Multi-Start Local Search Algorithm for the Minimum Connected Dominating Set Problems

Ruizhi Li 1,2, *, Shuli Hu 3, Huan Liu 3, Ruiting Li 3, Dantong Ouyang 1, * and Minghao Yin 3, *

1 School of Computer Science and Technology, Jilin University, Changchun 130012, China; lirz111@nenu.edu.cn
2 School of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China
3 School of Computer Science and Information Technology, Northeast Normal University, Changchun 130024, China; husl903@nenu.edu.cn (S.H.); liuh407@nenu.edu.cn (H.L.); lir610@nenu.edu.cn (R.L.)
* Correspondence: ouyd@jlu.edu.cn (D.O.); ymh@nenu.edu.cn (M.Y.)

Received: 17 October 2019; Accepted: 20 November 2019; Published: 3 December 2019

Abstract: The minimum connected dominating set (MCDS) problem is a very significant NP-hard combinatorial optimization problem, and it has been used in many fields such as wireless sensor networks and ad hoc networks. In this paper, we propose a novel multi-start local search algorithm (MSLS) to tackle the minimum connected dominating set problem. Firstly, we present the fitness mechanism to design the vertex score mechanism so that our algorithm can jump out of the local optimum. Secondly, we use the configuration checking (CC) mechanism to avoid the cycling problem. Then, we propose the vertex flipping mechanism to change the vertex state by combing the CC mechanism with the vertex score mechanism. Finally, we propose a multi-start local search framework based on these mechanisms. We compare the algorithm MSLS with other compared algorithms on extensive instances. The results of experiment show that MSLS is superior to other algorithms in solution quality and time efficiency on most instances.

Keywords: minimum connected dominating set; configuration checking; vertex score mechanism; multi-start local search

1. Introduction

Given an undirected connected graph $G = (V_t, E_g)$, the minimum dominating set problem is to seek a minimum vertex subset $S$ such that each vertex in $V_t \setminus S$ is adjacent to at least one vertex in $S$, where vertices $a, b$ are adjacent if there exists an edge $(a, b) \in E_g$. In a real network, there are some important issues that can be modeled through dominating sets. Especially in operations research, the location of key or auxiliary stations in a traffic network can be described by a framework based on a dominant set. Other applications come from areas such as sensor networks [1], medicine of cancer therapy [2,3] viral marketing in web based social networks [4], sociology [5], and biology [2,6].

The connected dominating set (CDS) problem is a classical dominating set problem with constraints. That is, a dominating set $S$ is a connected dominating set if the subgraph $G(S) = (S, E_g(S))$ is connected, where $E_g(S) = \{(a, b) \in E_g | a \in S, b \in S\}$. The minimum connected dominating set (MCDS) problem is to seek a connected dominating set $S$ with minimum cardinality. The MCDS problem plays a significant role in many real-world applications such as network testing [7], wireless sensor networks [8], and wireless ad hoc networks [9,10].

It is proved that the minimum connected dominating set problem is NP-hard [11]. Many researchers have devoted themselves to different algorithms to get the optimal solution or near-optimal solution of the problem. In recent years, some researchers have proposed a number of approximate algorithms
to tackle the minimum connected dominating set problem. The literature [12] is a good survey for this problem in ad hoc networks. Marathe et al. proposed efficient approximations for the problem on unit disk graphs, the size of the solution by proposed algorithm is at most $8|S|^*$ + 1, and the time complexity is $O(n^2)$ steps [13]. In [14], Bharghavan et al. proposed a distributed approximation algorithm. Its approximation ratio is $O(H(\Delta))$, and both its message complexity and time complexity are $O(n^2)$, where $\Delta$ is the maximum degree and $H()$ is the harmonic function. Wu et al. proposed a different distributed approximation algorithm in unit disk graphs [15]. Its approximation ratio is $O(n)$, and its message complexity and time complexity are $O(m)$ and $O(n^3)$, respectively. In [16], the approximation ratio is 8, and message complexity and time complexity of the both algorithms are $O(n)$ and $O(n\Delta)$, respectively. In [17], the approximation ratio is 8, and message complexity and time complexity are $O(n\log n)$ and $O(n)$, respectively. In [18], Wu et al. proposed a distributed approximation with approximation ratio $O(1)$ in unit disk graphs. In [19], the approximation ratio is $O(1)$, and both its message complexity and time complexity are $O(n)$. Simonetti et al. presented new valid inequalities, an integer programming formulation, and a branch-and-cut algorithm [20]. Gendron et al. proposed the branch-and-cut, benders decomposition, and memetic approaches for the problem [21]. However, the solution obtained by the approximate algorithm is usually much worse than the optimal solution. Recently, researchers are committed to heuristic development for the MCDS problem in order to produce a good or near optimal solution in a reasonable time. They are mainly one-step based algorithm [8] or two-step based algorithms [7,22,23], or greedy heuristic based algorithms [24,25]. Misra et al. proposed a multi-step collaborative cover heuristic approach in literature [9]. The MCDS problem was also solved by neural networks method [26], simulated annealing with tabu search [27], and the ACO algorithm [28,29]. Li et al. proposed the algorithm GRASP, which incorporates an efficient local search algorithm based on two key components (the tabu strategy and the greedy function) [30]. Wang et al. presented a variable-depth neighborhood search (VDNS) algorithm for solving the MCDS problem [31]. Salim et al. presented an ant colony optimization algorithm based on a reduced variable neighborhood search [32]. However, fewer heuristic algorithms are designed for solving the problem. Thus, in our paper, we propose a multi-start local search with a vertex score mechanism and configuration checking mechanism for solving the MCDS problem.

First, a vertex fitness based vertex score mechanism is proposed to enable the search to jump out of local optimum to a new area to continue the search. In the vertex score mechanism, each vertex’s score value is dynamically updated according to the vertex fitness mechanism. Second, we use a configuration checking mechanism to reduce the cycling problem in the local search phase. The configuration checking (CC) mechanism was proposed for solving a minimum vertex cover problem in the literature [33]. The CC mechanism considers the environmental information of the vertex to change its state rather than the direct properties of the vertex. The CC mechanism has been applied in local search algorithms for combinatorial optimization problems such as minimum weighted vertex cover [34,35] and set covering [36], as well as constraint satisfaction problems such as satisfiability [37–39], maximum satisfiability [40], and a minimum weighted clique problem [41]. However, as far as we know, the configuration checking mechanism has not been applied to MCDS problem.

