W \cup \{v\}$;
13: *tabu_vertex* $= v$;
14: update *SWCE*;
15: **end for**
16: **return** R;

3.4. Improved local search procedure

In the local search procedure of Algorithm 3, it searches a WCDS of size k-1 after finding a WCDS of size k based on exchanging two vertices in current candidate solution and *SWCE*. The above searching process is easy to fall into local optimum, since the size of target WCDS in current iteration is set to a fixed number k-1 and exchanging only two vertices happens in once iteration. On the one hand, only two vertices are selected to exchange them for some candidate solution W in one iteration, which means that only $|W| \times (|V| - |W|)$ (V is the collection of vertices in the input graph) combination schemes are provided for exploring the neighborhood solutions. Intuitively, polynomial combination selections in each iteration potentially increase the risk of cycling problem of local search. However, if any number of vertices are selected to exchange them in one iteration, it will create exponential permutation selections in each iteration. On the other hand, Algorithm 3 selects some vertex to change the current solution based on the *Dscore* values of all vertices. For the current solution, some vertex v is added to or remove from it only when v has the largest *Dscore* value. From the definition of *Dscore*, if it wants to increase the possibility of selecting some vertex v, neighbor vertices of v should be removed from or added to the current solution as many as possible. Therefore, only exchanging two vertices in each iteration reduces the mobility of vertices, and some vertex may be never removed from or added to the current solution. If any number of vertices are allowed to continuously removed from or added to the current solution, the vertices with very small *Dscore* will have more opportunities to be selected. Based on the above statements, the vertex exchange method in Algorithm 3 will make the local search easily fall into local optimization.

In order to searching more possible solution subspaces, we introduce two strategies for adding vertices into candidate solution and removing vertices from candidate solution, and an improved local search procedure is described as Algorithm 4. In Algorithm 4, the two restraint conditions that searching a WCDS with fixed vertex number and exchanging two vertices at a time in Algorithm 3 are broken.

Algorithm 4 *Improved local search*

Input: an initial weakly connected dominating set W, and the maximum iteration number *ITEM_NUM*
Output: a searched better weakly connected dominating set W

1: $R \leftarrow W$;
2: **for** $it = 0$; $it < ITER\_NUM$; it++ **do**
3: **while** W is a WCDS **do**
4: update R, if $|W| < |R|$;
5: **if** *with probability* $p < wp$ **then**
6: $w \leftarrow$select w from W with the lowest *Nscore*, breaking ties randomly;
7: **else**
8: $w \leftarrow$select w from W with the greatest *Dscore*, breaking ties randomly;
9: **end if**
10: $W \leftarrow W \backslash w$;
11: **end while**
12: $w \leftarrow$select w from W with the greatest *Dscore* and w is not the *tabu_vertex*, breaking ties randomly;
13: $W \leftarrow W \backslash w$;
14: $SWN \leftarrow N_2(W)$;
15: **while** there are non-dominated vertex **do**
16: $v \leftarrow$select v from *SWN* with the greatest *Dscore*, breaking ties randomly;
17: $W \leftarrow W \cup \{v\}$;
18: $SWN \leftarrow N_2(W)$;
19: **end while**
20: update *SWCE*;
21: **if** W is not a WCDS **then**
22: $v \leftarrow$randomly select v from *SWCE* with the greatest *Dscore* value, breaking ties in favor of vertices with small *time_stamp*;
23: $W \leftarrow W \cup \{v\}$;
24: $tabu\_vertex = v$;
25: update *SWCE*;
26: **end if**
27: **end for**
28: **return** R;

In Algorithm 4, two new processes of adding vertices into candidate solution and deleting vertices from candidate solution are introduced. Usually, removing the vertex v with lowest *Nscore* does not impact the weakly connectivity of the candidate solution with more probabilistically; and removing the vertex v with greatest *Dscore* can make the candidate solution dominate more vertices. So, in order to make a balance between weakly connectivity and dominating more vertices of the candidate solution, Algorithm 4 removes a vertex based on the probability p in line 5. If $p < wp$, it removes a vertex w with the lowest *Nscore* value from W, breaking ties by randomly in line 6; otherwise a vertex w with the greatest *Dscore* value is selected to remove from W, breaking ties by randomly in lines 8. In order to search more solution subspaces, Algorithm 4 deletes an extra vertex in lines 12–13. Since the candidate solution is changed by the operations in lines 3–13, the *SWN* is updated in line 14. While the candidate solution is not a dominating set, Algorithm 4 selects the vertex with biggest *Dscore* from *SWN* and adds it to the candidate solution until all vertices are dominated in lines 15–19. If the candidate solution is not weakly connected, Algorithm 4 selects a vertex from *SWCE* and adds it to the candidate solution in lines 22–23 just like Algorithm 3, and updates the *tabu_vertex* and *SWCE* in lines 24–25. The above processes mean that we can add one or more vertices into the candidate solution in one iteration.

For each step in the local search phase (Lines 3–26) in Algorithm 4, the complexity is $\max\{|W|^2 \times \triangle(G), |V| \times \triangle(G)^2\}$.

4. Experimental results

In this section, we present the experimental results obtained with the GRASP based on the basic local search procedure in Algorithm 3 (abbreviated as GRASP) and the GRASP based on the improved local search procedure in Algorithm 4 (abbreviated as GRASP_impLS).

4.1. Benchmark instances

There are four benchmarks selected in our experiments, including random UDG benchmarks, LPNMR'09 benchmarks, Common UDG benchmarks and NDR benchmarks. The benchmarks are introduced as follows.

