A Fast Local Search Algorithm for Minimum Weight Dominating Set Problem on Massive Graphs

Yiyuan Wang\textsuperscript{1,3}, Shaowei Cai\textsuperscript{2}, Jiejiang Chen\textsuperscript{1,3} and Minghao Yin\textsuperscript{1,3*}

\textsuperscript{1} School of Computer Science and Information Technology, Northeast Normal University, China
\textsuperscript{2} State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
\textsuperscript{3} Key Laboratory of Symbol Computation and Knowledge Engineering of Ministry of Education, Jilin University, China

yiyuanwangjlu@126.com, caisw@ios.ac.cn, chenjj016@nenu.edu.cn, ymh@nenu.edu.cn

Abstract

The minimum weight dominating set (MWDS) problem is NP-hard and also important in many applications. Recent heuristic MWDS algorithms can hardly solve massive real world graphs effectively. In this paper, we design a fast local search algorithm called FastMWDS for the MWDS problem, which aims to obtain a good solution on massive graphs within a short time. In this novel local search framework, we propose two ideas to make it effective. Firstly, we design a new fast construction procedure with four reduction rules to cut down the size of massive graphs. Secondly, we propose the three-valued two-level configuration checking strategy to improve local search, which is interestingly a variant of configuration checking (CC) with two levels and multiple values. Experiment results on a broad range of massive real world graphs show that FastMWDS finds much better solutions than state-of-the-art MWDS algorithms.

1 Introduction

Given a graph $G = (V, E)$, a dominating set is to find a subset $D$ of vertices $V$ such that every vertex belongs to $D$ or is adjacent to at least one vertex in $D$. The minimum dominating set (MDS) problem aims to identify the dominating set with the smallest size in a graph. An important generalization of MDS is the minimum weight dominating set (MWDS) problem, in which each vertex is associated with a positive integer, and the goal is to find a dominating set with the smallest weight. MWDS is an important combinatorial optimization problem with lots of valuable applications in many fields [Hedetniemi et al., 2003; Subhadrabandhu et al., 2004; Aoun et al., 2006; Chalupa, 2018]. For example, Wu et al. [2006] try to select good queries to rapidly harvest data records from Web databases, which has been proved to be equivalent to finding an MWDS of the corresponding database graph. Also, Shen and Li [2010] solve the multi-document problem by encoding this problem to the MWDS problem.

*Corresponding author

The MWDS problem can be encoded into the weighted partial maximum satisfiability (WPMS) problem and effectively solved by WPMS algorithms. MWDS is a special class of the subset selection problem [Qian et al., 2015] and the minimum weight set covering (MWSC) problem [Gao et al., 2014] which has many applications, such as service location and information retrieval [Caprara et al., 1997; Ceria et al., 1998; Bautista and Pereira, 2006].

MWDS is a classical NP-hard problem, which means there are no polynomial-time algorithms for the MWDS problem, unless NP=\text{P}. Approximation algorithms with good approximation ratios have been designed for special subclass of the MWDS problem. For example, an approximation scheme achieves a $(1+\varepsilon)$-approximation ratio ($\varepsilon > 0$) [Zhu et al., 2012] for unit disk graphs with smooth weights. Nevertheless, the general problem of MWDS remains hard to approximate, and approximation algorithms usually have poor performance in practice, especially for massive data sets.

1.1 Related Work

Because of its NP-hardness, many researchers on solving the MWDS problem focus on heuristic algorithms for obtaining a good weighted dominating set within a reasonable time. In the recent decade, various heuristic algorithms have been developed for solving the MWDS problem. A classical ant colony optimization ACO was proposed for solving the MWDS problem, by using the weight of each covered vertices as the scoring function [Jovanovic et al., 2010]. The ACO algorithm was further improved by taking into account the pheromone deposit on every vertex, resulting in the ACO-PP-LS algorithm [Potluri and Singh, 2013]. A local search MWSC algorithm which was used to solve the MWDS problem was proposed to perturb the candidate solution by a weighting scheme and tabu strategy [Gao et al., 2014]. A swarm intelligence algorithm named ABC applied an artificial bee colony method for tackling the MWDS problem [Nitash and Singh, 2014]. Chaurasia and Singh designed a hybrid MWDS algorithm called EA/G-IR by using an evolutionary algorithm and a guided mutation [Chaurasia and Singh, 2015]. An effective hybrid memetic MWDS algorithm HMA which was formulated as a constrained 0-1 programming problem was proposed and a memetic algorithm was
introduced to solve the resulting problem [Lin et al., 2016]. A randomized population-based iterated greedy MWDS algorithm R-PBIG was presented, which applied the iterated greedy algorithm to update each population [Bouamama and Blum, 2016]. Chalupa [2018] designed a multi-start variant of order-based randomised local search MSRLS, to solve MWDS. According to the literature, the current best heuristic for the MWDS problem is called CC²FS, which is based on two-level configuration checking and frequency based scoring function, and has better performance on a wide range of benchmarks than other MWDS algorithms [Wang et al., 2017a]. Moreover, CC²FS firstly tries to solve massive graphs and obtains some promising results.