In order to improve the robustness and efficiency of the algorithm, the multiple-start mechanism is adopted. By incorporating the mechanisms described above, an efficient algorithm MSLS is presented to tackle the MCDS problem. In order to verify the effectiveness of MSLS, we compare it with five other algorithms, called Greedy [42], ACO [28], ACO+PCS [28], GRASP [30], and VDNS [31]. The results of the experiment describe the fact that the presented algorithm performs better than other algorithms in solution quality and time efficiency, and we obtain new upper bounds on some instances.

The rest of this paper is structured as follows. The related definitions and concepts are introduced in Section 2. The vertex fitness mechanism and the vertex score mechanism are described in Section 3. Section 4 introduces how to apply the CC mechanism to the minimum connected dominating set problem. Section 5 designs a new vertex flipping mechanism to guide the direction of local search. Section 6 shows the presented multi-start local search algorithm, called MSLS. Section 7 shows and
analyzes the results of the experiment. Finally, Section 8 summarizes the whole paper and briefly discusses the future direction of work.

2. Preliminaries

Given an undirected connected graph \( G = (V_t, E_g) \), where \( V_t = \{v_1, v_2, \ldots, v_n\} \) is the vertex set and \( E_g = \{e_1, e_2, \ldots, e_m\} \) is the edge set. The network is modeled as an undirected connected graph, where the vertices represent the nodes and the edges represent the communication links. We use \( N(a) = \{b \in V_t \mid (a, b) \in E_g\} \) to denote the set of neighbors of a vertex \( a \), and use \( d(a) = |N(a)| \) to denote the degree of \( a \). We denote \( N[a] = N(a) \cup \{a\} \). Given a candidate solution \( S \), we use \( x_j \in \{0, 1\} \) to denote the state of a vertex \( a_j \), where \( x_j = 1 \) means \( a_j \in S \), and \( x_j = 0 \) means \( a_j \notin S \). We say a vertex \( a \in V_t \) dominates \( b \in V_t \) if \( a = b \) or \( (a, b) \in E_g \). In addition, we call the vertices belonging to \( S \) as dominating vertices, whereas we call the remaining vertices as dominated vertices. We define several different versions of dominating set problems as follows.

**Definition 1 (dominating set (DS)).** In an undirected graph \( G = (V_t, E_g) \), the dominating set problem is to find a vertex subset \( S \subseteq V_t \) such that every vertex in \( V_t \setminus S \) has at least one neighbor in \( S \).

**Definition 2 (minimum dominating set (MDS)).** In an undirected graph \( G = (V_t, E_g) \), the minimum dominating set problem is to find a dominating set \( S \) with the minimum cardinality.

**Definition 3 (connected dominating set (CDS)).** In an undirected connected graph \( G = (V_t, E_g) \), the connected dominating set problem is to seek a dominating set \( S \), such that the subgraph \( G(S) = (S, E_g(S)) \) is connected, where \( E_g(S) = \{(a, b) \in E_g \mid a \in S, b \in S\} \).

**Definition 4 (minimum connected dominating set (MCDS)).** In an undirected connected graph \( G = (V_t, E_g) \), the minimum connected dominating set problem is to seek a connected dominating set \( S \) with minimum cardinality. The linear programming model can be stated as follows:

\[
\text{Minimize } \sum_{a_i \in V_t} x_i \tag{1}
\]

Subject to:

\[
\sum_{a_j \in N[a_i]} y_{ij} \geq 1, \forall a_i \in V_t, \tag{2}
\]

\(\text{connected}(x)\)

\[
y_{ij} \leq x_i, \forall a_i, a_j \in V_t, \tag{3}
\]

\[
x_i, y_{ij} \in \{0, 1\}, \forall a_i, a_j \in V_t, \tag{4}
\]

where \( x_i \) is an indicator variable that is equal to 1 if vertex \( a_i \) is included in the candidate solution and otherwise \( x_i = 0 \); \( y_{ij} \) is an indicator variable that is equal to 1 if vertex \( a_i \) dominates vertex \( a_j \) and otherwise \( y_{ij} = 0 \). Equation (2) ensures that each vertex \( a_i \in V_t \setminus S \) is dominated by at least one vertex in \( S \) and Equation (3) imposes the connectivity of the subgraph induced by \( S \), by enforcing articulation points [21]. Equations (4) and (5) are the integer constraints.

3. Vertex Fitness Based Vertex Score Mechanism

In the local search procedure for solving the MCDS problem, the vertex score mechanism plays a key role in selecting which vertices to flip, which will allow the search to be conducted in a more efficient direction. In this paper, the vertex score mechanism mainly depends on the fitness value of the vertex. We shall describe the vertex fitness as follows.
3.1. Vertex Fitness Mechanism

The vertex fitness mechanism plays an important role in vertex score mechanism. Specifically, in the vertex fitness mechanism, each vertex is associated with an integer number, which is known as fitness. In the local search process, each undominated vertex’s fitness in $Vt \setminus S$ will be increased by one. The updating of fitness value will affect the scores of the related vertices, so that the algorithm can effectively jump out of the local optimal optimum. It is worth noting that the vertex fitness mechanism in our algorithm MSLS is a very generalized strategy, similar to a vertex weighting strategy, which has been successfully applied to solve MVC [33], MWVC [35], and MKDS [42]. To illustrate the problem, we first give the definition of vertex fitness.

**Definition 5 (vertex fitness).** In an undirected graph $G = (Vt, E_g)$, each vertex $a \in Vt$ is assigned a positive integer number $\text{fitness}(a)$, which is dynamically updated in the local search procedure.

In our implementation, we update the fitness values of the vertices according to the following two rules.

**Rule 1.** During initialization, the fitness value ($\text{fitness}(a)$) of each vertex $a \in Vt$ is assigned to 1.

**Rule 2.** During the local search process, we will check if each vertex $a \in Vt$ is dominated at the end of the iteration, and add the fitness value of the undominated vertex by 1.