- Random UDG benchmarks (24 instances)[1]: This group of benchmarks is randomly generated with the method used in Jovanovic and Tuba (2013), which is derived from ad hoc network clustering problems. The ad hoc network clustering problems are the mainly important application scenarios of MWCDSP, which are also used to test MWCDS in Ding et al. (2016). These benchmarks have small scale, which are mainly used to test the solving ability of CPLEX.
- LPNMR'09 benchmarks (9 instances)[2]: This group of benchmarks has been used on the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09), we select the problem instances that have had a satisfactory solution (the solution is known) given in the LPNMR benchmark, just like (Jovanovic & Tuba, 2013; Li et al., 2017).
- Common UDG benchmarks (41 instances): This group of benchmarks is provided by Jovanovic and Tuba in Jovanovic and Tuba (2013), which is derived from ad hoc network clustering problems. These benchmarks are also used in Jovanovic and Tuba (2013) and Li et al. (2017).
- NDR benchmarks (24 instances)[3]: These were transformed from the unweighted and connected graphs with 200-21498 vertices in Network Data Repository (NDR) online (Rossi & Ahmed, 2015), which is used to test the solvers' ability to solve large graphs.

4.2. Experiment preliminaries

In this section, we perform intensive experiments to evaluate the proposed GRASP and GRASP_impLS for solving MWCDSP. We compare GRASP and GRASP_impLS with the famous CPLEX, a self-stabilizing algorithm named MWCDS (Ding et al., 2016) and the memetic algorithm MA (Niu & Yin, 2022) on 98 benchmark instances.

CPLEX is a general MIP solver, which is widely used to solve many important constrained optimization problems, such as weighted vertex coloring problem (Sun et al., 2018), weighted sum coloring problem (Niu et al., 2021), sum coloring problem (Wang et al., 2013), maximum set k-covering problem (Wang, Ouyang et al., 2018). The version of CPLEX used in our experiments is v12.9, which is used to solve the ILP model for the MWCDSP in Section 2.2.

MWCDS is a linear time self-stabilizing algorithm for solving minimal weakly connected dominating set problem, which terminates in $O(n)$ steps by using a synchronous demon (Ding et al., 2016). The synchronous demon selects all the privileged nodes to move at each step, for any connected graph with n nodes. The solutions obtained by MWCDS are decided by the random initial solutions, so we run MWCDS 10 times independently for each instance with different random seeds in our experiments.

MA is a memetic algorithm for solving MWCDSP, in which the self-stabilizing algorithm MWCDS is used to transform the infeasible solutions created by the operations of initializing populations, crossover

[1] https://github.com/niudd304/GRASP_WCDS.git.
[2] http://dtai.cs.kuleuven.be/events/ASP-competition/encodings.shtml.
[3] https://networkrepository.com/networks.php.

Table 2
Parameter settings.

Parameter	Value
Running times of MWCDS, GRASP or GRASP_impLS for each instance	10
Time limit of MA, GRASP or GRASP_impLS	300 s
Time limit of CPLEX	3600 s
α in *RCL* of GRASP or GRASP_impLS	0.8
wp used in the selection of vertices in Algorithm 4	0.35

and mutation etc., into minimal feasible solutions (Niu & Yin, 2022). After a solution is obtained by the self-stabilizing algorithm, a simple local search algorithm is used to refine it in MA.

Our proposed algorithms are implemented in the C++ programming language. All computational experiments were performed on the Linux Ubuntu with Intel(R) Core(TM) i5-6600 CPU @3.30 GHz and 8 GB memory. The memetic algorithm MA and our proposed algorithms GRASP and GRASP_impLS run 10 times independently for each instance with different random seeds, until the time limit (300 s) is satisfied. The time limit with 300 s represents the single execution time of MA, GRASP or GRASP_impLS with some random seed on each instance. In our algorithms, the two important parameters α in *RCL* and *wp* used in Algorithm 4 are set to be 0.8 and 0.35 respectively. The running time of CPLEX is limited to 3600 s, which refers to the cutoff limit of CPLEX in Niu et al. (2021) and Sun et al. (2018). All of the above parameter settings are also listed in Table 2.

We use $|V|$ and $|E|$ to represent the sizes of vertices and edges in any graph $G = (V, E)$ respectively. For the UDG benchmarks, the 'Nodes' represents the given number of vertices in the input graph, so the $|V|$ values are omitted in the corresponding tables (Tables 5 and 7). For MWCDS, 'Best' denotes the value of best solution found, 'Avg' denotes the average value of the solutions obtained in 10 runs, and 'Time' denotes the actually running time for obtain the solution. For MA, GRASP and GRASP_impLS, we also use 'Best' and 'Avg' to denote the best value of found solutions and the average value of found solutions, respectively. Moreover, the average execution time of MA, GRASP and GRASP_impLS is denoted as 'AETs', which is 300 s for each instance. For CPLEX, the 'Solu' value is the solution for the corresponding instance, and 'Stat' denotes the solving status for the corresponding instance. If the corresponding instance is completely solved by CPLEX within 3600 s, the 'Stat' value is equal to the overall running time of CPLEX. The 'stat' value and the 'Solu' value are marked as 'N/A', when the algorithm fails to provide a solution within the given time limit for the corresponding graph. If the CPLEX is not terminated within 3600 s, but a solution is provided, the 'Stat' value of CPLEX is marked as 'Feasible', which means that the optimality of this solution is not proved by CPLEX within the 3600 s of execution time.

4.3. Parameter tuning

In this section, we mainly test the influences of the parameters α in *RCL* and the parameter *wp* in improved local search procedure, on the performance of GRASP_impLS. To support this test, 41 common UDG benchmarks and 19 instances selected from the NDR benchmarks in Section 4.1 are taken into account. Therefore, 60 instances for parameter tuning are collected. Also, some possible values of the α and *wp* are chosen, which are used to replace them in GRASP_impLS, and six competitors are obtained. In GRASP_impLS, α and *wp* are set as 0.8 and 0.35, respectively. In each of the above six competitors, only one parameter is changed with respect to GRASP_impLS. In order to comprehensively test the effects of different α values and *wp* values for GRASP_impLS, we compare GRASP_impLS with its competitors for their solution quality, stability and efficiency. The above three comparative aspects can be evaluated by the 'Best' values and 'Avg' values of solvers for the selected 60 instances, respectively. The competitive results of GRASP_impLS with the above competitors for the 'Best' values and 'Avg' values are shown in Tables 3 and 4, respectively.