1.2 Contributions and Paper Organization

Massive data sets [Rossi and Ahmed, 2015] can be modelled as massive graphs, and extensive studies have been carried out recently to tackle NP-hard problems on massive graphs [Wang et al., 2016; Jiang et al., 2017; Gao et al., 2017; Lin et al., 2017; Cai et al., 2017]. Although recent works have made great progress in solving the MWDS problem on some standard benchmarks, the improvements are limited on massive graphs. Therefore, we focus on solving massive graphs. We develop a fast local search algorithm for the MWDS problem called FastMWDS, which includes two phases, i.e., construction phase and local search phase. We are dedicated to reducing the time complexity of each phase. To improve the performance of FastMWDS on massive graphs, we design two heuristics, which are important in each phase of FastMWDS.

Firstly, we design a fast heuristic called ConstructDS for constructing a weighted dominating set. This heuristic can be divided into three parts: reducing, constructing, shrinking. Four reduction rules are proposed for the MWDS problem and used in ConstructDS procedure. The obtained solution will be used as the initial solution for the local search phase.

Secondly, we propose a three-valued two-level configuration checking strategy (CC²V3) to deal with the cycling problem in local search. The configuration checking strategy was firstly proposed by Cai [2011] and has been already used in many problems, including clique problem [Wang et al., 2016], vertex cover [Cai et al., 2013], dominating set [Wang et al., 2017a], boolean satisfiability [Abramé et al., 2017], maximum satisfiability [Luo et al., 2015], and set covering [Wang et al., 2017b]. Recently, the two-level configuration checking (CC²) [Wang et al., 2017a] was designed to solve the MWDS problem. Our CC²V3 strategy can be viewed as a multiple-value version of CC². While CC² only considers whether a vertex is configuration changed or not, CC²V3 distinguishes configuration changed vertices by different values.

We also apply a variant of the probabilistic heuristic “best from multiple selection” (BMS) [Cai et al., 2017] that combines random walk into FastMWDS. For massive graphs, when selecting the removal vertices from the solution, there exist too many candidate vertices. Therefore, this heuristic can decrease the time complexity of this part.

We carry out experiments to compare FastMWDS with four state of the art MWDS algorithms on a broad range of massive graphs from [Rossi and Ahmed, 2015]. Experiments results show that FastMWDS performs significantly better than the competitors, indicating the effectiveness of the proposed heuristics.

In the next section, we introduce some preliminary knowledge, including some definitions and notations. In Sections 3 and 4, we describe ConstructDS and CC²V3. After that, we present the FastMWDS algorithm. Then, we carry out our experiments to evaluate FastMWDS. Finally, we give some concluding marks.

2 Preliminaries

2.1 Definitions and Notations

An undirected graph $G = (V, E)$ consists of a vertex set $V$ and an edge set $E \subseteq V \times V$ in which each edge is a 2-element subset of $V$. For an edge $e = \{u, v\}$, vertices $u$ and $v$ are the endpoints of the edge $e$. A vertex weighted undirected graph is an undirected graph in which each vertex $v \in V$ is associated with a positive weight $w(v)$. We use $G = (V, E, w)$ to denote a vertex weighted graph. Two vertices are neighbors if and only if they both belong to one edge. $N(v) = \{u \in V \mid \{u, v\} \in E\}$ is the set of neighbors of a vertex $v$. The degree of a vertex $v$ is defined as $\deg(v) = |N(v)|$. $\text{dis}(u, v)$ is used to denote the number of edges in the shortest path from $u$ to $v$, i.e., the distance between these two vertices. We define its $i$th level neighborhood $N_i(v) = \{u \mid \text{dis}(u, v) = i\}$, as well as we denote $N^k(v) = \bigcup_{i=1}^{k} N_i(v)$ and $N_i[v] = N_i(v) \cup \{v\}$. We can easily find that $N_1(v) = N(v)$ and $N_1[v] = N[v]$. For a vertex set $S \subseteq V$, we use $N[S] = \bigcup_{v \in S} N[v]$ to denote the closed neighborhood of $S$.

