Local Search for Minimum Weight Dominating Set with Two-Level Configuration Checking and Frequency Based Scoring Function (Extended Abstract)

Yiyuan Wang[1], **Shaowei Cai**[2], **Minghao Yin**[1*]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
yiyuanwangjlu@126.com; caisw@ios.ac.cn; ymh@nenu.edu.cn

Abstract

The Minimum Weight Dominating Set (MWDS) problem is an important generalization of the Minimum Dominating Set (MDS) problem with extensive applications. This paper proposes a new local search algorithm for the MWDS problem, which is based on two new ideas. The first idea is a heuristic called two-level configuration checking (CC^2), which is a new variant of a recent powerful configuration checking strategy (CC) for effectively avoiding the recent search paths. The second idea is a novel scoring function based on the frequency of being uncovered of vertices. Our algorithm is called CC^{2}FS, according to the names of the two ideas. The experimental results show that, CC^{2}FS performs much better than some state-of-the-art algorithms in terms of solution quality on a broad range of MWDS benchmarks.

1 Introduction

Given an undirected graph G, a dominating set D is a subset of vertices such that every vertex not in D is adjacent to at least one member of D. The Minimum Dominating Set (MDS) problem consists in identifying the smallest dominating set in a graph. The Minimum Weight Dominating Set (MWDS) problem is a generalized version of MDS. In the MWDS problem, each vertex is associated with a positive value as its weight, and the task is to find a dominating set that minimizes the total weight of the vertices in it. The MWDS problem has played a prominent role in various real-world domains [Shen and Li, 2010; Golovach *et al.*, 2013], such as social networks, communication networks, and industrial applications.

Most practical algorithms for solving the MWDS problem are heuristic algorithms [Jovanovic *et al.*, 2010; Potluri and Singh, 2013; Nitash and Singh, 2014; Chaurasia and Singh, 2015; Bouamama and Blum, 2016]. However, the efficiency of existing heuristic algorithm are still not satisfactory, especially for hard and large-scaled instances (as will be shown in our experiments). The reason may be that the heuristic functions used in previous algorithms do not have enough information during the search procedure, and the cycling search problems can not be overcome by most algorithms as well.

In this paper, we develop a novel local search algorithm for the MWDS problem based on two new ideas. The first idea is a new variant of the Configuration Checking (CC) strategy. Initially proposed in [Cai *et al.*, 2011], the CC strategy aims to reduce the cycling phenomenon in local search, by considering the circumstance of the solution components, which is formally defined as the *configuration*. The CC strategy has been successfully applied to a number of well-known combinatorial optimization problems [Li *et al.*, 2016; Wang *et al.*, 2016b; 2016a; Luo *et al.*, 2015; Cai *et al.*, 2015]. In this work, we propose a variant of the CC strategy based on a new definition of configuration. In this strategy, the configuration of a vertex v refers to its two-level neighborhood, which is the union of the neighborhood $N(v)$ and the neighborhood of each vertex in $N(v)$. This new strategy is thus called two-level configuration checking (abbreviated as CC^2).

The second idea is a frequency based scoring function for vertices, according to which the score of each vertex is calculated. Local search algorithms for the MWDS problem maintain a candidate solution, which is a set of vertices selected for dominating. Then the algorithms will use a scoring function to decide which vertices will be selected to update the candidate solution, where the scores of vertices indicate the benefit (which may be positive or negative) produced by adding (or removing) a vertex to the candidate solution. In this work, we introduce a scoring function based on dynamic information of vertices, i.e., the frequency of being uncovered by the candidate solution. This scoring function exploits the information of the search process and that of the candidate solution.

By incorporating these two ideas, we develop a local search algorithm for the MWDS problem termed CC^{2}FS. We carry out experiments to compare CC^{2}FS with five state-of-the-art MWDS algorithms on benchmarks in the literatures including unit disk graphs and random generated instances, as well as two classical graphs benchmarks namely BHOSLIB [Xu *et al.*, 2007] and DIMACS [Johnson and Trick, 1996], and a broad range of real world massive graphs with millions of vertices and dozens of millions of edges [Rossi and Ahmed, 2015]. Experimental results show that CC^{2}FS significantly outperforms previous algorithms and improves the best known solution quality for some difficult instances.