Table 3
Parameter tuning of α and wp for the 'BEST' values.

Result	$wp = 0.35$			$\alpha = 0.8$		
	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.9$	$wp = 0.15$	$wp = 0.55$	$wp = 0.75$
# Better_ins	13	15	13	28	10	11
# Worse_ins	4	5	4	1	6	8
# Equal_ins	43	40	43	31	44	41

Table 4
Parameter tuning of α and wp for the 'AVG' values.

Result	$wp = 0.35$			$\alpha = 0.8$		
	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.9$	$wp = 0.15$	$wp = 0.55$	$wp = 0.75$
# Better_ins	23	20	20	37	15	14
# Worse_ins	6	8	7	1	11	12
# Equal_ins	31	32	33	22	34	34

In Tables 3 and 4, the '#Better_ins' value is the number of instances for which GRASP_impLS outperforms the competitor i.e. GRASP_impLS with the corresponding parameter setting in horizontal axis. Accordingly, the '#Worse_ins' value is the number of instances for which GRASP_impLS is not competitive with the competitor, and '#Equal_ins' value is the number of instances for which GRASP_impLS has the same evaluation values in the corresponding table with the competitor. In Tables 3 and 4, the sum of numbers in each column is equal to 60.

From Table 3, one can see that GRASP_impLS can find better 'Best' values for more instances with respect to the six competitors, which means that the found best solutions of GRASP_impLS are better than the competitors' in most cases. GRASP_impLS can obtain better 'Best' values on 13, 15 and 13 instances with respect to GRASP_impLS with different α values respectively. The 'Best' values of GRASP_impLS are worse than the competitors' on 4, 5, and 4 instances for α parameter respectively. For the parameter *wp*, GRASP_impLS can obtain 'Best' values on 28, 10 and 11 instances, and worse 'Best' values on 1, 6 and 8 instances, with respect to GRASP_impLS with different wp values respectively.

From Table 4, one can see that GRASP_impLS significantly outperforms all six competitors with different α values and *wp* values for the 'Avg' values, which means that GRASP_impLS is more stable than the competitors for finding good solutions with different random seeds. GRASP_impLS can obtain better 'Avg' values on 23, 20 and 20 instances with respect to GRASP_impLS with different α values respectively. The 'Avg' values of GRASP_impLS are worse than the competitors' on 6, 8, and 7 instances for α parameter respectively. For the parameter *wp*, GRASP_impLS can obtain better 'Avg' values on 37, 15 and 14 instances, and worse 'Avg' values on 1, 11 and 12 instances, with respect to GRASP_impLS with different wp values respectively.

In a word, for the algorithm's solution quality, stability and efficiency, it is the best choice that setting the parameter α as 0.8 and setting the parameter *wp* as 0.35 for GRASP_impLS based on the comparisons in Tables 3 and 4.

4.4. Results on random UDG benchmarks

Table 5 shows the comparative results of five solvers on the random UDG benchmarks.

For MWCDS, we use 'Best', 'Avg' and 'Time' to denote the best value of found solutions, the average value of found solutions and the average solving time for obtaining the solutions, respectively. For MA, GRASP and GRASP_impLS, we also use 'Best' and 'Avg' to denote the best value of found solutions and the average value of found solutions, respectively. Moreover, the average execution time of MA, GRASP and GRASP_impLS is denoted as 'AETs' for each instance. The above marks are used in Tables 5–8. For CPLEX, 'Solu' and 'Stat' are used to denote

Table 5
Experiment results of five solvers on the random UDG graph.

Area($N\times N$)	R	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		AETs
Nodes			Solu.	Stat.	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	
	15	18	**6**	5.5	**6**	6.6	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	16	22	**6**	4.42	**6**	6.0	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
80	17	22	**6**	4.11	**6**	6.0	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
15	18	19	**6**	3.54	**6**	6.0	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	19	19	**6**	3.23	**6**	6.0	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	20	22	**4**	0.69	5	5.5	<0.01	**4**	4.0	**4**	4.0	**4**	4.0	300
	20	27	**8**	Feasible	**8**	8.5	<0.01	**8**	8.0	**8**	8.0	**8**	8.0	300
	21	32	**6**	58.86	**6**	6.8	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
100	22	35	**5**	6.46	**5**	6.6	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
20	23	34	**6**	62.77	7	7.4	<0.01	**6**	6.0	7	7.0	**6**	6.0	300
	24	29	**6**	47.91	**6**	7.3	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	25	34	**5**	5.19	6	6.4	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
	20	44	**7**	Feasible	9	9.0	<0.01	**7**	7.0	8	8.0	**7**	7.0	300
	21	47	**7**	Feasible	8	8.8	<0.01	**7**	7.0	8	8.0	**7**	7.0	300
100	22	49	**6**	753.01	8	8.8	<0.01	**6**	6.0	7	7.0	**6**	6.0	300
25	23	53	**6**	1296.45	7	8.5	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	24	59	**6**	2240.62	7	7.8	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	25	45	**7**	Feasible	8	8.7	<0.01	**7**	7.1	**7**	7.0	**7**	7.1	300
	25	46	N/A	N/A	12	12.3	<0.01	**11**	11.0	**11**	11.0	**11**	11.0	300
	26	42	**9**	Feasible	10	12.0	<0.01	**9**	9.0	9	9.0	**9**	9.0	300
150	27	40	N/A	N/A	**11**	12.6	<0.01	**11**	11.0	13	13.0	**11**	11.0	300
30	28	43	N/A	N/A	**10**	11.9	<0.01	**10**	10.0	11	11.0	**10**	10.0	300
	29	49	**9**	Feasible	**9**	10.6	<0.01	**9**	9.0	**9**	9.0	**9**	9.0	300
	30	49	**9**	Feasible	**9**	10.6	<0.01	**9**	9.0	**9**	9.0	**9**	9.0	300
Min			–	–	5	5.5	0.00	4	4.0	4	4.0	4	4.0	300
Max			–	–	12	12.6	0.00	11	11.0	13	13.0	11	11.0	300
Average			–	–	7.54	8.36	0.00	7.00	7.00	7.29	7.29	7.00	7.00	300
Stand.Dev.			–	–	1.91	2.24	0.00	1.89	1.89	2.16	2.16	1.89	1.89	0.00

the solution value and the solving status respectively, which are also used in Tables 5–8.