3.2. Vertex Score Mechanism

Based on the vertex fitness mechanism introduced in the Section 3.1, we propose a vertex scoring mechanism to measure the benefits of flipping the vertex state. The score of vertex $a$ is denoted by $v_{-score}(a)$ as follows:

$$v_{-score}(a) = \text{solution}_\_\text{score}(S) - \text{solution}_\_\text{score}(S'),$$  \hspace{1cm} (6)

where $S$ denotes the current candidate solution, $S' = S \setminus \{a\}$ if $a \in S$, and $S' = S \cup \{a\}$ otherwise, and $\text{solution}_\_\text{score}(S)$ denotes the total fitness of vertices undominated by $S$. $\text{solution}_\_\text{score}(S)$ is used to measure the quality of candidate solution $S$, and the smaller the value $\text{solution}_\_\text{score}(S)$, the better the quality of the solution $S$. When the value $\text{solution}_\_\text{score}(S)$ is 0, the candidate solution $S$ is feasible. $\text{solution}_\_\text{score}(S)$ is defined as follows:

$$\text{solution}_\_\text{score}(S) = \sum_{a \in Vt: N(a) \cap S = \emptyset} \text{fitness}(a).$$  \hspace{1cm} (7)

4. Configuration Checking Mechanism

The cycling problem is an inherent problem in local search, and the effective mechanism of avoiding cycling problem can improve the performance and efficiency of the search algorithm. The configuration checking mechanism is designed to avoid the cycling problem, which is originally introduced in [33] for solving minimum vertex cover problem. We first introduce the basic idea of configuration checking mechanism. For vertex $a$, its configuration denotes the states of its all neighbor vertices. If $a$’s configuration is not changed since it was deleted from the candidate solution, then it is inhibitory added back into the candidate solution. The configuration checking mechanism is generally implemented with a Boolean vector called $\text{config}$, if $\text{config}(a)$ equals 1, then the vertex $a$ is permitted to be added into the candidate solution; otherwise, vertex $a$ is not allowed to be added into the candidate solution.

The configuration checking mechanism for MVC can be modified for the MCDS problem straightforwardly. In the beginning, the value of $\text{config}(a)$ is initialized as 1 for each vertex in $Vt$. Afterwards, if a vertex $a$ is removed from $S$, then the value of $\text{config}(a)$ is assigned to 0 and for each vertex $b \in N(a)$, $\text{config}(b)$ is assigned to 1. Similarly, if a vertex $a$ is added into $S$, for each vertex $b \in N(a)$, $\text{config}(b)$ is assigned to 1.
5. Vertex Flipping Mechanism

The process of local search for solving the connected dominating set is to flip the state of a vertex in each iteration in order to obtain a new candidate solution and judge whether the candidate solution is better than the current optimal solution. It is important to flip the state of which vertex in each iteration. Using the vertex fitness based vertex score mechanism introduced in Section 3 and the CC mechanism introduced in Section 4, we propose the vertex flipping mechanism. First, we shall describe the definition of solution connected elements.

**Definition 6 (solution connected elements (SCE)).** Solution connected elements are vertices with the most times appearing in the neighbors of each connected component of the candidate solution $S$ [30].

One of the difficulties in solving the connected dominating set problem is how to guarantee the connectivity of the solution. Just like reference [30], we also use SCE to guarantee the connectivity of the solution. According to the definition of SCE, if the number of the connected components is equal to the time that vertex $a$ appearing in each connected component neighbors and vertex $a$ is flipped, then the candidate solution will change from not connected to connected. If they are not equal, then the algorithm will flip the vertex with the most times appearing in each connected component neighbors, so that the candidate solution can more easily connect in the future local search step.

Based on previous consideration, Algorithm 1 describes which vertex’s state is flipped to add it into the candidate solution. In Algorithm 1, from the vertices in $SCE$ with $config$ value of 1, the procedure prefers to flip the vertices with the maximum $v\_score$ value (lines 3–6). If there are multiple vertices, Algorithm 1 will flip the vertex with the old age (lines 7–8). At the end of the procedure, the state of the flipped vertex will change from 0 to 1 (line 11), the $config$s and $v\_scores$ of related vertices and $SCE$ set will be updated accordingly (lines 12–16).

**Algorithm 1** Vertex\_flipping\_add()
1: Initial $max\_score = -\infty, add\_v = -1$;
2: for each vertex $a$ in $SCE$
3: if $config(a) == 0$ continue;
4: if $v\_score(a) > max\_score$
5: $max\_score = v\_score(a)$;
6: $add\_v = a$;
7: else $v\_score(a) == max\_score \&\& age(a) > age(add\_v)$
8: $add\_v = a$;
9: endif
10: endfor
11: $S \leftarrow S \cup \{add\_v\}$;
12: for each vertex $b$ in $N(add\_v)$
13: $config(b) = 1$;
14: endfor
15: update $v\_scores$ of related vertices;
16: update $SCE$;

Similarly, Algorithm 2 describes which vertex’s state is flipped to remove it from the candidate solution. In Algorithm 2, from the candidate solution, we prefer to flip the vertices with the maximum $v\_score$ value (lines 3–5). If there are multiple vertices, Algorithm 2 will flip the vertex with the old age (lines 6–7). At the end of the procedure, the state of the flipped vertex $remove\_v$ will change from 1 to 0 (line 10), and the $config$s and $v\_scores$ of related vertices and $SCE$ set will be updated accordingly (lines 11–16).
Algorithm 2 Vertex_flipping_remove()

1: initial max_score = −∞, remove_v = −1;
2: for each vertex a in the candidate solution
3:     if v_score(a) > max_score
4:         max_score = v_score(a);
5:     remove_v = a;
6:     elseif v_score(a) == max_score && age(a) > age(remove_v)
7:         remove_v = a;
8:     endif
9: endfor
10: S ← S \{remove_v};
11: config(remove_v) = 0;
12: for each vertex b in N(remove_v)
13:     config(b) = 1;
14: endfor
15: update v_scores of related vertices;
16: update SCE;

6. Framework of MSLS

In this section, we will discuss the main framework of our algorithm MSLS and its parts. Before we introduce the framework of MSLS, we first introduce the construction of the initial solution.

6.1. Initial Solution Construction

Algorithm 3 describes how to construct an initial solution in each restart of our MSLS algorithm. At the beginning of the procedure, each vertex’s v_score in Vt is initialized to the vertex’s degree. Afterwards, the candidate solution S is initialized NULL, and SN (solutions neighbors) is used to store the neighbors of the vertices in the candidate solution S, flipping such a vertex does not break the candidate solution’s connectivity. If the candidate solution S is empty, each vertex in Vt can be flipped into S, so SN is initialized to Vt; otherwise, SN contains the neighbors of the vertices in the candidate solution S, that is, SN ← ∪a∈S N(a) ∪ Vt \ S. In the main loop of the initial solution construction procedure, the v_score of each vertex in Vt is first calculated according to Formula (6). Then, a vertex is flipped to be added into the candidate solution greedily (breaking ties randomly) from SN to construct the initial solution. The cycle will continue until the candidate solution becomes a feasible solution.