[*]This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Wang *et al.*, 2017]

2 Preliminaries

An undirected graph $G = (V, E)$ comprises a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices together with a set $E = \{e_1, e_2, \ldots, e_m\}$ of m edges, where each edge $e = \{v, u\}$ connects two vertices u and v, and these two vertices are called the *endpoints* of edge e. The distance between two vertices u and v, denoted by $dist(u, v)$, is the number of edges in a shortest path from u to v, and $dist(u, u) = 0$ particularly. For a vertex v, we define its ith level neighborhood as $N_i(v) = \{u|dist(u, v) = i\}$, and we denote $N^k(v) = \bigcup_{i=1}^{k} N_i(v)$. The first-level neighborhood $N_1(v)$ is usually denoted as $N(v)$ as well, and we denote $N_i[v] = N_i(v) \cup \{v\}$. Also, we define the closed neighborhood of a vertex set S, $N[S] = \bigcup_{v \in S} N[v]$.

A dominating set of G is a subset $D \subseteq V$ such that every vertex in G either belongs to D or is adjacent to a vertex in D. The Minimum Dominating Set (MDS) problem calls for finding a dominating set of minimum cardinality. In the Minimum Weight Dominating Set (MWDS) problem, each vertex v is associated with a positive weight $w(v)$, and the task is to find a dominating set D which minimizes the total weight of vertices in D (i.e., min$\sum_{v \in D} w(v)$).

3 Two-Level Configuration Checking

In this section, we define the CC^2 strategy and present an implementation for it. We start from the formal definition of the configuration of a vertex v.

Definition 1 *Given an undirected graph $G = (V, E)$ and S the candidate solution, the configuration of a vertex $v \in V$ is a vector consisting of state of all vertices in $N^2(v)$.*

Based on the above definition, we can define an important vertex in local search as follows.

Definition 2 *Given an undirected graph $G = (V, E)$ and S the candidate solution, for a vertex $v \notin S$, v is configuration changed if at least one vertex in $N^2(v)$ has changed its state since the last time v is removed from S.*

In the CC^2 strategy, only the configuration changed vertices are allowed to be added to the candidate solution S.

We implement CC^2 with a Boolean array $ConfChange$ whose size equals the number of vertices in the input graph. For a vertex v, the value of $ConfChange[v]$ is an indicator — $ConfChange[v]$=1 means v is a configuration changed vertex and is allowed to be added to the candidate solution S; otherwise, $ConfChange[v]$=0 and it cannot be added to S. During the search procedure, the $ConfChange$ array is maintained as follows.

CC^2-RULE1. At the start of search process, for each vertex v, $ConfChange[v]$ is initialized as 1.

CC^2-RULE2. When removing a vertex v from the candidate solution S, $ConfChange[v]$ is set to 0, and for each vertex $u \in N^2(v)$, $ConfChange[u]$ is set to 1.

CC^2-RULE3. When adding a vertex v into the candidate solution S, for each vertex $u \in N^2(v)$, $ConfChange[u]$ is set to 1.

To understand RULE2 and RULE3, we note that if $u \in N^2(v)$, then $v \in N^2(u)$. Thus, if a vertex v changes its state (i.e., either being removed or added w.r.t. the candidate solution), the $Configuration$ of any vertex $u \in N^2(v)$ is changed.

The details of the relationship between CC and CC^2 strategies are presented in [Wang *et al.*, 2017].

4 The Frequency based Scoring Function

In this paper, we introduce a novel scoring function by taking into account of the vertices' frequency, which can be viewed as some kind of dynamic information indicating the accumulative effectiveness that the search has on the vertex. Intuitively, if a vertex is usually uncovered, then we should encourage the algorithm to select a vertex to make it covered.

In detail, in a graph, each vertex $v \in V$ has an additional property, frequency, denoted by $freq[v]$. The $freq$ of each vertex is initialized to 1. After each iteration of local search, the $freq$ value of each uncovered vertex is increased by one. During the search process, we apply the $freq$ of vertex to decide which vertex to be added or removed. Based on this consideration, we propose a new score function, which is formally defined as below.