The minimum value, maximum value, average value and the standard deviation of the computation results for each kind of graphs are also reported in Tables 5–8, which are denoted as 'Min', 'Max', 'Average' and 'Stand. Dev.', respectively. Since the completion statuses of CPLEX for the tested graphs are not stable, we do not report the above values for CPLEX in Tables 5–8.

In Table 5, CPLEX completely solves 14 instances, i.e., the optimal solutions are found and proved by CPLEX for these 14 instances. CPLEX gives undefined solutions for 7 instances, which means that the optimality of these solutions is not proved. For the remaining 3 instances, CPLEX cannot solve them, since the fourth constraint (2)(d) in formula (2) creates too many restrictions even if this constraint has been optimized. For the four incomplete algorithms in Table 5, MWCDS, MA, GRASP and GRASP_impLS obtains minimum best solutions for 14 instances, 24 instances, 17 instances and 24 instances, respectively. Obviously, GRASP_impLS and MA significantly outperform the competitors on these random UDG benchmarks. For the comparison of 'Average' values and 'Stand. Dev.' values, GRASP_impLS and MA have the same performance, and GRASP_impLS outperforms GRASP on the 'Best' columns and 'Avg' columns, which reflects that the improved local search procedure in GRASP_impLS promotes the stability of GRASP.

4.5. Results on LPNMR'09 benchmarks

Table 6 shows the comparative results on the LPNMR'09 benchmarks.

In Table 6, CPLEX only gives undetermined solutions, whose optimality are not proved, for 2 instances. For the remaining 7 instances, CPLEX cannot solve them. For the four incomplete algorithms in Table 6, MWCDS, MA, GRASP and GRASP_impLS obtain minimum best solutions for 0 instances, 9 instances, 9 instances and 9 instances, respectively. Obviously, MA, GRASP and GRASP_impLS significantly outperform MWCDS algorithm on these LPNMR'09 benchmarks. For the comparison of 'Average' values and 'Stand. Dev.' values, GRASP_impLS has the same values on the 'Best' columns and 'Avg' columns with respect to GRASP and MA.

4.6. Results on common UDG benchmarks

Table 7 shows the comparative results on the common UDG benchmark. For these instances, GRASP_impLS can obtain smaller WCDSs with respect to MWCDS, GRASP and MA. CPLEX cannot solve any of the instances in Table 7. Both of the best solution value and average solution value obtained by GRASP_impLS are better than GRASP's for most of these instances, and GRASP_impLS outperforms MA on 13 instances in terms of the obtained best solution value. For the comparison of 'Average' values and 'Stand. Dev.' values, GRASP_impLS comprehensively outperforms GRASP and MA on both of the two columns.

The instances in Table 7 are classified according to different areas and radiuses. The graph with bigger radiuses is denser than the graph with smaller radiuses in the case of the same area and node number, since the disks are more likely to intersect with others in the former graph. So, the size of WCDS is evidently decreasing with radius increased for each area.

4.7. Results on NDR benchmarks

Table 8 shows the comparative results of five solvers on the large graph benchmark.

The comparative results in Table 8 further prove the conclusions obtained in Tables 5–7. GRASP_impLS can obtain the smaller best solution values and average solution values with respect to MWCDS, MA and GRASP for most of instances. GRASP finds better best-solutions with respect to GRASP_impLS on three instances (delaunay_n13, t3dl_a and vibrobox) in Table 8, since our algorithms execute based on some random strategy. However, the average solution values of GRASP_impLS are better than GRASP's for the above three instances, which means that GRASP_impLS is more stable than GRASP. For the comparison of

Table 6
Experiment results of five solvers on the LPNMR'09 graph.

Ins.	$\|V\|$	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		AETs
			Solu.	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	
40 × 200	40	200	**5**	Feasible	8	9.4	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
45 × 250	45	250	**5**	Feasible	8	9.2	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
50 × 250(1)	50	236	N/A	N/A	11	11.9	<0.01	**7**	7.0	**7**	7.0	**7**	7.0	300
50 × 250(2)	50	240	N/A	N/A	9	11.3	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
55 × 250	55	250	N/A	N/A	10	12.3	<0.01	**7**	7.0	**7**	7.0	**7**	7.0	300
60 × 400	60	400	N/A	N/A	10	11.0	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
70 × 250	70	250	N/A	N/A	17	19.4	<0.01	**11**	11.0	**11**	11.0	**11**	11.0	300
80 × 500	80	500	N/A	N/A	14	15.9	<0.01	**8**	8.0	**8**	8.0	**8**	8.0	300
90 × 600	90	600	N/A	N/A	16	17.8	<0.01	**9**	9.0	**9**	9.0	**9**	9.0	300
Min			–	–	8	9.2	0.00	5	5.0	5	5.0	5	5.0	300
Max			–	–	17	19.4	0.00	11	11.0	11	11.0	11	11.0	300
Average			–	–	11.44	13.13	0.00	7.11	7.11	7.11	7.11	7.11	7.11	300
Stand.Dev.			–	–	3.40	3.68	0.00	1.96	1.96	1.96	1.96	1.96	1.96	0.00

Table 7
Experiment results of five solvers on the common UDG graph.