Algorithm 3 Initial_solution()

1: Initialize v_score(a) = d(a), a∈Vt;
2: S ← ∅;
3: SN ← Vt;
4: While S is not CDS do
5:     b ← such a vertex with greatest v_score in SN, breaking ties randomly;
6:     S ← S ∪ {b};
7:     SN ← ∪a∈S N(a) ∪ Vt \ S;
8: endwhile
9: return S;

6.2. Algorithm MSLS

Our algorithm MSLS (Algorithm 4) is based on a multi-restart framework. In the multi-restart framework, the algorithm repeatedly performs the initial solution construction stage (line 3) and the local search stage (lines 4–14). Local search starts with the initial candidate solution to find a higher quality solution. Specifically, the initial candidate solution is constructed by the greedy method
(line 3), on which we use the local search method to seek the local optimal optimum (lines 6–12). Once a connected dominating set \( S \) is obtained, the algorithm flips a vertex in the candidate solution to remove it from \( S \) (line 8) until there are some undominated vertices. Then, the algorithm exchanges two vertices iteratively until \( S \) becomes a feasible solution (lines 11 and 12). Repeat this process until the best solution has not been improved within specific threshold \( \text{MAX\_no\_improve} \) (lines 5–14). Finally, we restart the previous process for a given time in order to search in different promising areas. With the techniques above, MSLS is described as follows in Algorithm 1. Note that \( S \) denotes the local optimum found by local search, \( S^* \) records the best solution obtained so far, and \( N_{\text{start}} \) indicates the maximum restarting times. In our experiments, \( N_{\text{start}} \) is set to be 100, and \( \text{MAX\_no\_improve} \) is set to be 10,000.

**Algorithm 4 Pseudo-code of MSLS**

**Input:** An undirected connected graph \( G = (V_t, E_g) \), \( N_{\text{start}} \), \( \text{MAX\_no\_improve} \)

**Output:** The best solution \( S^* \)

1. \( \text{iter} \leftarrow 0; S^* \leftarrow \emptyset; \)
2. while \( \text{iter} < N_{\text{start}} \) do
3. \( S \leftarrow \text{Initial\_solution}(); \) /*see Section 6.1*/
4. \( \text{no\_improve} \leftarrow 0; \)
5. while \( \text{no\_improve} < \text{MAX\_no\_improve} \) do
6. if \( S \) is better than \( S^* \);
7. \( S^* \leftarrow S; \)
8. \( \text{vertex\_flipping\_remove}(); \) /*see Section 5*/
9. \( \text{no\_improve} \leftarrow 0; \)
10. endif
11. \( \text{vertex\_flipping\_remove}(); \) /*see Section 5*/
12. \( \text{vertex\_flipping\_add}(); \) /*see Section 5*/
13. \( \text{no\_improve} \leftarrow \text{no\_improve} + 1; \)
14. endwhile
15. \( \text{iter} \leftarrow \text{iter} + 1; \)
16. endwhile
17. return \( S^*; \)

### 7. Computational Experiments

In this section, we will test the performance of our algorithm MSLS. We compare our algorithm MSLS with the current best algorithms on a large number of benchmark instances. In addition, we analyzed the effectiveness of each proposed mechanism.

#### 7.1. Benchmark Instances

We use the benchmarks provided in [21] and [28] to test our algorithm. The benchmark instances are divided into three categories. The first group of instances (Type I) was tested on the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’09) [28]. The second benchmark instances (Type II) corresponds to ad hoc network clustering problems [28], which are generated by Jovanovic et.al presented in the paper [43]. The third one (Type III) was introduced in [44] used in [21]. The densities of the third instances are ranging from 5 to 70%, and the vertex number of each instance is ranged from \( \{30, 50, 70, 100, 120, 150, 200\} \).

#### 7.2. Comparison with State-of-the-Art Algorithms

In this section, we compare our proposed algorithm MSLS with the current best algorithms, i.e., Greedy [7], ACO [28], ACO+PCS [28], GRASP [30], and VDNS [31]. The greedy algorithm first uses a greedy heuristic approach to find a dominating set, and then applies a steiner tree algorithm to make the dominating set connected. ACO algorithm is a basic Ant Colony Optimization Algorithm. ACO+PCS is an improved version of ACO, which incorporated the Pheromone Correction Strategy. The GRASP
algorithm integrates two key components (greedy function and tabu strategy) to ensure the high efficiency. VNDS is a variable-depth neighborhood search algorithm with an effective partition-based neighborhood structure, pruning the search branch strategy, and the incremental evaluation technique.

Note that our presented algorithm is executed 10 times on each instance of the three types of instances like algorithms ACO, ACO + PCS, and GRASP.

Our algorithm MSLS is coded in C, and it runs on a computer with Intel(R) Xeon(R) CPU E7-4830 (Dell, Xiamen, China) with 2.13 GHz. LPNMR’09 runs on a computer with Genuine Intel Intel(R) Core(TM)2 CPU E8500 with 3.16 GHz. GRASP runs on a computer with Intel(R) Xeon(R) CPU E7-4830 with 2.13 GHz. VNDS runs on a computer with Intel i3 CPU of 2.1 GHz.

7.3. Experimental Results

To test the effectiveness of MSLS algorithm, we compare MSLS with other current best algorithms in terms of both the best solutions and running time. We also show the lower bound (denoted LB) obtained from the Lagrangian dual problem solved by the subgradient method and the lower bound can be used as a stopping criterion to verify if optimum has been found. Our heuristic algorithms all give the upper bound of the objective value. In Table 1, we give the experimental results of the Type I instances.

| Problem | LB | LPNMR | Greedy | ACO | ACO + PCS | GRASP | VDNS | MSLS |
|---------|----|-------|--------|-----|----------|-------|------|------|
|         |    |       |        | Avg | Avg      | Avg   | Avg  | Avg  |
|         |    |       |        | St. | St.      | St.   | St.  | St.  |
|         |    |       |        | Tim | Tim      | Tim   | Tim  | Tim  |
| 40 × 200 | 5  | 5     | 10     | 5.8 | 0.6      | 3.2   | 5.3  | 0.45 |
| 45 × 250 | 5  | 5     | 15     | 5.8 | 0.4      | 3.5   | 5.5  | 0.5  |
| 50 × 250 | 6  | 8     | 15     | 8.1 | 0.54     | 4.8   | 8    | 0.7  |
| 50 × 250_1 | 6  | 7     | 17     | 7.5 | 0.5      | 5.1   | 6.5  | 7    |
| 50 × 250_2 | 6  | 8     | 20     | 8.8 | 0.98     | 5.6   | 8.3  | 0.45 |
| 55 × 250 | 6  | 8     | 20     | 8.8 | 0.98     | 5.6   | 8.3  | 0.45 |
| 60 × 400 | 5  | 7     | 15     | 7   | 0       | 6.1   | 7    | 9.1  |
| 70 × 250 | 11 | 13    | 32     | 14.2| 0.74     | 11    | 13.9 | 1.04 |
| 80 × 500 | 7  | 9     | 20     | 10  | 0.44     | 12    | 9.8  | 0.4  |
| 90 × 600 | 10 | 19    | 19     | 10.9| 0.83     | 14    | 10.6 | 1.01 |
| Average | 6.4| 8.0   | 18.1   | 8.7 | 8.3      | 7.7   | 8.0  | 7.7  |