Definition 3 *For a graph $G = (V, E)$, and a candidate solution S, the frequency based scoring function denoted by $score_f$, is a function such that*

$$score_f(u) = \begin{cases} \dfrac{1}{w(u)} \times \displaystyle\sum_{v \in C_1} freq[v], u \notin S, \\ -\dfrac{1}{w(u)} \times \displaystyle\sum_{v \in C_2} freq[v], u \in S, \end{cases} \tag{1}$$

where C_1=$N[u] \setminus N[S]$ and C_2=$N[u] \setminus N[S \setminus \{u\}]$.

Remark that, in the above definition, C_1 is indeed the set of uncovered vertices that would become covered by adding u into S and C_2 is the set of covered vertices that would become uncovered by removing u from S.

5 The Selection Vertex Strategy

During the search process, for preventing visiting previous candidate solutions, we not only use the CC^2 strategy in the adding process, but also use the forbidding list in the removing process. The $forbid\_list$ used here is a tabu list which keeps track of the vertices added in the last step, and these vertices are prevented from being removed within the tabu tenure. In this sense, this frequency based prohibition mechanism can be viewed as an instantiation of the longer term memory tabu search, and the main difference is that our method also consider the information from the CC^2 strategy.

The algorithm picks a vertex to add or remove, using the frequency based scoring function and the above two strategies. Firstly, we give two rules for removing vertices.

REMOVE-RULE1. Removing one vertex v, which has the highest value of $score_f(v)$, breaking ties by selecting the oldest one.

REMOVE-RULE2. Removing one vertex v, which is not in $forbid\_list$ and has the highest value of $score_f(v)$, breaking ties by selecting the oldest one.

When the algorithm finds a solution, it removes one vertex from the solution and continues to search for a solution with smaller weight. In this process, we use REMOVE-RULE1 to pick the vertex. During the search for a solution, the algorithm exchanges some vertices, i.e., removing one vertex from the candidate solution and then iteratively adding vertices into the candidate solution. In this case, we select one vertex to remove according to REMOVE-RULE2.

The rule to select the adding vertices is given below.

ADD-RULE. Adding one vertex v with $ConfChange[v] \neq 0$, which has the greatest value $score_f(v)$, breaking ties by selecting the oldest one.

When adding one vertex into the candidate solution, we try to make the resulting candidate solution's cost (i.e., the total weight of uncovered vertices) as small as possible. When adding one configuration changed vertex with the highest value $score_f(v)$, breaking ties by preferring the oldest vertex.

6 CC^{2}FS Algorithm

Based on CC^2 and the frequency based scoring function, we develop a local search algorithm named CC^{2}FS. During the process of local search, we maintain a set from which the vertex to be added is chosen. The set for finding a vertex to be removed from the candidate solution is simply S.

$CCV^2 = \{v|ConfChange[v] = 1, v \notin S\}$

The pseudo code of CC^{2}FS is shown in Algorithm 2. At first, CC^{2}FS initializes $ConfChange$, $forbid\_list$ and the $frequency$ and $score_f$ of vertices. Then it gets an initial candidate solution S greedily by iteratively adding the vertex that covers the most remaining uncovered vertices until S covers all vertices. At the end of initialization, the best solution S^* is updated by S.

After initialization, the main loop from lines 3 to 16 begins by checking whether S is a solution (i.e., covers all vertices). When the algorithm finds a better solution, S^* is updated. Then one vertex with the highest $score_f$ value in S is selected to be removed, breaking tie in favor of the oldest one. Finally, the values of $ConfChange$ are updated by CC^2-RULE2.

If there are uncovered vertices, CC^{2}FS first picks one vertex to remove from S with the highest value $score_f$, breaking tie in favor of the oldest one. Note that when choosing a vertex to remove, we do not consider those vertices in $forbid\_list$, as they are forbidden to be removed by the forbidden list. After removing a vertex, CC^{2}FS updates the $ConfChange$ values according to CC^2-RULE2, and clear $forbid\_list$. Additional, since the tabu tenure is set to be 1, the $forbid\_list$ shall be cleared to allow previous forbidden vertices to be added in subsequent loop.

After the removing process, CC^{2}FS iteratively adds one vertex into S until it covers all vertices, i.e. the candidate solution is a dominating set. CC^{2}FS first selects $v \in CCV^2$ with the greatest $score_f(v)$, breaking ties in favor of the oldest one. When the picked uncovered vertex is added into the candidate solution, the $ConfChange$ values are updated according to CC^2-RULE3 and this added vertex is added into the $forbid\_list$. After adding an uncovered vertex each time, the frequency of uncovered vertices is increased by one. When the time limit reaches, the best solution will be returned. For each iteration, the local search stage of CC^{2}FS has a time complexity of $O(max\{\Delta(G)|V|, \Delta(G)^3\})$, where $\Delta(G) = max\{|N[v]||v \in V, G = (V, E)\}$.