Given a vertex weighted graph $G = (V, E, w)$, a candidate solution for the MWDS problem is a subset of vertices. A vertex $v \in V$ is dominated by a candidate solution $D$ if $v \in N[D]$, and is non-dominated otherwise. During the search procedure, FastMWDS always maintains a current candidate solution. For convenience, we use $D$ to denote the current candidate solution, i.e., the set of vertices currently selected for dominating. The age of a vertex is the number of steps since its state was last changed.

2.2 Preliminaries of Scoring Function for MWDS

Recently, the frequency based scoring function [Wang et al., 2017a] is proposed to decide which vertex should be added or removed, which can further improve the performance of the local search phase. The main idea of this new scoring function is as follow: Each vertex $v \in V$ has an additional property: frequency, denoted by $freq[v]$. In the beginning, the $freq$ of each vertex is set to 1. Then, at the end of local search, the $freq$ of each non-dominated vertex is increased by one. We use $\text{score}_f$ to denote the frequency based scoring function: If $u \in D$, $\text{score}_f(u) = -\sum_{v \in C_1} freq[v]/w(u)$ where $C_1$ is the set of dominated vertices that would become non-dominated by removing $u$ from $D$; otherwise, if $u \notin D$, then $\text{score}_f(u) = \sum_{v \in C_2} freq[v]/w(u)$ where $C_2$ is the set of non-dominated vertices that would become dominated by adding $u$ into $D$. 

1515
3 A New Construction with Reduction Rules

In this section, we propose a new fast construction procedure ConstructDS, which includes three parts: reducing, constructing, shrinking. Firstly, we introduce four rules to reduce the size of the MWDS problem and then show how to construct a dominating set quickly. Finally, redundant vertices will be removed. The resulting solution will serve as an initial candidate solution for subsequent local search.

3.1 Reduction Rules

Below are four reduction rules used in ConstructDS.

Weighted-Degree-0 Rule. An isolated vertex \( u \) with \( \text{deg}(u) = 0 \) must be in an optimal solution. Thus, \( u \) is fixed as a dominating vertex in \( D \).

Weighted-Degree-1 Rule 1. If \( G \) includes \( v \) s.t. \( N(v) = \{ u \} \) and \( w(v) > w(u) \), then \( u \) is fixed as a dominating vertex in \( D \) and \( G \) is simplified by deleting \( v \) and its incident edges.

Weighted-Degree-1 Rule 2. If \( G \) contains \( v_1, v_2, \ldots, v_t \) s.t. \( N(v_1) = N(v_2) = \cdots = N(v_t) = \{ u \} \) and \( w(v_1) + w(v_2) + \cdots + w(v_t) > w(u) \), then \( u \) is fixed as a dominating vertex in \( D \) and \( G \) is simplified by deleting \( v_1, v_2, \ldots, v_t \) and their incident edges.

Weighted-Degree-2 Rule. If \( G \) includes \( v_1, v_2, u \) s.t. \( N(v_1) = \{ v_2, u \} \), \( N(v_2) = \{ v_1, u \} \), \( w(v_1) > w(u) \) and \( w(v_2) > w(u) \), then \( u \) is fixed as a dominating vertex in \( D \) and \( G \) is simplified by deleting \( v_1, v_2 \) and their incident edges.

Although these weighted reduction rules are inspired by the dominance rule [Fomin et al., 2009], these rules for the MWDS problem contain additionally a weighted constraint for handling the weights, and thus extend the original reduction rules for unweighted graphs. Moreover, these rules have never been applied into heuristic MWDS algorithms.

3.2 The ConstructDS Procedure

The ConstructDS procedure is presented in Algorithm 1. ConstructDS consists of three parts: reducing (Line 1-5), constructing (Line 6-9), and shrinking (Line 10-11).

In the reducing part, we apply four reduction rules into \( G \). If these exist some vertices which must be in an optimal solution, then these vertices are fixed as dominating vertices and we add these vertices into the candidate solution. In the subsequent local search phase, these fixed vertices are forbidden to be removed from the candidate solution.

In the constructing part, during each step, ConstructDS randomly picks a non-dominated vertex \( v \). Afterwards, among all closed neighborhood of \( v \), the vertex \( u \) with the biggest score value is picked. Finally, we put \( u \) into the candidate solution \( D \).

In the shrinking part, ConstructDS will remove one of the vertices whose score value is 0 at each step. Assuming that when removing vertex \( v \) with score \( f(v) = 0 \), for \( \forall u \in N^2(v) \), score \( f(u) \) will be updated and then for the previous vertices whose score \( f(u) = 0 \), these score \( f(u) \) values may be changed. Therefore, ConstructDS removes only one redundant vertex at each step.