Area($N\times N$)	R	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		AETs
Nodes			Solu.	Stat.	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time	
400 80	60	262	N/A	N/A	18	20.4	<0.01	**13**	13.0	**13**	13.0	**13**	13.0	300
	70	329	N/A	N/A	16	17.9	<0.01	**11**	11.0	**11**	11.0	**11**	11.0	300
	80	400	N/A	N/A	13	15.7	<0.01	**8**	8.0	**8**	8.0	**8**	8.0	300
	90	474	N/A	N/A	11	12.5	<0.01	**8**	8.0	**8**	8.0	**8**	8.0	300
	100	563	N/A	N/A	9	11.1	<0.01	**7**	7.0	**7**	7.0	**7**	7.0	300
	110	647	N/A	N/A	8	9.6	<0.01	**6**	6.0	**6**	6.0	**6**	6.0	300
	120	735	N/A	N/A	8	9.2	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
600 100	80	335	N/A	N/A	21	23.5	<0.01	**15**	15.0	**15**	15.6	**15**	15.0	300
	90	368	N/A	N/A	21	22.5	<0.01	**14**	14.0	15	15.0	**14**	14.0	300
	100	435	N/A	N/A	18	20.0	<0.01	**12**	12.0	**12**	12.0	**12**	12.0	300
	110	500	N/A	N/A	17	18.5	<0.01	**11**	11.0	**11**	11.0	**11**	11.0	300
	120	575	N/A	N/A	14	16.3	<0.01	**10**	10.0	**10**	10.0	**10**	10.0	300
700 200	70	756	N/A	N/A	43	47.3	<0.01	29	29.0	30	30.6	**28**	28.4	300
	80	921	N/A	N/A	37	39.0	<0.01	**24**	24.8	26	26.0	**24**	24.0	300
	90	1113	N/A	N/A	29	32.7	<0.01	**20**	20.0	21	21.0	**20**	20.0	300
	100	1305	N/A	N/A	27	28.6	<0.01	**18**	18.0	19	19.0	**18**	18.0	300
	110	1501	N/A	N/A	23	25.3	<0.01	**16**	16.0	**16**	16.3	**16**	16.0	300
	120	1730	N/A	N/A	20	22.5	<0.01	**13**	13.8	14	14.0	**13**	13.3	300
1000 200	100	756	N/A	N/A	43	47.3	<0.01	29	29.0	30	30.6	**28**	28.4	300
	110	871	N/A	N/A	40	42.5	<0.01	26	26.0	27	27.0	**25**	25.0	300
	120	997	N/A	N/A	32	36.2	<0.01	**23**	23.0	24	24.0	**23**	23.0	300
	130	1127	N/A	N/A	29	32.9	<0.01	**20**	20.0	21	21.0	**20**	20.0	300
	140	1269	N/A	N/A	28	29.7	<0.01	**18**	18.0	19	19.0	**18**	18.0	300
	150	1400	N/A	N/A	26	27.6	<0.01	**16**	16.2	17	17.5	**16**	16.0	300
	160	1541	N/A	N/A	23	24.8	<0.01	**15**	15.0	16	16.0	**15**	15.0	300
1500 250	130	903	N/A	N/A	57	60.7	<0.01	**37**	37.8	39	39.3	**37**	37.0	300
	140	1011	N/A	N/A	52	54.1	<0.01	**34**	34.8	36	36.7	**34**	34.0	300
	150	1119	N/A	N/A	44	49.2	<0.01	**31**	31.5	33	33.2	**31**	31.0	300
	160	1246	N/A	N/A	44	45.4	<0.01	29	29.0	30	30.5	**28**	28.0	300
2000 300	200	1577	N/A	N/A	50	53.5	<0.01	**32**	32.8	34	34.7	**32**	32.0	300
	210	1710	N/A	N/A	48	50.4	<0.01	**30**	30.8	32	32.3	**30**	30.0	300
	220	1849	N/A	N/A	46	47.0	<0.01	28	28.5	29	29.9	**27**	27.3	300
	230	1990	N/A	N/A	41	43.7	<0.01	**26**	26.5	28	28.0	**26**	26.0	300
2500 350	200	1461	N/A	N/A	72	76.7	<0.01	49	49.8	51	51.8	**46**	46.3	300
	210	1555	N/A	N/A	69	73.2	<0.01	44	45.5	48	48.3	**43**	43.7	300
	220	1668	N/A	N/A	65	69.1	<0.01	42	42.8	46	46.0	**40**	40.0	300
	230	1787	N/A	N/A	60	64.4	<0.01	40	40.8	43	43.3	**39**	39.0	300
3000 400	210	1522	N/A	N/A	90	94.4	<0.01	61	62.0	62	63.5	**57**	57.6	300
	220	1621	N/A	N/A	84	88.6	<0.01	59	59.2	61	61.2	**55**	55.1	300
	230	1750	N/A	N/A	79	85.0	<0.01	54	55.2	56	57.0	**51**	51.7	300
	240	1880	N/A	N/A	77	80.1	<0.01	51	51.2	53	53.5	**47**	48.6	300
Min			–	–	8	9.2	0.00	5	5.0	5	5.0	5	5.0	300
Max			–	–	90	94.4	0.00	61	62.0	62	63.5	57	57.6	300
Average			–	–	37.85	40.71	0.00	25.22	25.54	26.39	26.65	24.56	24.69	300
Stand.Dev.			–	–	22.67	23.57	0.00	15.15	15.44	15.94	16.17	14.14	14.32	0.00

Table 8
Experiment results of five solvers on the NDR graph.