In Table 1, the first column Problem indicates the number of vertices and edges. In addition, the third column LPNMR shows the best results known as originally provided in literature [28]. In all benchmark instances, algorithms ACO, ACO + PCS, GRASP, VDNS, and MSLS can obtain the optimal solutions, so we omit the best found solutions. Column Avg indicates the average objective values of each algorithm. For Avg, our algorithm MSLS and GRASP are better than other algorithms on two instances (70 × 250, and 90 × 600). For standard deviation (St.), algorithms MSLS and GRASP both obtain 0 for each instance, which means that MSLS and GRASP obtain best solutions each run. Columns Tim shows average running time in second for 10 runs. Our algorithm MSLS is faster than other algorithm on most instances. The bold values indicate the best solution values obtained among the compared algorithms. The bold values in other tables mean the same thing.

In Table 2, we show the results of the Type II instances. The first column Problem indicates the instance names. For these types of instances, we only compared the best found solutions (Min) and the average solutions (Avg) for the five algorithms ACO, ACO+PCS, GRASP, VDNS, and MSLS. We observed that the greedy algorithm performs poorly on most problems compared to other algorithms. Compared with the greedy algorithm, ACO and ACO+PCS methods can improve the best solution by about 2–3 times, GRASP, VDNS and MSLS methods improve the best solution by about 3–4 times. Specifically, MSLS improves the best found solutions on 14 instances, obtains the best found solutions for other 26 instances, and only gets a worse solution (2000 × 300 × 200) than algorithm VDNS. For the average solution, MSLS improves the result quality in 40 instances except one (400 × 80 × 80). For the
instance, GRASP and MSLS obtains the same average solution. Comparing the MSLS algorithm and GRASP algorithm, MSLS is faster than GRASP by 1–3 times. Comparing the MSLS algorithm and VDNS algorithm, VDNS obtained the best solutions within 30 s, and MSLS could obtain the same or better solutions within less than 30 s.

Table 2. Results of each algorithm on Type II instances.

| Problem      | LB | Greedy | ACO  | ACO + PCS | GRASP | VDNS | MSLS |
|--------------|----|--------|------|-----------|-------|------|------|
|              |    |        | Min  | Avg       |       |      |      |
|              |    |        | Mean |           | Mean  |      |      |
|              |    |        | Mean |           | Mean  |      |      |
|              |    |        | Mean |           | Mean  |      |      |
|              |    |        | Mean |           | Mean  |      |      |
|              |    |        | Mean |           | Mean  |      |      |
| 3000_400_230 | 4  | 25     | 8    | 8         | 8.5   | 8.5  | 8.5  |
| 3000_400_220 | 4  | 25     | 8    | 8         | 8.5   | 8.5  | 8.5  |
| 5000_400_210 | 4  | 25     | 8    | 8         | 8.5   | 8.5  | 8.5  |

In Table 3, we show the results of the Type III instances. The first column Problem indicates the instance names. For these types of instances, we compared the best found solutions (Min) and the average solutions (Avg) of the three algorithms GRASP, VDNS, and MSLS. The optimal solutions are given in the literature [21]. Column OPT indicates the optimal solution of each instance. For 41 instances, GRASP can obtain 33 optimal solutions, VDNS can obtain 40 optimal solutions, and MSLS can obtain all optimal solutions. From Tables 1–3, we can see that there’s a big gap between the solution obtained by MSLS and the lower bound (LB) for most instances.
These experiments are tested on the Type II Instances.

Table 3. Results of each algorithm on Type III instances.