Most massive graphs with power-law degree distribution [Eubank et al., 2004] can be reduced considerably by using some strategies. In this case, removing some redundant vertices by our reduction rules will benefit the subsequent construction procedure. Also, we try to reduce the complexity of the constructing and shrinking parts since the remaining reduction massive graphs still have vertices with the large size. In our algorithm, the complexity of the ConstructDS procedure is \( O(|V| \cdot N_{\text{max}} + |D| \cdot N_{\text{max}} + |D|) = O(|V| \cdot N_{\text{max}}) \), where \( N_{\text{max}} \) is the maximum number of \( \text{deg}(v) \), for \( \forall v \in V \).

In our experiments, after the reducing part, an initial solution has already dominated on average 59.07% vertices of all massive graphs. Also, the run time of ConstructDS on all massive graphs but 9 is less than 10 seconds.

4 A New Configuration Checking Strategy

The CC strategy has been used successfully in local search algorithms for many NP-hard combinatorial problems, and several variants of CC have been proposed. Among these, the two-level configuration checking (CC\(^2\)) has been shown to be promising for the MWDS problem, with experiment results supporting its superiority over other CC strategies.

4.1 Review of CC\(^2\) Strategy

The two-level configuration checking (CC\(^2\)) was proposed to improve a local search algorithm for MWDS [Wang et al., 2017a]. CC\(^2\) is implemented with a Boolean array \( \text{ConfChange} \) whose size equals the number of vertices in the given graph. In CC\(^2\), the configuration of a vertex is defined to be a vector consists of its neighbors and its neighbors’ neighbors. The vertex is considered configuration-changed, if the value of any bit of the vector has changed. The CC\(^2\) strategy forbids any vertex to be added into the candidate solution if it is not configuration-changed since its last removal from the candidate solution.

The CC\(^2\) strategy works as follows.

Updating rules: (1) At the beginning of local search, for each vertex \( v \), \( \text{ConfChange}[v] \) is set to 1; (2) When
vertex $v$ is removed, ConfChange[$v$] is reset to 0, and ConfChange[$u$] is set to 1 for all $u \in N^2(v)$; (3) When vertex $v$ is added, ConfChange[$u$] is set to 1 for all $u \in N^2(v)$.

Using rule: When choosing an added vertex $v$, CC2 forbids any vertex to be added into the candidate solution if its configuration has not been changed, i.e., ConfChange[$v$] = 0.

4.2 Intuition and Data Structure of CC2V3

For the MWDS problem, local search algorithms usually modify the candidate solution by adding or removing a vertex. Although CC2 leads to more candidate added vertices than the original configuration checking, it does not distinguish the vertices that are allowed to be added. That is, any vertex only has two states, either forbidden (0) or allowed (1).

In fact, for such CC strategy with two levels, we can further exploit the different levels to distinguish the priority of the allowed vertices. Based on this consideration, we propose a new version of configuration checking, that is, a three-valued two-level configuration checking denoted by CC2V3.

To implement CC2V3, we employ an integer array $conf$, whose size equals the number of vertices in the input graph. For each vertex $v$, the $conf[v]$ value has three possibilities, and their meanings are explained below.

- $conf[v] = 0$ means that vertex $v$ should be forbidden to be added;
- $conf[v] = 1$ means that $v$ is clearly dominated by the current selected vertex.
- $conf[v] = 2$ means that the dominated state of $v$ is not changed by the current selected vertex or is possibly become to “non-dominated” by the current selected vertex.

4.3 Updating Rules of CC2V3

In this subsection, we will explain the different configuration values of each vertex for the removing process and adding process of local search as below.

At the start of local search, since an initial solution is already a dominating set, each vertex in the given graph must be dominated by some vertices of the candidate solution. Thus, $conf[v]$ should be initialized to 1, for $\forall v \in V$.

When adding a vertex $v$ into the candidate solution, for $\forall u_1 \in N_1(v)$, $u_1$ is dominated by vertex $v$ and $conf[u_1]$ is set to 1. But, the added vertex $v$ cannot change the dominated or non-dominated state of $u_2$, for $\forall u_2 \in N_2(v)$. Thus, we mark $conf[u_2]$ as 2.

When removing a vertex $v$ from the candidate solution, no matter whether $u_1 \in N_1(v)$ or $u_2 \in N_2(v)$, they both have the chance to be still dominated by the candidate solution, since other vertices in the candidate solution maybe dominate them. Therefore, the $conf[u_1]$ and $conf[u_2]$ need to be set to 0. For the removal vertex $v$, we should forbid this vertex to be added back to the candidate solution until its configuration has been changed. For this reason, we set $conf[v]$ to 0.