Ins.	$\|V\|$	$\|E\|$	CPLEX		MWCDS			MA		GRASP		GRASP_impLS		AETs
			Solu.	Stat	Best	Avg	Time	Best	Avg	Best	Avg	Best	Avg	
ba_1k_30k	1000	30 039	N/A	N/A	114	121.5	<0.01	34	35.2	**21**	21.7	**21**	21.0	300
ba_1k_6k	1000	5964	N/A	N/A	273	284.0	<0.01	172	176.0	89	133.3	**82**	83.0	300
c-fat200-1	200	1534	N/A	N/A	**18**	18.0	<0.01	18	18.0	**18**	18.0	**18**	18.0	300
DD_g140	560	2710	N/A	N/A	164	167.6	<0.01	137	137.5	**131**	137.0	**131**	131.6	300
DD_g142	288	1290	N/A	N/A	88	90.6	<0.01	73	73.5	72	73.3	**70**	71.0	300
DD_g143	414	2088	N/A	N/A	112	116.4	<0.01	93	94.2	96	98.2	**91**	91.3	300
DD_g144	288	1514	N/A	N/A	77	79.3	<0.01	60	60.2	60	60.0	**58**	58.0	300
DD_g145	404	2320	N/A	N/A	96	99.7	<0.01	76	76.8	79	79.3	**74**	74.6	300
DD_g146	327	1506	N/A	N/A	95	99.2	<0.01	77	79.0	78	79.4	**76**	76.5	300
delaunay_n11	2048	6127	N/A	N/A	515	520.7	0.01	458	459.0	394	395.6	**387**	388.0	300
delaunay_n12	4096	12 264	N/A	N/A	1034	1043.5	0.03	965	969.0	801	803.7	**786**	791.3	300
delaunay_n13	8192	24 547	N/A	N/A	2062	2090.3	0.13	2003	2004.0	**1596**	1751.6	1635	1721.5	300
DSJC500-5	500	125 248	N/A	N/A	8	8.8	<0.01	**5**	5.0	**5**	5.0	**5**	5.0	300
er_graph_1k_6k	1000	6000	N/A	N/A	198	213.1	<0.01	153	158.0	127	127.3	**118**	118.6	300
socfb-Haverford76	1446	59 589	N/A	N/A	190	206.2	0.01	115	116.2	101	103.3	**85**	89.0	300
str_0	363	2454	N/A	N/A	104	127.7	<0.01	59	60.0	58	58.7	**57**	57.0	300
str_200	363	3068	N/A	N/A	102	110.8	<0.01	52	53.2	50	50.7	**48**	48.3	300
str_400	363	3157	N/A	N/A	106	113.4	<0.01	53	53.8	51	51.0	**48**	48.0	300
str_600	363	3279	N/A	N/A	98	108.2	<0.01	44	46.5	44	45.1	**42**	42.3	300
SW-1000-6-0d3-trial3	1000	3000	N/A	N/A	267	278.7	<0.01	227	227.5	197	197.9	**183**	183.4	300
SW-100-6-0d3-trial3	100	300	N/A	N/A	26	27.3	<0.01	**16**	16.0	**16**	16.0	**16**	16.0	300
t3dl_a	20 360	265 113	N/A	N/A	2219	2262.0	1.12	2089	2108.5	**1368**	1884.0	1415	1636.0	300
tube1	21 498	459 277	N/A	N/A	1007	1037.8	1	939	944.2	743	755.5	**689**	690.2	300
vibrobox	12 328	177 578	N/A	N/A	1362	1382.8	0.41	1296	1310.6	**964**	1043.0	971	1004.0	300
Min			–	–	8	8.8	0.00	5	5.0	5	5.0	5	5.0	300
Max			–	–	2219	2262.0	1.12	2089	2108.5	1596	1884.0	1635	1721.5	300
Average			–	–	430.63	441.98	0.11	383.92	386.75	298.29	332.86	296.08	310.98	300
Stand.Dev.			–	–	635.11	643.97	0.30	615.71	619.06	451.73	534.26	460.32	497.77	0.00

'Average' values and 'Stand. Dev.' values, GRASP_impLS is only slightly weaker than GRASP on the 'Best' columns for the 'Stand. Dev.' value, but is better than GRASP on the other remaining aspects.

5. Discussion

Overall, for the best solution values, GRASP_impLS can obtain best values for 95 instances of all the 98 instances in Tables 5–8, which is the best solution for solving MWCDSP. The memetic algorithm MA outperforms GRASP for the instances of random UDG graph and common UDG graph in Tables 5 and 7, but is weaker than GRASP for the comparison of obtained best solution values in Table 8. The main reason is that MA can search more solution subspaces with respect to GRASP, since many operations, such as initializing populations, crossover and mutation etc., are used to find more possible solution subspaces. So, MA can quickly find high-quality solutions for some small graphs, after refining the solutions obtained by MWCDS with a simple local search algorithm. However, for the large graphs, a great many solution subspaces are required for exploring and the simple local search algorithm used in MA cannot satisfy the searching requirements in these graphs, which limits the performance of MA. Comparatively, GRASP is more stable than MA for solving MWCDSP.

Comparing the three metaheuristic approaches (MA, GRASP and GRASP_impLS) with the MIP solver CPLEX and the self-stabilizing algorithm MWCDS, the average execution time of the three metaheuristic approaches is longer than MWCDS for all instances, but is shorter than CPLEX for most instances. Though more execution time is spent, these metaheuristic approaches can significantly obtain higher quality solution with respect to the self-stabilizing algorithm. And the metaheuristic approaches can give high quality solutions for all instances which cannot be solved by CPLEX. So, it is worth setting the average execution time of three metaheuristic approaches as 300 s.