Algorithm 1: CC^{2}FS (G, *cutoff*)

```
   Input: a weighted graph G = (V, E, W), the cutoff time
   Output: dominating set of G
1  initialize ConfChange, forbid_list, and the freq and
   score_f of of vertices;
2  S := InitGreedyConstruction() and S* := S;
3  while elapsed time < cutoff do
4      if there are no uncovered vertices then
5          if w(S) < w(S*) then S* := S;
6          v := a vertex in S with the highest value score_f(v),
           breaking ties in the oldest one;
7          S := S \ {v} and update ConfChange according to
           CC^2-RULE2;
8          continue;
9      v := a vertex in S with the highest value score_f(v) and
       v ∉ forbid_list, breaking ties in the oldest one;
10     S := S \ {v} and update ConfChange according to
       CC^2-RULE2;
11     forbid_list := ∅;
12     while there are uncovered vertices do
13         v := a vertex in CCV^2 with the highest value
           score_f(v), breaking ties in the oldest one;
14         S := S ∪ {v} and update ConfChange according to
           CC^2-RULE3;
15         forbid_list := forbid_list ∪ {v};
16         freq[v] := freq[v] + 1, for v ∉ N[S];
17 return S*;
```

7 Empirical Results

We compare CC^{2}FS with five competitors on a broad range of benchmarks, with respect to both solution quality and run time. The run time is measured in CPU seconds. We run CC^{2}FS on a broad range of test instances, namely T1, T2, UDG, DIMACS, BHOSLIB, as well as many real world massive graphs. For T1 and T2 instances [Jovanovic *et al.*, 2010], we note that ten instances are generated for each combination of number of nodes and transmission range.

We compare CC^{2}FS with HGA [2013], ABC [2014], ACO-PP-LS [2013], EA/G-IR [2015], and R-PBIG [2016]. Among them, R-PBIG and ACO-PP-LS are the best available algorithms for solving MWDS.

We implement CC^{2}FS in C++ and compile it by g++ with the -O2 option. All the experiments are run on Ubuntu Linux, with 3.1 GHZ CPU and 8GB memory. For T1 and T2 instances, CC^{2}FS and ACO-PP-LS are performed once, where one run is terminated upon reaching a given time limit. Among this, the parameter time limit is set to 50 seconds when the number of vertices is less than 500, otherwise the time limit is set to 1000 seconds. We report the real time $RTime$ of ACO-PP-LS and CC^{2}FS, while we also give the finial execution time $FTime$ of EA/G-IR and R-PBIG. The real time is a time when ACO-PP-LS and CC^{2}FS obtain the best solution respectively. The MEAN contains the average solution

Table 1: Experiment results of HGA, ACO-PP-LS, ABC, EA/G-IR, R-PBIG, and CC^{2}FS on the T1 benchmark.