For these considerations, we formally give the configuration updating rules as below.

CC2V3 Rule1. In the beginning, for $\forall v \in V$, $conf[v]$ is initialized to 1.

**CC2V3 Rule2.** When vertex $v$ is added into candidate solution $D$, for $\forall u_1 \in N_1(v)$, $conf[u_1]$ is set to 1, and for $\forall u_2 \in N_2(v)$, $conf[u_2]$ is set to 2.

**CC2V3 Rule3.** When vertex $v$ is removed from candidate solution $D$, for $\forall u \in (N_1(v) \cup N_2(v))$, $conf[u]$ is set to 2 and $conf[v]$ should be reset to 0.

4.4 Using Rule of CC2V3

We apply the CC2V3 strategy into the added vertex selection procedure. The resulting adding rule is described as below.

**Adding Rule.** When adding one vertex into the candidate solution $D$, select a vertex $v$ with the biggest score$_f(v)$ value and $conf[v] \neq 0$, breaking ties by picking one vertex with the highest $conf[v]$ value.

When selecting an added vertex $v$, if there is more than one vertex with the same biggest score value, then we have to select one vertex among these vertices. Although these vertices have the same biggest values, the configuration values of these vertices may be different. Assuming that both of vertex $u$ with $conf[u] = 2$ and vertex $v$ with $conf[v] = 1$ have the same biggest score value. Compared with vertex $v$ that has already dominated by the candidate solution, we prefer to choose vertex $u$ with the uncertain dominated state, which may decrease the number of non-dominated vertices as much as possible. Further ties are broken randomly if more than one vertex has the biggest score value and the highest configuration value.

In fact, the scoring function seen as the global view strategy selects an added vertex, while the CC2V3 strategy mainly reflects the relevant information of the local added vertex. Therefore, the reasonable selection vertex strategy should take both of them into account.

5 The FastMWDS Algorithm

In this section, we propose a local search algorithm for the MWDS problem named FastMWDS, which is outlined in Algorithm 2. FastMWDS can be divided into two parts: construction (Line 3) and local search (Line 5-22). After constructing an initial candidate solution, the algorithm works iteratively by removing some vertices and adding some other vertices until time limit is reached. At last, the best found solution is returned (Line 23).

At the beginning of each step in local search phase, a current candidate solution $D$ is already a weighted dominating set. If there exist some vertices whose score$_f$ value is 0, then among these vertices the algorithm randomly removes a vertex $u$ from $D$ (Line 6-9), which means that the candidate solution is still a weighted dominating set after removing this vertex. Otherwise, the algorithm turns to check whether the current candidate solution $D$ is better than the best solution $D^*$, and if so, $D^*$ is updated by $D$ (Line 10). Afterwards, the algorithm selects two removal vertices, according to two different situations.

The first situation (Line 11-13): the current candidate solution $D$ remains a weighted dominating set. In this case, the algorithm tries to find the best move to update $D$. When some vertices have the same highest score value, the algorithm selects the oldest vertex $u_1$ to remove. The favour towards the oldest vertex introduces diversification into this greedy move.
The second situation (Line 14-16): the current candidate solution $D$ is not a weighted dominating set. In this case, we consider the algorithm should do some perturbations. On the other hand, pure randomized strategies would lose the information during the search. Also, after a long period, the algorithm still cannot find a better solution, and thus the strong random walk should be brought into the algorithm. Based on the above considerations, we propose a new variant of “best from multiple selection” (BMS) heuristic [Cai et al., 2017]. We formalize the BMS heuristic with some random walks in Algorithm 3 as below. In our BMS heuristic, we use non-impr-step to denote the number of non-improvement steps, which is initialized as 0 (Line 2). It is increased by one after each step (Line 22), and is reset to 0 when obtaining a better weighted dominating set (Line 10). The parameter $k$ is used to control the greediness and a large $k$ value usually indicates a great greediness and more computation time. Inspired by Metropolis algorithm [Bertsimas et al., 1993] used in Simulate Anneal, we allow the algorithm to accept $k = 1024$ with the probability of $e^{-\text{step}}$, otherwise $k = 50 + \text{rand}(10)$ with the probability of $1 - e^{-\text{step}}$. At first, the algorithm dedicates to accurately finding a vertex with the highest score value, rather than a random vertex in $D$. Therefore, the algorithm has a high probability to assign $k$ as 1024, and has a low probability to assign $k$ as 50 + rand(10). After many steps, more random walks will be added into the algorithm, i.e. having a high probability to assign $k$ as 50 + rand(10), leading to explore many different spaces. After updating the value of $k$, the algorithm randomly picks $k$ vertices and among these vertices the best move will be selected.