The competitive results of average best values for GRASP_impLS vs. competitors are shown in Fig. 5. The values in y-axis is computed by AVG(competitor)/AVG(GRASP_impLS), in which AVG(S) represents the average result of best values obtained by the solver S for some graph. From Fig. 5, one can observe that, GRASP_impLS can obtain

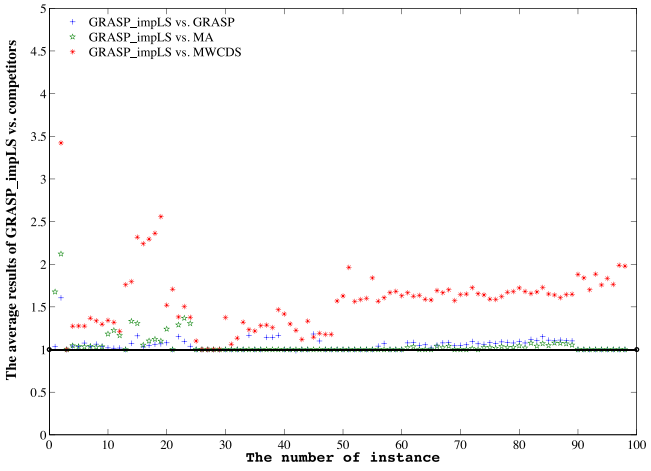

Fig. 5. The comparison of 'AVG' values of MWCDS, GRASP and GRASP_impLS.

better average results of the best solutions with respect to GRASP, MA and MWCDS for most instances. These results further verify that GRASP_impLS can usually obtain better solutions with respect to the above competitors.

We also give the statistical results for comparing the best solutions and average solutions of GRASP_impLS and GRASP with three alternative solvers on four kinds of graphs in Tables 5–8, which are listed in Table 9.

In Table 9, '#win_B.' (resp. '#win_A.') represents the number of instances on which the corresponding solver obtain the minimum 'Best' (resp. 'Avg') solutions. The maximum '#win_B.' and '#win_A.' in each line are marked with bold and framed respectively. The solutions obtained by the CPLEX algorithm are stable, which are not affected by different settings of random seeds, so only '#win_B.' is considered for CPLEX. From Table 9, one can observe that GRASP_impLS significantly outperforms the other four solvers in all kinds of graphs for both aspects

Table 9
Statistical comparative results of the MWCDSP solvers on all 98 instances.

Ins.	#ins.	CPLEX	MWCDS		MA		GRASP		GRASP_impLS	
		#win_B.	#win_B.	#win_A.	#win_B.	#win_A.	#win_B.	#win_A.	#win_B.	#win_A.
Random UDG	24	21	13	4	**24**	**23**	17	17	**24**	**23**
LPNMR'09	9	2	0	0	**9**	**9**	**9**	**9**	**9**	**9**
Common UDG	41	0	0	0	28	19	12	10	**41**	**41**
NDR	24	0	1	1	2	3	8	3	**21**	**24**

of '#win_B.' and '#win_A.'. Especially, GRASP_impLS almost always obtains better 'Best' values and 'Avg' values with respect to the other four solvers for the NDR instances and the Common UDG instances whose scale are greater than the other instances'. Furthermore, the statistical results in Table 9 indicate that our GRASPs outperform the competitors and the designed improved local search procedure is efficient.

From the comparative results of Fig. 5 and Table 9, one can see that GRASP_impLS always outperforms GRASP. The main reason is that the improved local search procedure of GRASP_impLS in Algorithm 4 breaks the limit of the local search procedure in Algorithm 3, i.e., only two vertices are selected to exchange them in one iteration. The method of selecting vertices used in the improved local search procedure of GRASP_impLS allows multiple vertices to be exchanged, which gives more combinations of selecting vertices and improves the mobility of most vertices, just like the detailed analysis in Section 3.4.

6. Conclusions and future work

In this paper, a 0–1 integer linear programming model is designed for the minimum weakly connected dominating set problem (MWCDSP), and we also introduce the framework of GRASP, which comprises of a *ConstructionGreedyRandomized* procedure and a local search procedure, for MWCDSP. The *ConstructionGreedyRandomized* procedure is proposed to construct a weakly connected dominating set (WCDS) after making a balance between the random strategy and greedy strategy. The above WCDS is provided to a local search procedure as its initial candidate solution. Two local search procedures are introduced in this paper, in which two introduced greedy functions *Dscore* and *Nscore* are used to select some vertices. The first local search procedure investigates the neighborhood of current candidate solution based on tabu strategy and the solution weakly connect elements, and searches (k-1)-size WCDS after finding a k-size WCDS. The second local search procedure improves the first one based on new strategies for adding vertices into candidate solution and removing vertices from candidate solution. Experimental results on four types of test problems suggest that our GRASPs can solve MWCDSP efficiently, and the GRASP based on the improved local search procedure outperforms the competitors.

In the future, we would like to further improve the GRASP algorithm for MWCDSP by some other ideas, and try to solve other massive instances.

CRediT authorship contribution statement

Dangdang Niu: Conceptualization, Methodology, Writing – original draft. **Xiaolin Nie:** Software. **Lilin Zhang:** Visualization, Investigation. **Hongming Zhang:** Validation, Writing – review & editing. **Minghao Yin:** Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Funding

This work is supported by the National Key Research and Development Program of China (under Grant No. 2020YFD1100601), the National Natural Science Foundation of China (under Grant Nos. 62206222, 61976050, 61972384, 41771315), the Research Fund of Guangxi Key Lab of Multi-source Information Mining & Security (under Grant No. MIMS19-05), the Natural Science Basic Research Program of Shaanxi Province (under Grant No. 2020JQ-280), the Fundamental Research Funds for the Central Universities (under Grant No. 2412019ZD013), the Natural Science Foundation of Jilin Province (under Grant No. 20200201280JC).

