Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

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

Yiyuan Wang[1,3], **Shaowei Cai**[2*], **Jiejiang Chen**[1,3] **and Minghao Yin**[1,3*]
[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
[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.

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=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+ε)-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

*Corresponding author

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_o to solve MWDS. According to the literature, the current best heuristic algorithm for the MWDS problem is called CC^2FS, 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, CC^2FS 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 ($\text{CC}^2\text{V3}$) 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 (CC^2) [Wang *et al.*, 2017a] was designed to solve the MWDS problem. Our $\text{CC}^2\text{V3}$ strategy can be viewed as a multiple-value version of CC^2. While CC^2 only considers whether a vertex is configuration changed or not, $\text{CC}^2\text{V3}$ 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]. Experiment 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 $\text{CC}^2\text{V3}$. 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. $N(v) = \{u \in V|\{u, v\} \in E\}$ is the set of neighbors of a vertex v. The *degree* of vertex v is defined as $deg(v) = |N(v)|$. $dist(u, v)$ 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 ith level neighborhood as $N_i(v) = \{u|dist(u, v) = i\}$, 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 $score_f$ to denote the frequency based scoring function: If $u \in D$, $score_f(u) = -\sum_{v \in C_1} freq[v]/w(u)$ 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 $score_f(u) = \sum_{v \in C_2} freq[v]/w(u)$ where C_2 is the set of non-dominated vertices that would become dominated by adding u into D.

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 $deg(u) = 0$ must be in an optimal solution. Thus, u is fixed as a dominating vertex in D.

Weighted-Degree-1 Rule1. 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 Rule2. If G contains $v_1, v_2, \dots v_t$ 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 $v_1, v_2, \dots v_t$ 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 three parts: reducing (Line 1-5), constructing (Line 6-9), and shrinking (Line 10-11).

Algorithm 1: the ConstructDS procedure

Input: a weighted graph $G = (V, E, w)$
Output: an initial weighted dominating set D
1 **while** *any reduction rules are satisfied* **do**
2   repeatedly apply **Weighted-Degree-0 Rule** into G until it is not satisfied;
3   repeatedly apply **Weighted-Degree-1 Rule1** into G until it is not satisfied;
4   repeatedly apply **Weighted-Degree-1 Rule2** into G until it is not satisfied;
5   repeatedly apply **Weighted-Degree-2 Rule** into G until it is not satisfied;
6 **while** *there exist non-dominated vertices* **do**
7   randomly select a vertex v from all non-dominated vertices;
8   select a vertex $u \in N[v]$ with the biggest $score_f$ value;
9     $D := D \cup \{u\}$;
10 **while** *there exists vertex v with $score_f(v) = 0$* **do**
11     $D := D \setminus \{v\}$;
12 **return** D;

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 $score_f$ value is picked. Finally, we put u into the candidate solution D.

In the shrinking part, ConstructDS will remove one of the vertices whose $score_f$ 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 $score_f$ value is 0, these $score_f$ 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 $O(|V| * N_{max} + |D| * N_{max} + |D|) = O(|V| * N_{max})$, where N_{max} is the maximum number of $deg(v)$, 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 $ConfChange$ 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, $ConfChange[v]$ 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, CC^2 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 CC^{2}V3

For the MWDS problem, local search algorithms usually modify the candidate solution by adding or removing a vertex. Although CC^2 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 CC^{2}V3.

To implement CC^{2}V3, 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 CC^{2}V3

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 2. 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.

CC^{2}V3 Rule1. In the beginning, for $\forall v \in V$, $conf[v]$ is initialized to 1.

CC^{2}V3 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.

CC^{2}V3 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 CC^{2}V3

We apply the CC^{2}V3 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 CC^{2}V3 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 step, 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^{-step}, otherwise $k = 50 + rand(10)$ with the probability of 1-e^{-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 $O(max\{|D|, 2 * N_{max}^3\})$, where $|D|$ is the size of D and N_{max} is the maximum number of $deg(v)$, for $\forall v \in V$.

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 CC^{2}V3 Rule1;
2 $non\text{-}impr\text{-}step := 0$;
3 $D := ConstructDS(G)$;
4 $D^* := D$;
5 **while** *elapsed time* $<$ *cutoff* **do**
6   **if** *there exists vertex* u *with* $score_f(u) = 0$ **then**
7   select a random vertex u with $score_f(u) = 0$;
8     $D := D \setminus \{u\}$;
9   continue;
10   **if** $w(D) < w(D^*)$ **then** $D^* := D$ and $non\text{-}impr\text{-}step := 0$;
11   select vertex $u_1 \in D$ with the biggest $score_f$ value, breaking ties by the oldest one;
12   $D := D \setminus \{u_1\}$;
13   update $conf$ according to the CC^{2}V3 Rule3;
14   select vertex $u_2 \in D$ by BMS(*non-impr-step*);
15   $D := D \setminus \{u_2\}$;
16   update $conf$ according to the CC^{2}V3 Rule3;
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 CC^{2}V3 Rule2;
21   **until** D *is a weighted dominating set*;
22   $non\text{-}impr\text{-}step := non\text{-}impr\text{-}step$+1;
23 **return** D^*;