Instance	HGA	ACO-PP-LS		ABC	EA/G-IR		R-PBIG		CC2FS	
T1	MEAN	MEAN	RTime	MEAN	MEAN	FTime	MEAN	FTime	MEAN	RTime
v50e50	**531.3**	**531.3**	0.2	534	532.9	0.21	**531.3**	0.5	**531.3**	<0.01
v50e100	371.2	371.2	0.17	371.2	371.5	0.23	371.1	0.8	**370.9**	<0.01
v50e250	**175.7**	**175.7**	0.12	**175.7**	**175.7**	0.18	**175.7**	1.3	**175.7**	<0.01
v50e500	**94.9**	**94.9**	0.05	**94.9**	**94.9**	0.16	95	2.3	**94.9**	<0.01
v50e750	**63.1**	**63.1**	0.03	**63.1**	63.3	0.1	63.8	2.5	63.3	<0.01
v50e1000	**41.5**	**41.5**	0.01	**41.5**	**41.5**	0.1	**41.5**	3	**41.5**	<0.01
v100e100	1081.3	1065.6	2.05	1077.7	1065.5	0.76	1061.9	1.4	**1061**	0.04
v100e250	626.2	623.1	1.3	621.6	620	0.72	619.3	2.2	**618.9**	0.04
v100e500	358.3	360.6	0.54	356.4	355.9	0.6	356.5	3	**355.6**	<0.01
v100e750	261.2	261	0.46	255.9	256.7	0.52	256.5	3.7	**255.8**	<0.01
v100e1000	205.6	207.3	0.38	**203.6**	**203.6**	0.49	**203.6**	4.3	**203.6**	<0.01
v100e2000	108.2	108.4	0.2	108.2	108.1	0.45	108	6.2	**107.4**	0.19
v150e150	1607	1582	4.88	1607.9	1587.4	1.64	1582.5	2.4	**1580.5**	0.02
v150e250	1238.6	1228.4	3.67	1231.2	1224.5	1.71	1219.5	3.3	**1218.2**	0.06
v150e500	763	763	2.2	752.1	755.3	1.45	745	4.3	**744.6**	0.05
v150e750	558.5	554	1.46	549.3	550.8	1.27	548.2	5.2	**546.1**	0.04
v150e1000	438.7	440.7	1.34	435.1	435.2	1.11	433.6	5.9	**432.9**	0.03
v150e2000	245.7	251.8	0.89	242.2	241.5	0.88	241.5	8.7	**240.8**	0.17
v150e3000	169.2	171.4	0.8	167.8	168.1	0.82	168.4	11.1	**166.9**	0.06
v200e250	1962.1	1919.5	9.54	1941.1	1924.1	3.68	1914.6	4.7	**1910.4**	0.19
v200e500	1266.3	1252.9	4.92	1246.9	1251.3	3.3	1235.3	6.2	**1232.8**	1.01
v200e750	939.8	934.3	3.71	923.7	927.3	2.78	914.9	7.4	**911.2**	0.45
v200e1000	747.8	741.8	2.99	730.4	731.1	2.39	725.2	8.2	**724**	0.25
v200e2000	432.9	437.3	1.36	417.6	417	1.68	414.8	10.9	**412.7**	0.45
v200e3000	308.5	308.8	1.24	294.4	294.7	1.42	294.2	14.2	**292.8**	0.41
v250e250	2703.4	2646.6	16.09	2685.7	2653.7	4.65	2653.7	6	**2633.4**	0.2
v250e500	1878.8	1840.1	10.91	1836	1853.3	4.66	1812.6	8.6	**1805.9**	0.92
v250e750	1421.1	1396.8	7.15	1391.9	1399.2	4.25	1368.6	9.6	**1362.2**	0.65
v250e1000	1143.4	1120.2	5.53	1115.3	1114.9	3.69	1097.1	10.9	**1091.1**	0.48
v250e2000	656.6	666	3.23	630.5	637.5	2.65	624.7	14	**621.9**	0.44
v250e3000	469.3	469.4	2.64	454.9	456.3	2.16	451.5	17.9	**447.9**	0.72
v250e5000	300.5	307	2.29	292.4	291.8	5.16	291.5	25.4	**289.5**	0.17
v300e300	3255.2	3190.6	24.39	3240.7	3213.7	8.73	3189.3	7.6	**3178.6**	1.47
v300e500	2509.8	2461.4	21	2484.6	2474.8	7.21	2446.9	10.2	**2438.1**	1.46
v300e750	1933.9	1885.1	16.91	1901.4	1896.3	6.48	1869.6	12	**1854.6**	1.6
v300e1000	1560.1	1532.7	10.49	1523.4	1531	5.7	1503.4	13.2	**1495**	0.61
v300e2000	909.6	900.5	5.95	875.5	880.1	4.03	872.5	16.8	**862.5**	1.93
v300e3000	654.9	658.8	4.03	635.3	638.2	3.27	629	21.3	**624.3**	1.12
v300e5000	428.3	432.3	3.67	411	415.7	2.59	409.4	29.8	**406.1**	1.72
v500e500	5498.3	5370.4	99.09	5480.1	5380.1	37.52	5378.4	21.1	**5305.7**	2.63
v500e1000	3798.6	3675.8	64.96	3707.6	3695.2	26.36	3642.2	29.1	**3607.8**	4.24
v500e2000	2338.2	2236.2	32.85	2266.1	2264.3	25.54	2203.9	36.1	**2181**	4.83
v500e5000	1122.7	1105.8	15.57	1070.9	1083.5	9.02	1055.9	55.9	**1043.3**	5.07
v500e10000	641.1	640.9	10.57	596	606.8	6.08	596.3	76.2	**587.2**	6.19
v800e1000	8017.7	7991.6	174.62	7907.3	7792.2	129.82	7768.6	67.9	**7663.4**	10.58
v800e2000	5317.7	5298.4	95.34	5193.2	5160.7	102.09	5037.9	83.3	**4982.1**	8.83
v800e5000	2633.4	2578.8	59.23	2548.6	2561.9	53.02	2465.4	122.6	**2441.2**	6.58
v800e10000	1547.7	1512.7	41.86	1471.7	1497	31.17	1420	171.1	**1395.6**	6.84
v1000e1000	11095.2	10984.9	412.14	10992.4	10771.7	249.82	10825	96.9	**10585.3**	12.18
v1000e5000	3996.6	3977.7	91.07	3853.7	3876.3	107.41	3693.1	184.2	**3671.8**	8.49
v1000e10000	2334.7	2291.8	83.82	2215.9	2265.1	63.22	2140.3	254.9	**2109**	9.43
v1000e15000	1687.5	1647.4	63.15	1603.2	1629.4	45.86	1549.1	282	**1521.5**	11.91
v1000e20000	1337.2	1297.5	44.21	1259.5	1299.9	36.35	1219	289.8	**1203.6**	11.4