References

Abu-Khzam, F. N. (2022). An improved exact algorithm for minimum dominating set in chordal graphs. *Information Processing Letters*, *174*, Article 106206.
Cai, S., Li, Y., Hou, W., & Wang, H. (2019). Towards faster local search for minimum weight vertex cover on massive graphs. *Information Sciences*, *471*, 64–79.
Cravo, G. L., & Amaral, A. R. (2019). A GRASP algorithm for solving large-scale single row facility layout problems. *Computers & Operations Research*, *106*, 49–61.
Ding, Y., Wang, J. Z., & Srimani, P. K. (2016). A linear time self-stabilizing algorithm for minimal weakly connected dominating sets. *International Journal of Parallel Programming*, *44*(1), 151–162.
Domke, G. S., Hattingh, J. H., & Markus, L. R. (2005). On weakly connected domination in graphs II. *Discrete Mathematics*, *305*(1–3), 112–122.
Du, H., Wu, W., Shan, S., Kim, D., & Lee, W. (2012). Constructing weakly connected dominating set for secure clustering in distributed sensor network. *Journal of Combinatorial Optimization*, *23*.
Dunbar, J. E., Grossman, J. W., Hattingh, J. H., Hedetniemi, S. T., & McRae, A. A. (1997). On weakly connected domination in graphs. *Discrete Mathematics*, *167*, 261–269.
Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71.
Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.
Ferdi, I., & Layeb, A. (2018). A GRASP algorithm based new heuristic for the capacitated location routing problem. *Journal of Experimental and Theoretical Artificial Intelligence*, *30*(3), 369–387.
Han, B., & Jia, W. (2007). Clustering wireless ad hoc networks with weakly connected dominating set. *Journal of Parallel and Distributed Computing*, *67*(6), 727–737.
Hu, S., Liu, H., Wang, Y., Li, R., Yin, M., & Yang, N. (2021). Towards efficient local search for the minimum total dominating set problem. *Applied Intelligence*, *51*(12), 8753–8767.
Jovanovic, R., & Tuba, M. (2013). Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. *Computer Science and Information Systems*, *10*(1), 133–149.
Kamei, S., & Kakugawa, H. (2007). A self-stabilizing approximation algorithm for the minimum weakly connected dominating set with safe convergence. In *Proceedings of the 1st international workshop on reliability, availability, and security* (pp. 57–66).
Li, R., Hu, S., Gao, J., Zhou, Y., Wang, Y., & Yin, M. (2017). GRASP for connected dominating set problems. *Neural Computing and Applications*, *28*(1), 1059–1067.
Nakkala, M. R., Singh, A., & Rossi, A. (2022). Swarm intelligence, exact and matheuristic approaches for minimum weight directed dominating set problem. *Engineering Applications of Artificial Intelligence*, *109*, Article 104647.
Niu, D., Liu, B., & Yin, M. (2021). Local search for weighted sum coloring problem. *Applied Soft Computing*, *106*, Article 107290.
Niu, D., & Yin, M. (2022). A self-stabilizing memetic algorithm for minimum weakly connected dominating set problems. In *The 2nd international workshop on heuristic search in industry (HSI), in conjunction with the 31st international joint conference on artificial intelligence and the 25th European conference on artificial intelligence (IJCAI-ECAI 2022)*.
Pan, S., Ma, Y., Wang, Y., Zhou, Z., Ji, J., Yin, M., & Hu, S. (2022). An improved master-apprentice evolutionary algorithm for minimum independent dominating set problem. *Frontiers of Computer Science*, http://dx.doi.org/10.1007/s11704-022-2023-7.

Pathan, A. S. K., & Hong, C. S. (2006). A key-predistribution-based weakly connected dominating set for secure clustering in DSN. In *LNCS*: *Vol. 4208*, *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*.

Pitsoulis, L. S., & Resende, M. G. (2002). Greedy randomized adaptive search procedures. *Handbook of Applied Optimization*, 168–183.

Raczek, J., & Cyman, J. (2019). Weakly connected roman domination in graphs. *Discrete Applied Mathematics*, *267*, 151–159.

Resende, M. G., & Ribeiro, C. C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications.

Rossi, R. A., & Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the national conference on artificial intelligence, Vol. 6*.

Sandueta, E. P. (2020). Weakly connected total domination critical graphs. *Advances and Applications in Discrete Mathematics*, *25*.

Shin, I., Shen, Y., & Thai, M. T. (2010). On approximation of dominating tree in wireless sensor networks. *Optimization Letters*, *4*(3), 393–403.

Srimani, P. K., & Xu, Z. (2007). Self-stabilizing algorithms of constructing spanning tree and weakly connected minimal dominating set. In *Proceedings - International Conference on Distributed Computing Systems* (p. 3). IEEE.

Sun, W., Hao, J. K., Lai, X., & Wu, Q. (2018). Adaptive feasible and infeasible tabu search for weighted vertex coloring. *Information Sciences*, *466*, 203–219.

Uribe, N. R., Herrán, A., Colmenar, J. M., & Duarte, A. (2021). An improved GRASP method for the multiple row equal facility layout problem. *Expert Systems with Applications*, *182*, Article 115184.

Wang, Y., Cai, S., Chen, J., & Yin, M. (2018). A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *IJCAI international joint conference on artificial intelligence, Vol. 2018-July* (pp. 1514–1522).

Wang, Y., Cai, S., Chen, J., & Yin, M. (2020). SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, *280*, 103230.

Wang, Y., Cai, S., & Yin, M. (2017). Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. In *IJCAI international joint conference on artificial intelligence, Vol. 58* (pp. 267–295).

Wang, Y., Hao, J.-K., Glover, F., & Lü, Z. (2013). Solving the minimum sum coloring problem via binary quadratic programming. arXiv preprint arXiv:1304.5876.

Wang, Y., Ouyang, D., Yin, M., Zhang, L., & Zhang, Y. (2018). A restart local search algorithm for solving maximum set k-covering problem. *Neural Computing and Applications*, *29*(10), 755–765.

Yu, J., Wang, N., & Wang, G. (2012). Constructing minimum extended weakly-connected dominating sets for clustering in ad hoc networks. *Journal of Parallel and Distributed Computing*, *72*(1), 35–47.