After removing two vertices from $D$, the algorithm adds some vertices into $D$ until $D$ becomes a dominating set (Line 17-21). The algorithm chooses the added vertices from $N^2(u_1) \cup N^2(u_2)$ rather than $V - D$, and thus the number of the candidate added vertices is very small. During the adding procedure, the algorithm prefers to pick a vertex $v$ with the biggest score value and the highest configuration value.

For each step in the local search phase (Line 5-22), the complexity is $O(\max\{|D|, 2 \times N^{3_{max}}\})$, where $|D|$ is the size of $D$ and $N_{max}$ is the maximum number of $\text{deg}(v)$, for all $v \in V$.

6 Experimental Results

In this section, we carry out extensive experiments to test the performance of FastMWDS on a broad range of massive real world graphs, compared against a state of the art local search algorithm for MWDS, as well as three state of the art WPMS solvers because MWDS can be easily translated into WPMS. The WPMS solvers include MaxHS [Davies, 2013; Bacchus, 2017], OpenWBO [Martins et al., 2014; 2017], and WPM3-2015-in [Ansotegui et al., 2013; 2015], where MaxHS and OpenWBO are the complete versions submitted to the MaxSAT Evaluations 2017 and WPM3-2015-in as a recently incomplete algorithm is submitted to the MaxSAT Evaluations 2016. We consider 187 massive real world graphs from the Network Data Repository [Rossi and Ahmed, 2015]. There are a few bipartite graphs in the benchmark, and we choose to ignore them. For the sake of space, we do not report the results on graphs with less than 110,000 vertices and less than 1,000,000 edges in which the performance of our algorithm is always best. In total, 63 instances are reported in this section. To obtain the corresponding MWDS instances, we use the same method as in [Wang et al., 2017a], i.e., for the $i$th vertex $v_i$, $w(v_i) = (i \mod 200) + 1$.

FastMWDS is implemented in C++ and compiled using GNU g++ -O3. All experiments are performed on Ubuntu Linux, with 3.1 GHz CPU and 16GB memory. For both of them, 10 independent runs with different seeds are performed for each instance. The time limit of all algorithms is 1000 seconds. For each instance, $w_{\text{min}}$ is the weight of the best

---

**Algorithm 2: the FastMWDS algorithm**

**Input**: a weighted graph $G = (V, E, w)$, the cutoff time

**Output**: a weighted dominating set of $G$

1. initialize $conf$ according to the CC2 V3 Rule 1;
2. non-impr-step := 0;
3. $D := \text{ConstructDS}(G)$;
4. $D^* := D$;
5. while elapsed time < cutoff do
   6. if there exists vertex $u$ with $\text{score}_f(u) = 0$ then
      7. select a random vertex $u$ with $\text{score}_f(u) = 0$;
      8. $D := D \setminus \{u\}$;
      9. continue;
   10. if $w(D) < w(D^*)$ then $D^* := D$ and non-impr-step := 0;
   11. select vertex $u_1 \in D$ with the biggest $\text{score}_f$ value, breaking ties by the oldest one;
   12. $D := D \setminus \{u_1\}$;
   13. update $conf$ according to the CC2 V3 Rule 3;
   14. select vertex $u_2 \in D$ by BMS(non-impr-step);
   15. $D := D \setminus \{u_2\}$;
   16. update $conf$ according to the CC2 V3 Rule 3;
   17. repeat
      18. select a vertex $v \in (N^2(u_1) \cup N^2(u_2))$ according to the Adding Rule;
      19. $D := D \cup \{v\}$;
      20. update $conf$ according to CC2 V3 Rule 2;
   21. until $D$ is a weighted dominating set;
   22. non-impr-step := non-impr-step+1;
23. return $D^*$;