Table 2: Experiment results of HGA, ACO-PP-LS, ABC, EA/G-IR, R-PBIG, and CC^{2}FS on the T2 benchmark.

Instance	HGA	ACO-PP-LS		ABC	EA/G-IR		R-PBIG		CC^{2}FS	
T2	MEAN	MEAN	RTime	MEAN	MEAN	FTime	MEAN	FTime	MEAN	RTime
v50e50	**60.8**	**60.8**	0.08	**60.8**	60.8	0.19	**60.8**	0.5	**60.8**	<0.01
v50e100	**90.3**	**90.3**	0.17	**90.3**	90.3	0.27	**90.3**	0.8	**90.3**	<0.01
v50e250	**146.7**	**146.7**	0.09	**146.7**	146.7	0.23	**146.7**	1.4	**146.7**	<0.01
v50e500	**179.9**	**179.9**	0.04	**179.9**	179.9	0.09	**179.9**	2.1	**179.9**	<0.01
v50e750	**171.1**	**171.1**	0.01	**171.1**	171.1	0.07	**171.1**	2.4	**171.1**	<0.01
v50e1000	**146.5**	**146.5**	0.01	**146.5**	146.5	0.06	**146.5**	2.9	**146.5**	<0.01
v100e100	124.5	**123.5**	1.05	124.4	123.5	0.6	**123.5**	1.2	**123.5**	<0.01
v100e250	211.4	210.1	0.89	209.6	**209.2**	0.92	**209.2**	2.1	**209.2**	<0.01
v100e500	306	**305.7**	0.57	305.8	**305.7**	0.78	**305.7**	2.9	**305.7**	<0.01
v100e750	385.3	**384.5**	0.45	**384.5**	**384.5**	0.7	386.9	3.5	**384.5**	<0.01
v100e1000	429.1	427.7	0.21	**427.3**	**427.3**	0.67	**427.3**	4.2	**427.3**	<0.01
v100e2000	**550.6**	**550.6**	0.15	**550.6**	**550.6**	0.54	552.7	6.3	**550.6**	<0.01
v150e150	186	**184.5**	2.9	185.9	**184.5**	1.85	**184.5**	2.1	**184.5**	0.07
v150e250	234.9	233	2.7	233.4	232.8	2.03	**232.8**	3.1	**232.8**	<0.01
v150e500	350	350.3	1.49	**349.5**	349.7	1.95	349.7	4.4	**349.5**	<0.01
v150e750	455.8	453	1.81	453.7	**452.4**	1.78	**452.4**	5.4	**452.4**	<0.01
v150e1000	547.5	549	1.3	547.8	548.2	1.61	547.8	6	**547.2**	<0.01
v150e2000	**720.1**	720.8	0.88	**720.1**	**720.1**	1.2	**720.1**	8.4	**720.1**	<0.01
v150e3000	792.6	**792.4**	0.56	793.2	**792.4**	1.07	793.2	11.7	**792.4**	0.66
v200e250	275.1	272.2	5.01	273.5	272.3	4.38	**271.7**	4.3	**271.7**	<0.01
v200e500	390.7	387.4	3.71	387.6	388.4	4.51	386.8	6.1	**386.7**	0.04
v200e750	507	499.7	3.56	498.5	497.2	4.18	**497.1**	7.2	**497.1**	<0.01
v200e1000	601.1	598.9	2.69	599.3	598.2	3.89	**596.8**	8.5	**596.8**	0.04
v200e2000	893.5	887.3	1.88	885.5	885.8	2.78	**884.6**	11.4	**884.6**	0.09
v200e3000	1021.3	1027	1.01	1021.3	1019.7	2.16	**1019.2**	14.1	**1019.2**	0.06