| Problem | OPT | LB | GRASP | VDNS | MSLS |
|---------|-----|----|-------|------|------|
|         |     |    | Min  | Avg  | Tim | Min | Avg | Tim |
| 30_d10  | 15  | 12 | 15   | 15   | 3.5 | 15  | 7   | 15  | 15  | 3.1 |
| 30_d20  | 7   | 6  | 7    | 7    | 2.9 | 7   | <1  | 7   | 7   | 2.4 |
| 30_d30  | 4   | 4  | 4    | 4    | 3.5 | 4   | <1  | 4   | 4   | 2.9 |
| 30_d50  | 3   | 2  | 3    | 3    | 3.1 | 3   | <1  | 3   | 3   | 4.3 |
| 30_d70  | 2   | 2  | 2    | 2    | 3.7 | 2   | <1  | 2   | 2   | 5.2 |
| 50_d5   | 31  | 21 | 33   | 33.7 | 2.5 | 31  | <1  | 31  | 31  | 5.2 |
| 50_d10  | 12  | 11 | 12   | 12   | 2.3 | 12  | <1  | 12  | 12  | 3.0 |
| 50_d20  | 7   | 6  | 7    | 7    | 3.1 | 7   | <1  | 7   | 7   | 4.5 |
| 50_d30  | 5   | 4  | 5    | 5    | 2.4 | 5   | <1  | 5   | 5   | 5.7 |
| 50_d50  | 3   | 3  | 3    | 3    | 3.3 | 3   | <1  | 3   | 3   | 10.4 |
| 50_d70  | 2   | 2  | 2    | 2    | 2.8 | 2   | <1  | 2   | 2   | 13.6 |
| 70_d5   | 27  | 22 | 29   | 30.6 | 5.1 | 27  | <1  | 27  | 27  | 4.8 |
| 70_d10  | 13  | 11 | 13   | 13   | 3.4 | 13  | <1  | 13  | 13  | 4.6 |
| 70_d20  | 7   | 6  | 7    | 7    | 3.0 | 7   | <1  | 7   | 7   | 6.1 |
| 70_d30  | 5   | 4  | 5    | 5    | 2.9 | 5   | <1  | 5   | 5   | 9.9 |
| 70_d50  | 3   | 2  | 3    | 3    | 1.9 | 3   | <1  | 3   | 3   | 17.6 |
| 70_d70  | 2   | 2  | 2    | 2    | 3.2 | 2   | <1  | 2   | 2   | 21.4 |
| 100_d5  | 24  | 21 | 26   | 26.7 | 4.2 | 24  | 18  | 24  | 24  | 5.2 |
| 100_d10 | 13  | 11 | 13   | 13   | 2.8 | 13  | 1   | 13  | 13  | 6.6 |
| 100_d20 | 8   | 5  | 8    | 8    | 3.2 | 8   | 1   | 8   | 8   | 10.3 |
| 100_d30 | 6   | 4  | 6    | 6    | 1.7 | 6   | 1   | 6   | 6   | 14.8 |
| 100_d50 | 4   | 2  | 4    | 4    | 1.7 | 4   | 1   | 4   | 4   | 27.1 |
| 100_d70 | 3   | 3  | 3    | 3    | 1.8 | 3   | 1   | 3   | 3   | 41.4 |
| 120_d5  | 25  | 23 | 27   | 27.8 | 2.1 | 25  | 3   | 25  | 25  | 6.4 |
| 120_d10 | 13  | 11 | 14   | 14.3 | 1.9 | 13  | 1   | 13  | 13  | 7.9 |
| 120_d20 | 8   | 5  | 8    | 8.2  | 1.7 | 8   | 2   | 8   | 8   | 13.1 |
| 120_d30 | 6   | 4  | 6    | 6    | 1.7 | 6   | 3   | 6   | 6   | 19.5 |
| 120_d50 | 4   | 2  | 4    | 4    | 2.1 | 4   | 1   | 4   | 4   | 35.2 |
| 120_d70 | 3   | 3  | 3    | 3    | 2.0 | 3   | 1   | 3   | 3   | 56.2 |
| 150_d5  | 26  | 22 | 27   | 28.2 | 2.0 | 26  | 15  | 26  | 26  | 8.4 |
| 150_d10 | 14  | 11 | 15   | 15.3 | 2.2 | 14  | 17  | 14  | 14  | 11.1 |
| 150_d20 | 9   | 5  | 9    | 9    | 1.9 | 9   | 195 | 9   | 9   | 17.8 |
| 150_d30 | 4   | 4  | 6    | 6    | 1.6 | 6   | 8   | 6   | 6   | 26.5 |
| 150_d50 | 4   | 2  | 4    | 4    | 1.7 | 4   | 3   | 4   | 4   | 50.0 |
| 150_d70 | 3   | 3  | 3    | 3    | 1.6 | 3   | 3   | 3   | 3   | 82.3 |
| 200_d5  | 27  | 22 | 29   | 30.7 | 2.2 | 28  | 18  | 27  | 27  | 11.8 |
| 200_d10 | 16  | 10 | 16   | 16.9 | 2.1 | 16  | 247 | 16  | 16  | 16.6 |
| 200_d20 | 9   | 5  | 9    | 9.7  | 1.8 | 9   | 163 | 9   | 9   | 26.5 |
| 200_d30 | 7   | 4  | 7    | 7    | 1.7 | 7   | 59  | 7   | 7   | 40.4 |
| 200_d50 | 4   | 2  | 4    | 4    | 1.7 | 4   | 9   | 4   | 4   | 79.4 |
| 200_d70 | 3   | 2  | 3    | 3    | 1.7 | 3   | 2   | 3   | 3   | 136.0 |
| Average | 9.6 | 7.5 | 9.9 | 10.1 | 9.6 | 9.6 | 9.6 |

7.4. Comparison of Different Versions of MSLS

From the above experimental results and discussion, it can be seen that our algorithm MSLS is superior to other comparison algorithms in most cases. In this section, we will further study the influence of various important factors in our presented algorithm MSLS: vertex fitness mechanism (Section 3), configuration checking mechanism (Section 4). These experiments are tested on the Type II Instances.

We obtain two variants of MSLS algorithm by rewriting the algorithm.

MSLS-CC: the algorithm does not use the configuration checking mechanism.

MSLS-VF: the only difference between MSLS-VF and MSLS is that MSLS-VF uses a static score function instead of the vertex fitness based dynamic score functions described in Section 3.
We test the two variants together with MSLS on the Type II Instances. As is shown in Table 4, MSLS performs better than MSLS-CC on almost all instances for best solutions (Min) and average solutions (Avg). For best solution, MSLS and MSLS-VF could obtain similar solutions. For average solutions, MSLS outperforms MSLS-VF on most instances. Therefore, we can come to the conclusion that the configuration checking mechanism and the vertex fitness mechanism presented in our algorithm are really efficient and play a significant role in tackling the MCDS problems.

Table 4. Results of MSLS-CC, MSLS-VF and MSLS on Type II graphs.

| Problem       | Min-CC | Avg-CC | Min-VF | Avg-VF | Min  | Avg  |
|---------------|--------|--------|--------|--------|------|------|
| 400_80_70     | 15     | 15.0   | 14     | 14.0   | 14   | 14.0 |
| 400_80_110    | 7      | 7.4    | 7      | 7.0    | 7    | 7.0  |
| 400_80_120    | 6      | 6.0    | 7      | 7.0    | 6    | 6.0  |
| 600_100_100   | 17     | 17.0   | 16     | 16.0   | 16   | 16.0 |
| 600_100_110   | 15     | 15.0   | 14     | 14.9   | 14   | 14.5 |
| 600_100_120   | 14     | 14.0   | 13     | 13.0   | 13   | 13.0 |
| 700_200_100   | 24     | 24.5   | 23     | 23.1   | 22   | 22.5 |
| 700_200_110   | 21     | 21.0   | 20     | 20.3   | 20   | 20.0 |
| 700_200_120   | 18     | 18.0   | 17     | 17.1   | 17   | 17.0 |
| 1000_200_130  | 27     | 27.8   | 26     | 26.4   | 26   | 26.8 |
| 1000_200_140  | 24     | 24.8   | 23     | 23.7   | 23   | 23.1 |
| 1000_200_160  | 20     | 20.5   | 19     | 19.8   | 19   | 19.6 |
| 1500_250_130  | 50     | 51.2   | 50     | 50.1   | 49   | 50.0 |
| 1500_250_140  | 44     | 45.0   | 44     | 44.4   | 44   | 44.3 |
| 1500_250_160  | 37     | 38.4   | 37     | 37.2   | 37   | 37.7 |
| 2000_300_210  | 38     | 39.1   | 38     | 38.6   | 38   | 38.6 |
| 2000_300_220  | 36     | 36.6   | 36     | 36.1   | 36   | 36.3 |
| 2000_300_230  | 36     | 36.0   | 34     | 34.8   | 34   | 34.8 |
| 2500_350_200  | 61     | 62.5   | 61     | 61.5   | 61   | 61.3 |
| 2500_350_210  | 58     | 59.5   | 56     | 57.2   | 56   | 57.0 |
| 2500_350_220  | 54     | 55.8   | 52     | 53.5   | 52   | 54.3 |
| 3000_400_220  | 72     | 73.2   | 71     | 72.0   | 71   | 71.9 |
| 3000_400_230  | 68     | 69.1   | 66     | 67.0   | 66   | 67.0 |
| 3000_400_240  | 62     | 64.3   | 61     | 62.6   | 61   | 62.5 |