---

**Algorithm 3: the BMS heuristic with some random walks**

**Input**: the number of non-improvement step

**Output**: a removal vertex $u$;

1. if with probability $q < e^{-\text{step}}$ then
2. $k := 1024$;
3. else
4. $k := 50 + \text{rand}(10)$;
5. pick a random vertex $u_2 \in D$;
6. $\text{score}_f^* := \text{score}_f(u_2)$ and $u^* := u_2$;
7. for $i = 0; i < k; i++$ do
8. pick a random vertex $u_2 \in D$;
9. if $\text{score}_f(u_2) > \text{score}_f^*$ then
10. $\text{score}_f^* := \text{score}_f(u_2)$ and $u^* := u_2$;
11. return $u^*$;
<table>
<thead>
<tr>
<th>Instance</th>
<th>MaxHS wmin</th>
<th>OpenWBO wmin</th>
<th>WPM3-2015-in wmin</th>
<th>CC²FS wmin</th>
<th>FastMWDS wmin</th>
<th>FastMWDS time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bn-human-BNU</td>
<td>119567023</td>
<td>119829952</td>
<td>119864215</td>
<td>N/A</td>
<td>N/A</td>
<td>119322667</td>
</tr>
<tr>
<td>25865_session,1-bg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bn-human-BNU</td>
<td>155868115</td>
<td>156089311</td>
<td>156101609</td>
<td>N/A</td>
<td>N/A</td>
<td>155703082</td>
</tr>
<tr>
<td>25865_session,2-bg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ca-coauthors-dblp</td>
<td>3203654</td>
<td>2621360</td>
<td>4010227</td>
<td>2584027</td>
<td>991.65</td>
<td>2539457</td>
</tr>
<tr>
<td>ca-dblp-2012</td>
<td>4048614</td>
<td>3898186</td>
<td>4756639</td>
<td>3757029</td>
<td>999.62</td>
<td>3681795</td>
</tr>
<tr>
<td>ca-hollywood-2009</td>
<td>5507112</td>
<td>4368759</td>
<td>6986234</td>
<td>3723085</td>
<td>380.06</td>
<td>3654205</td>
</tr>
<tr>
<td>channel-500x</td>
<td>5243638</td>
<td>4563957</td>
<td>52592043</td>
<td>N/A</td>
<td>N/A</td>
<td>27826097</td>
</tr>
<tr>
<td>100x100-b050</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dbpedia-link</td>
<td>154591900</td>
<td>154526642</td>
<td>154526642</td>
<td>N/A</td>
<td>N/A</td>
<td>133739180</td>
</tr>
<tr>
<td>delaunay_n22</td>
<td>8303582</td>
<td>83774141</td>
<td>88465848</td>
<td>N/A</td>
<td>N/A</td>
<td>55243217</td>
</tr>
<tr>
<td>delaunay_n23</td>
<td>165834069</td>
<td>171803113</td>
<td>176359228</td>
<td>N/A</td>
<td>N/A</td>
<td>112013949</td>
</tr>
<tr>
<td>delaunay_n24</td>
<td>331967826</td>
<td>345763164</td>
<td>354066023</td>
<td>N/A</td>
<td>N/A</td>
<td>226251102</td>
</tr>
<tr>
<td>friendster</td>
<td>66264132</td>
<td>66275673</td>
<td>66275673</td>
<td>N/A</td>
<td>N/A</td>
<td>55506341</td>
</tr>
<tr>
<td>hugebubbles-00020</td>
<td>708625717</td>
<td>747989696</td>
<td>755304622</td>
<td>N/A</td>
<td>N/A</td>
<td>516458602</td>
</tr>
<tr>
<td>hugetrace-00010</td>
<td>406897159</td>
<td>444315493</td>
<td>450388589</td>
<td>N/A</td>
<td>N/A</td>
<td>307261619</td>
</tr>
<tr>
<td>hugetrace-00020</td>
<td>535479596</td>
<td>573230746</td>
<td>579039050</td>
<td>N/A</td>
<td>N/A</td>
<td>400022798</td>
</tr>
<tr>
<td>inf-europe_osm</td>
<td>N/A</td>
<td>N/A</td>
<td>39549008</td>
<td>N/A</td>
<td>N/A</td>
<td>1820445787</td>
</tr>
<tr>
<td>inf-germany_osm</td>
<td>409829603</td>
<td>400775441</td>
<td>400775441</td>
<td>N/A</td>
<td>N/A</td>
<td>403783914</td>
</tr>
<tr>
<td>inf-roadNet-CA</td>
<td>64889764</td>
<td>63953632</td>
<td>63769717</td>
<td>N/A</td>
<td>N/A</td>
<td>28919193</td>
</tr>
<tr>
<td>inf-roadNet-PA</td>
<td>36000505</td>
<td>34776092</td>
<td>36446447</td>
<td>N/A</td>
<td>N/A</td>
<td>646087601</td>
</tr>
<tr>
<td>inf-road-usa</td>
<td>852776450</td>
<td>899703836</td>
<td>890326743</td>
<td>N/A</td>
<td>N/A</td>
<td>516458602</td>
</tr>
<tr>
<td>rec-dating</td>
<td>913220</td>
<td>875523</td>
<td>909326</td>
<td>853349</td>
<td>996.64</td>
<td>928802</td>
</tr>
<tr>
<td>rec-epinions</td>
<td>815003</td>
<td>756642*</td>
<td>982297</td>
<td>760187</td>
<td>997.83</td>
<td>757593</td>
</tr>
<tr>
<td>rec-libimeseti-dir</td>
<td>1012517</td>
<td>956296</td>
<td>1345525</td>
<td>932442</td>
<td>996.46</td>
<td>928802</td>
</tr>
<tr>
<td>rgg_n2,23,30</td>
<td>72025903</td>
<td>41746129</td>
<td>72232302</td>
<td>N/A</td>
<td>N/A</td>
<td>19032060</td>
</tr>
<tr>
<td>rgg_n2,24,30</td>
<td>139001281</td>
<td>92187644</td>
<td>139417747</td>
<td>N/A</td>
<td>N/A</td>
<td>35956171</td>
</tr>
<tr>
<td>Rt-retweet-crawl</td>
<td>7642932</td>
<td>9241645</td>
<td>9241645</td>
<td>7337141</td>
<td>998.98</td>
<td>7124971</td>
</tr>
</tbody>
</table>