v250e250	310.1	306.5	8.86	308.6	306.5	5.26	**306**	4.9	306.1	0.01
v250e500	444	441.9	9.11	442.6	441.6	6.1	441	8.2	**440.7**	0.16
v250e750	578.2	571.4	7.64	569.9	569.2	6.09	567.9	10	**567.4**	0.2
v250e1000	672.8	671.5	4.81	670.3	671.7	5.89	669.2	11.4	**668.6**	0.17
v250e2000	1030.8	1018.9	3.88	1010.4	1010.3	4.23	1009.5	14.5	**1007**	0.48
v250e3000	1262	1261.2	2.75	1251.3	**1250.6**	3.5	1251.6	18.1	**1250.6**	0.57
v250e5000	1480.9	1469.6	1.36	1464.7	**1464.2**	2.59	**1464.2**	25.5	**1464.2**	0.01
v300e300	375.6	371.1	14.46	373.5	370.5	9.01	**369.9**	6.3	**369.9**	0.13
v300e500	484.2	479.9	11.93	481.6	480	8.83	478	9.6	**477.8**	0.06
v300e750	623.8	616.1	13.63	617.6	613.8	7.57	613.6	11.7	**613.3**	0.37
v300e1000	751.1	740.9	11.37	743.6	742.2	8.96	738.3	13.5	**737.9**	0.28
v300e2000	1106.7	1104.5	6.86	1095.9	1094.9	6.67	1094.6	17.4	**1093.8**	0.03
v300e3000	1382.1	1398.4	6.25	1361.7	1359.5	5.41	**1358.5**	20.9	**1358.5**	0.08
v300e5000	1686.3	1691.5	3.21	**1682.7**	1683.6	4.02	1683.2	29.5	**1682.7**	0.01
v500e500	632.9	627.3	33.4	630.4	625.8	31.06	624.2	17.7	**623.6**	0.29
v500e1000	919.2	907.6	70.92	906.7	906	28.27	901.3	28.1	**899.8**	2.08
v500e2000	1398.2	1381.5	38.78	1383.6	1376.7	23.41	1364.4	37.2	**1363.3**	2.28
v500e5000	2393.2	2406.9	11.87	2337.9	2340.3	17.36	2341.5	59	**2333.7**	0.31
v500e10000	3264.9	3277.9	6.48	**3211.5**	3216.4	10.8	3216.1	80.5	**3211.5**	0.06
v800e1000	1128.2	1121.7	274.35	1119.2	1107.9	132.36	1107.6	59.6	**1104.3**	2.36
v800e2000	1679.2	1674.9	97.55	1656.4	1641.7	111.84	1634.6	83.5	**1632.3**	3.59
v800e5000	3003.6	3065.7	47.02	2917.4	2939.3	68.14	2884.8	128.3	**2878.5**	3.65
v800e10000	4268.1	4357.1	26.77	4121.3	4155.1	40.15	**4103.7**	183.9	4105.6	1.55
v1000e1000	1265.2	1254.4	564.71	1256.2	1240.8	202.08	1243.6	80.9	**1237.7**	0.86
v1000e5000	3320.1	3371.6	95.9	3240.7	3222	132.94	3195.7	196	**3178.7**	8.87
v1000e10000	4947.5	5041.6	55.26	4781.2	4798.6	84.82	4722.4	274.6	**4711.8**	4.06
v1000e15000	6267.6	6336.1	46.07	5931	5958.1	61.64	5884.2	305.2	**5874.2**	2.97
v1000e20000	7088.5	7166.7	37.65	6729	6775.8	59.2	6678	319.2	**6662.1**	2.68

values for each of the ten instances of graphs of a particular size.