8. Conclusions

In this paper, we presented a new local search algorithm, namely MSLS, to solve the minimum connected dominating set problem. The vertex score mechanism can make the algorithm jump out of the local optimum when it is trapped in the local optimum. Then, the configuration checking mechanism was applied to avoid the cycling problem in local search phase. Furthermore, we proposed a vertex flipping mechanism to change the vertex state by combing the CC mechanism with the vertex score mechanism. Based on the above mechanisms, we proposed the multi-start local search algorithm. We compared our algorithm MSLS with other state-of-the-art algorithms on a number of benchmark instances. The results of the experiment show that our proposed algorithm is superior to other algorithms in solution quality and time efficiency. For the Type I instance, MSLS found optimal solutions for all nine instances. For the Type II instance, MSLS improved 14 optimal solutions of 41 instances. For the Type III instance, MSLS obtained optimal solutions for all 41 instances. In the future work, our algorithm can be extended to solve other combinatorial optimization problems, such as the set–union knapsack problem [45], the minimum conductance graph partitioning problem [46], the set k-covering problem [47], and the partial vertex cover problem [48].
Author Contributions: Software, R.L. (Ruizhi Li) and S.H.; Methodology, H.L. and R.L. (Ruiting Li); Writing—original draft preparation, R.L. (Ruizhi Li); Writing—review and editing, D.O. and M.Y.

Funding: This work was supported the National Natural Science Foundation of China (NSFC) under Grant No. 61806082, the National Social Foundation of China under Grant No. 19BYY246, the Certificate of China Postdoctoral Science Foundation Grant No. 2019M651208, the Jilin provincial science and technology department project Nos. 20190103030JH and 20190201186JC, and the Jilin education department 13th five-year science and technology project Nos. JJKH20190726KJ and JJKH20190725KJ.

Conflicts of Interest: The authors declare no conflict of interest.

References
1. Nehéz, M.; Bernát, D.; Klaučo, M. Comparison of algorithms for near-optimal dominating sets computation in real-world networks. In Proceedings of the 16th International Conference on Computer Systems and Technologies, Dublin, Ireland, 25–26 June 2015; pp. 199–206.
2. Wang, H.; Zheng, H.; Browne, F.; Wang, C. Minimum dominating sets in cell cycle specific protein interaction networks. In Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Belfast, UK, 2–5 November 2014; pp. 25–30.
3. Wong, B. Cancer Informatics in the Post Genomic Era; Igor, J., Dennis, W., Eds.; Springer: Boston, MA, USA, 2007; Volume 137.
4. Nehéz, M.; Mäsiar, J. On Propagation in Social Networks—A Graph Mining Approach. In Proceedings of the 6th International Workshop on Knowledge Management, Trenčín, Slovakia, 21–22 October 2011; pp. 107–118.
5. Kelleher, L.L.; Cozzens, M.B. Dominating sets in social network graphs. Math. Soc. Sci. 1988, 16, 267–279. [CrossRef]
6. Wuchty, S. Controllability in protein interaction networks. Proc. Natl. Acad. Sci. USA 2014, 111, 7156–7160. [CrossRef] [PubMed]
7. Guha, S.; Khuller, S. Approximation algorithms for connected dominating sets. Algorithmica 1998, 20, 374–387. [CrossRef]
8. Ruan, L.; Du, H.; Jia, X.; Wu, W.; Li, Y.; Ko, K.I. A greedy approximation for minimum connected dominating sets. Theor. Comput. Sci. 2004, 329, 325–330. [CrossRef]
9. Misra, R.; Mandal, C. Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks. IEEE Trans. Parallel Distrib. Syst. 2010, 21, 292–302. [CrossRef]
10. Thai, M.T.; Tiwari, R.; Du, D.Z. On construction of virtual backbone in wireless ad hoc networks with unidirectional links. IEEE Trans. Mob. Comput. 2008, 7, 1098–1109. [CrossRef]
11. Garey, M.; Johnson, D.S. Computers and Intractability, a Guide to the Theory of NP-Completeness; Freeman: San Francisco, CA, USA, 1979.
12. Kamei, S.; Kakugawa, H. A self-stabilizing distributed approximation algorithm for the minimum connected dominating set. Int. J. Found. Comput. Sci. 2010, 21, 459–476. [CrossRef]
13. Marathe, M.V.; Breu, H.; Hunt, H.B., III; Ravi, S.S.; Rosenkrantz, D.J. Simple heuristics for unit disk graphs. Networks 1995, 25, 59–68. [CrossRef]
14. Das, B.; Bharghavan, V. Routing in ad-hoc networks using minimum connected dominating sets. In Proceedings of the ICC’97-International Conference on Communications, Montreal, QC, Canada, 12 June 1997; pp. 376–380.
15. Wu, J.; Li, H. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Seattle, WA, USA, 20 August 1999; pp. 7–14.
16. Cheng, X.; Du, D.Z.; Jia, X.; Xue, G. Virtual backbone-based routing in multihop ad hoc wireless networks. IEEE Trans. Parallel Distrib. Syst. 2002. [CrossRef]
17. Wan, P.J.; Alzoubi, K.M.; Frieder, O. Distributed construction of connected dominating set in wireless ad hoc networks. In Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York, NY, USA, 23–27 June 2002; pp. 1597–1604.
18. Wu, J.; Lou, W. Forward-node-set-based broadcast in clustered mobile ad hoc networks. Wirel. Commun. Mob. Comput. 2003, 3, 155–173. [CrossRef]
19. Gao, B.; Yang, Y.; Ma, H. A new distributed approximation algorithm for constructing minimum connected dominating set in wireless ad hoc networks. Int. J. Commun. Syst. 2005, 18, 743–762. [CrossRef]
20. Simonetti, L.; da Cunha, A.S.; Lucena, A. The minimum Connected Dominating Set Problem: Formulation, Valid Inequalities and a Branch-and-Cut Algorithm. In Network Optimization; Springer: Berlin, Germany, 2011; pp. 162–169.