Table 1: Experiment results of MaxHS, OpenWBO, WPM3-2015-in, CC²FS and FastMWDS on massive graphs I.

Experiment results on the massive graphs are shown in Tables 1 and 2. CC²FS under the time limit (1000 seconds) are essentially worse than FastMWDS under the same time limit. Especially, there are 37 graphs for which CC²FS cannot find weighted dominating sets within the time limit (i.e., failing to finish its initialization phase within the time limit), while FastMWDS finds weighted dominating sets for all graphs.

The WPMS algorithms MaxHS, OpenWBO and WPM3-
In this paper, we develop a local search algorithm for the MWDS problem called FastMWDS, which can solve massive instances in a reasonable amount of time. For 41 massive instances, the percentage of the vertices dominated by the candidate solution in FastMWDS is larger than 100%. Figure 1 shows the initialization run time of FastMWDS for these instances, except for one instance soc-ljournal-2008. Experimental results in Table 3 show that FastMWDS finds better solutions than FastMWDS+CC FS. For 9 instances, the initialization run time of FastMWDS is larger than 10 seconds. Figure 2 shows the percentage of the vertices dominated by the candidate solution after applying four reduction rules into the FastMWDS algorithm. For 41 massive instances, the percentage of the dominated vertices is more than 50%. Figure 1 and Figure 2 intuitively display the effectiveness of our construction procedure.

6.2 The Effectiveness of the CC2V3 Strategy

In this subsection, to further study the effectiveness of the CC2V3 strategy, we compare FastMWDS with its alternative version named FastMWDS+CC2, which uses the CC2 strategy instead of our CC2V3 strategy. We select 26 instances where CC2 fails to obtain the solutions within the time limit (3600 seconds). Table 3 shows that FastMWDS finds better solutions than FastMWDS+CC2 on these instances, except for one instance soc-journal-2008. Experimental results indicate that the CC2V3 strategy makes an important role in the FastMWDS algorithm.

7 Conclusion and Future Work

In this paper, we develop a local search algorithm for the MWDS problem called FastMWDS, which can solve massive instances in a reasonable amount of time.
sive graphs within a short time. Two new ideas are used to improve FastMWDS. Firstly, a new fast construction procedure with four reduction rules is proposed, which includes three parts: reducing, constructing, shrinking. After this construction procedure, the size of massive graphs is reduced. Secondly, we design a new configuration checking with multiple values named CC. V3, which can exploit the relevant information of the current solution and is better than previous CC strategies. Experiments on massive graphs show that FastMWDS finds better solutions than the state of the art algorithms. Also, FastMWDS obtains a good weighted dominating set for all massive graphs within a short time limit (1000 seconds), while CC^2FS and three WPMS algorithms fail to find a weighted dominating set on a considerable portion of massive graphs. As for future work, we will try to further improve our FastMWDS algorithm by some other ideas, and try to solve other massive instances.

Acknowledgments

This work is supported by the Fundamental Research Fund of the Central Universities 2412018QD022. NSFC (under Grant Nos. 61370156, 61520246, 61503074), China National 973 Program 2014CB340301. Shaowei Cai is also supported by Youth Innovation Promotion Association, Chinese Academy of Sciences.

References


Table 3: Experiment results of FastMWDS and FastMWDS+CC^2 on massive graphs. A positive δ_min indicates FastMWDS finds better quality dominating set than FastMWDS+CC^2.