The performance results of previous algorithms on the T1 benchmark are displayed in Table 1. More importantly, this table also summarizes the experimental results on the first benchmark for our algorithm.

Among previous algorithms, for most instances, R-PBIG and ACO-PP-LS can find better solutions than HGA, ABC and EA/G-IR, with only a few exceptions.

For our algorithm, we show the minimum solution value and the run time. As is clear from the Table 1, CC^{2}FS shows significant superiority on the T1 benchmark, except v50e750. By comparing these algorithms, we can easily conclude that CC^{2}FS outperforms other algorithms.

The experimental results on the T2 benchmark are presented in Table 2. The quality of the solutions found by CC^{2}FS is always much smaller than those found by other algorithms on all instances with 2 exceptions, i.e. v250e250 and v800e10000. Details of the experiemental results as well as the analysis based on benchmarks of the DIMACS, BHOSLIB, and real world massive graphs are presented in [Wang *et al.*, 2017].

8 Conclusions

This paper presented a local search algorithm called CC^{2}FS for solving the minimum weight dominating set (MWDS) problem. We proposed a new configuration checking strategy namely CC^2 based on the two-level neighborhood of vertices to remember the relevant information of removed and added vertices and prevent visiting the recent paths. Moreover, we introduced a new frequency based scoring function for solving MWDS. The experimental results showed that CC^{2}FS performs essentially better than state of the art algorithms on almost all instances in terms of solution quality and run time.

Acknowledgments

This work was supported in part by NSFC (under Grant Nos. 61370156, 61503074, 61502464, 61402070, and 61403077).

References

[Bouamama and Blum, 2016] Salim Bouamama and Christian Blum. A hybrid algorithmic model for the minimum weight dominating set problem. *Simulation Modelling Practice and Theory*, 64:57–68, 2016.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9):1672–1696, 2011.

[Cai *et al.*, 2015] Shaowei Cai, Zhong Jie, and Kaile Su. An effective variable selection heuristic in SLS for weighted Max-2-SAT. *Journal of Heuristics*, 21(3):433–456, 2015.

[Chaurasia and Singh, 2015] Sachchida Nand Chaurasia and Alok Singh. A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set. *Applied Intelligence*, 43(3):512–529, 2015.

[Golovach *et al.*, 2013] Petr A Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An incremental polynomial time algorithm to enumerate all minimal edge dominating sets. In *Automata, Languages, and Programming*, pages 485–496. 2013.

[Johnson and Trick, 1996] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[Jovanovic *et al.*, 2010] Raka Jovanovic, Milan Tuba, and Dana Simian. Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th WSEAS international conference on Automatic control, modelling & simulation*, pages 322–326. World Scientific and Engineering Academy and Society, 2010.

[Li *et al.*, 2016] Ruizhi Li, Shuli Hu, Haochen Zhang, and Minghao Yin. An efficient local search framework for the minimum weighted vertex cover problem. *Information Sciences*, 372:428–445, 2016.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Kaile Su, and Wei Wu. Clause states based configuration checking in local search for satisfiability. *Cybernetics, IEEE Transactions on*, 45(5):1014–1027, 2015.

[Nitash and Singh, 2014] CG Nitash and Alok Singh. An artificial bee colony algorithm for minimum weight dominating set. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–7. IEEE, 2014.

[Potluri and Singh, 2013] Anupama Potluri and Alok Singh. Hybrid meta-heuristic algorithms for minimum weight dominating set. *Applied Soft Computing*, 13(1):76–88, 2013.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 4292–4293, 2015.

[Shen and Li, 2010] Chao Shen and Tao Li. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 984–992. Association for Computational Linguistics, 2010.

[Wang *et al.*, 2016a] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 805–811, 2016.

[Wang *et al.*, 2016b] Yiyuan Wang, Minghao Yin, Dantong Ouyang, and Liming Zhang. A novel local search algorithm with configuration checking and scoring mechanism for the set k-covering problem. *International Transactions in Operational Research*, pages 1–23, 2016.

[Wang *et al.*, 2017] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Local search for minimum weighted dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, 58:267–295, 2017.

[Xu *et al.*, 2007] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8):514–534, 2007.