21. Gendron, B.; Lucena, A.; da Cunha, A.S.; Simonetti, L. Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. INFORMS J. Comput. 2014, 26, 645–657. [CrossRef]

22. Cheng, X.; Ding, M.; Chen, D. An approximation algorithm for connected dominating set in ad hoc networks. In Proceedings of the International Workshop on Theoretical Aspects of Wireless Ad Hoc, Sensor and Peer-to-Peer Networks (TAWN), Chicago, IL, USA, 11–12 June 2004.

23. Min, M.; Du, H.; Jia, X.; Huang, C.X.; Huang, S.C.H.; Wu, W. Improving construction for connected dominating set with steiner tree in wireless sensor networks. J. Glob. Optim. 2006, 35, 111–119. [CrossRef]

24. Butenko, S.; Cheng, X.; Oliveira, C.; Pardalos, P. A New Heuristic for the Minimum Connected Dominating Set Problem on ad hoc Wireless Networks. In Recent Developments in Cooperative Control and Optimization; Kluwer Academic Publishers: Norwell, MA, USA, 2004; pp. 61–73.

25. Butenko, S.; Oliveira, C.; Pardalos, P. A New Algorithm for the Minimum Connected Dominating Set Problem on ad hoc Wireless Networks. In CCCT’03; International Institute of Informatics and Systematics (IIFS): Winter Garden, FL, USA, 2003; pp. 39–44.

26. He, H.; Zhu, Z.; Makinen, E. A neural network model to minimize the connected dominating set for self-configuration of wireless sensor networks. IEEE Trans. Neural Netw. 2009, 20, 973–982. [PubMed]

27. Morgan, M.; Grout, V. Metaheuristics for Wireless Network Optimization. In Proceedings of the Third Advanced International Conference on Telecommunications, Morne, Mauritius, 13–19 May 2007; p. 15.

28. Jovanovic, R.; Tuba, M. Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Comput. Sci. Inf. Syst. 2013, 10, 133–149. [CrossRef]

29. Nimishia, T.S.; Ramalakshmi, R. Energy efficient Connected Dominating Set construction using Ant Colony Optimization technique in Wireless Sensor Network. In Proceedings of the Innovations in Information, Embedded and Communication Systems (ICIIES), Coimbatore, India, 19–20 March 2015; pp. 1–5.

30. Li, R.; Hu, S.; Gao, J.; Zhou, Y.; Wang, Y.; Yin, M. GRASP for connected dominating set problems. Neural Comput. Appl. 2017, 28, 1059–1067. [CrossRef]

31. Wang, L.; Zhou, T.; Wu, X. Variable-depth neighborhood search algorithm for the minimum-connected dominating-set problem. Sci. Sin. Inf. Sci. 2016, 46, 445–460.

32. Bouamama, S.; Blum, C.; Fages, J.G. An algorithm based on ant colony optimization for the minimum connected dominating set problem. Appl. Soft Comput. 2019, 80, 672–686. [CrossRef]

33. Cai, S.W.; Su, K.L.; Sattar, A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. 2011, 175, 1672–1696. [CrossRef]

34. Li, R.; Hu, S.; Zhang, H.; Yin, M. An efficient local search framework for the minimum weighted vertex cover problem. Inf. Sci. 2016, 372, 428–445. [CrossRef]

35. Li, R.; Hu, S.; Cai, S.; Gao, J.; Wang, Y.; Yin, M. NuMWVC: A novel local search for minimum weighted vertex cover problem. J. Oper. Res. Soc. 2019. [CrossRef]

36. Wang, Y.; Ouyang, D.; Zhang, L.; Yin, M. A novel local search for unicast set covering problem using hyperedge configuration checking and weight diversity. Sci. China Inf. Sci. 2017, 60, 062103. [CrossRef]

37. Cai, S.W.; Su, K.L. Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. 2013, 204, 75–98. [CrossRef]

38. Cai, S.; Su, K. Configuration checking with aspiration in local search for sat. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012.

39. Luo, C.; Cai, S.; Su, K.; Wu, W. Clause states based configuration checking in local search for satisfiability. IEEE Trans. Cybern. 2015, 45, 1014–1027. [PubMed]

40. Luo, C.; Cai, S.; Su, K.; Wu, W. Ccls: An efficient local search algorithm for weighted maximum satisfaction. IEEE Trans. Comput. 2015, 64, 1830–1843. [CrossRef]

41. Wang, Y.Y.; Cai, S.W.; Yin, M.H. Two efficient local search algorithms for maximum weight clique problem. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016.

42. Li, R.; Liu, H.; Wu, X.; Wu, J.; Yin, M. An Efficient Local Search Algorithm for the Minimum k-Dominating Set Problem. IEEE Access 2018, 6, 62062–62075. [CrossRef]
43. Ho, C.K.; Singh, Y.P.; Ewe, H.T. An enhanced ant colony optimization metaheuristic for the minimum dominating set problem. *Appl. Artif. Intell.* 2006, 20, 881–903. [CrossRef]

44. Lucena, A.; Maculan, N.; Simonetti, L. Reformulation and solution algorithms for the maximum leafspanning tree problem. *Comput. Manag. Sci.* 2010, 7, 289–311. [CrossRef]

45. Wei, Z.; Hao, J.K. Iterated two-phase local search for the Set-Union Knapsack Problem. *Future Gener. Comput. Syst.* 2019, 101, 1005–1017. [CrossRef]

46. Lu, Z.; Hao, J.K.; Zhou, Y. Stagnation-aware Breakout Tabu Search for the Minimum Conductance Graph Partitioning Problem. *Comput. Oper. Res.* 2019, 111, 43–57. [CrossRef]

47. Wang, Y.; Li, C.; Sun, H.; Chen, J.; Yin, M. MLQCC: An improved local search algorithm for the set k-covering problem. *Int. Trans. Oper. Res.* 2019, 26, 856–887. [CrossRef]

48. Zhou, Y.; Wang, Y.; Gao, J.; Luo, N.; Wang, J. An efficient local search for partial vertex cover problem. *Neural Comput. Appl.* 2018, 30, 2245–2256. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).