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

Yiyuan Wang1, Shaowei Cai2, Minghao Yin1*

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

yiyuanwangjl@126.com; cai-sw@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 (CC2), 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 CC2FS, according to the names of the two ideas. The experimental results show that, CC2FS 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 algorithms 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., 2016a; 2016b; 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 CC2).

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 CC2FS. We carry out experiments to compare CC2FS 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 CC2FS 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 \( v \) and \( u \), and these two vertices are called the endpoints of edge \( e \). The distance between two vertices \( v \) and \( u \), denoted by \( \text{dist}(u, v) \), is the number of edges in a shortest path from \( u \) to \( v \), and \( \text{dist}(u, u) = 0 \) particularly. For a vertex \( v \), we define its ith level neighborhood as \( N_i(v) = \{u|\text{dist}(u, v) = i\} \), 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 defined 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 \( \text{ConfChange} \) whose size equals the number of vertices in the input graph. For a vertex \( v \), the value of \( \text{ConfChange}[v] \) is an indicator — \( \text{ConfChange}[v] = 1 \) means \( v \) is a configuration changed vertex and is allowed to be added to the candidate solution \( S \); otherwise, \( \text{ConfChange}[v] = 0 \) and it cannot be added to \( S \). During the search procedure, the \( \text{ConfChange} \) array is maintained as follows.

CC\(^2\)-RULE1. At the start of search process, for each vertex \( v \), \( \text{ConfChange}[v] \) is initialized as 1.

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

CC\(^2\)-RULE3. When adding a vertex \( v \) into the candidate solution \( S \), for each vertex \( u \in N^2(v) \), \( \text{ConfChange}[u] \) 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 \( \text{freq}[v] \). The \( \text{freq} \) of each vertex is initialized to 1. After each iteration of local search, the \( \text{freq} \) value of each uncovered vertex is increased by one. During the search process, we apply the \( \text{freq} \) 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 \( \text{score}_f \), is a function such that

\[
\text{score}_f(u) = \begin{cases} 
\frac{1}{w(u)} \times \sum_{v \in C_1} \text{freq}[v], u \notin S, \\
-\frac{1}{w(u)} \times \sum_{v \in C_2} \text{freq}[v], u \in S,
\end{cases}
\]

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 \( \text{forbid}_\text{list} \) 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 \( \text{score}_f(v) \), breaking ties by selecting the oldest one.

REMOVE-RULE2. Removing one vertex \( v \), which is not in \( \text{forbid}_\text{list} \) and has the highest value of \( \text{score}_f(v) \), 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 \( \text{ConfChange}[v] \neq 0 \), which has the greatest value \( \text{score}_f(v) \), 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 \( \text{score}_f(v) \), 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 \( S \).

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

The pseudo code of CC\(^2\)FS is shown in Algorithm 2. At first, CC\(^2\)FS initializes \( \text{ConfChange} \), \( \text{forbid} \_\text{list} \) and the frequency and \( \text{score}_f \) 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 \( \text{score}_f \) value in \( S \) is selected to be removed, breaking tie in favor of the oldest one. Finally, the values of \( \text{ConfChange} \) 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 \( \text{score}_f \), breaking tie in favor of the oldest one. Note that when choosing a vertex to remove, we do not consider those vertices in \( \text{forbid} \_\text{list} \), as they are forbidden to be removed by the forbidden list. After removing a vertex, CC\(^2\)FS updates the \( \text{ConfChange} \) values according to CC\(^2\)-RULE2, and clear \( \text{forbid} \_\text{list} \). Additional, since the tabu tenure is set to be 1, the \( \text{forbid} \_\text{list} \) 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 \( v \in \text{CCV}^2 \) with the greatest \( \text{score}_f(v) \), breaking ties in favor of the oldest one. When the picked uncovered vertex is added into the candidate solution, the \( \text{ConfChange} \) values are updated according to CC\(^2\)-RULE3 and this added vertex is added into the \( \text{forbid} \_\text{list} \). 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 \( \Delta(G) = max\{|N[v]||v \in V, G = (V, E)|\} \).

### 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 \( \text{RTime} \) of ACO-PP-LS and CC\(^2\)FS, while we also give the final execution time \( \text{FinalTime} \) 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
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 2 exceptions, i.e. v250e250 and v800e10000. Details of the experimental results as algorithms on all instances with 2 exceptions, i.e. v250e250 and v800e10000. By comparing these algorithms, we can easily conclude that FS outperforms other algorithms.

The experimental results on the T2 benchmark are presented in Table 2. The quality of the solutions found by CC²FS is always much smaller than those found by other algorithms on all instances with 2 exceptions, i.e. \(v250e250\) and \(v800e10000\). Details of the experimental 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 CC²FS for solving the minimum weight dominating set (MDWS) problem. We proposed a new configuration checking strategy namely CC² 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 MDWS. The experimental results showed that CC²FS 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 Boussenart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8):514–534, 2007.