Algorithm 3: the BMS heuristic with some random walks

Input: the number of non-improvement step $step$
Output: a removal vertex u^*
if *with probability* $q < e^{-step}$ **then**
  $k := 1024$;
else
  $k :=$ 50+rand(10);
pick a random vertex $u_2 \in D$;
$score_f^* := score_f(u_2)$ and $u^* := u_2$;
for $i = 0; i < k; i$++ **do**
  pick a random vertex $u_2 \in D$;
  if $(score_f(u_2) > score_f^*)$ **then**
    $score_f^* := score_f(u_2)$ and $u^* := u_2$;
return u^*;

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 [Ansótegui *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 ith 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, $wmin$ is the weight of the best

Instance	MaxHS	OpenWBO	WPM3-2015-in	CC^2FS		FastMWDS	
	$wmin$	$wmin$	$wmin$	$wmin$	$time$	$wmin$	$time$
bn-human-BNU_1_0025865_session_1-bg	119567023	119829952	119864215	N/A	N/A	**119322667**	997.15
bn-human-BNU_1_0025865_session_2-bg	155868115	156089311	156101609	N/A	N/A	**155703082**	764.47
ca-coauthors-dblp	3203654	2621360	4010227	2584027	991.65	**2539457**	971.04
ca-dblp-2012	4048614	3898186	4756639	3757029	999.62	**3681795**	933.59
ca-hollywood-2009	5507112	4368759	6986234	3732085	380.06	**3654205**	998.27
channel-500x100x100-b050	52436338	45639579	52592043	N/A	N/A	**27826097**	918.87
dbpedia-link	154591900	154526642	154526642	N/A	N/A	**133739180**	5.97
delaunay_n22	83035082	83774141	88465848	N/A	N/A	**55243217**	919.21
delaunay_n23	165834069	171803113	176359228	N/A	N/A	**112013949**	989.98
delaunay_n24	331967826	345763164	354066023	N/A	N/A	**226251102**	990.91
friendster	66264132	66275673	66275673	N/A	N/A	**55506341**	18.94
hugebubbles-00020	708625717	747989696	755304622	N/A	N/A	**516458602**	999.87
hugetrace-00010	406897159	444315493	450388589	N/A	N/A	**307261619**	929.92
hugetrace-00020	535479596	573230746	579039050	N/A	N/A	**400022798**	946.93
inf-europe_osm	N/A	N/A	N/A	N/A	N/A	**1820445787**	914.77
inf-germany_osm	409829603	**399549008**	400775441	N/A	N/A	403783914	993.99
inf-roadNet-CA	64889764	63935632	65769717	N/A	N/A	**53077217**	997.59
inf-roadNet-PA	36000505	34776092	36446447	N/A	N/A	**28919193**	990.95
inf-road-usa	852776450	899703836	903206743	N/A	N/A	**646087601**	978.8
rec-dating	913220	875523	909326	853349	996.64	**851783**	824.91
rec-epinions	815003	**756642***	982297	760187	997.83	757593	804.92
rec-libimeseti-dir	1012517	956296	1345525	932442	996.46	**928802**	964.5
rgg_n_2_23_s0	72025903	41476129	72223302	N/A	N/A	**19032060**	969.93
rgg_n_2_24_s0	139001281	92187644	139417747	N/A	N/A	**35956171**	987.98
Rt-retweet-crawl	7642932	9241645	9241645	7337141	998.98	**7124971**	776.26

Table 1: Experiment results of MaxHS, OpenWBO, WPM3-2015-in, CC^2FS and FastMWDS on massive graphs I.

dominating set found, and $time$ is the average run time when algorithms obtain the minimal solution values. The bold value indicates the best value between FastMWDS and its competitors. For some instances, four competitors fail to obtain a weighted dominating set within the given time limit, then we use "N/A" to mark it. If an algorithm proves the optimal solution, the corresponding column is marked with *.

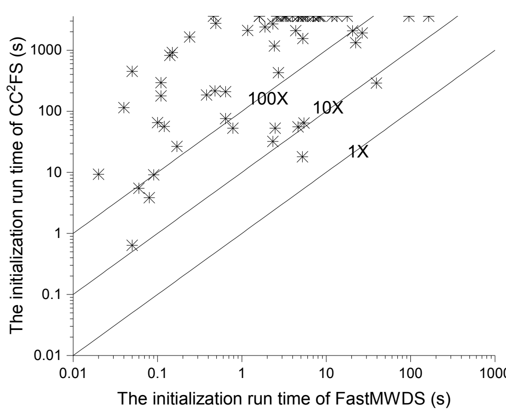

Figure 1: The initialization run time of FastMWDS and CC^2FS

Experiment results on the massive graphs are shown in Tables 1 and 2. CC^2FS under the time limit (1000 seconds) are essentially worse than FastMWDS under the same time limit. Especially, there are 37 graphs for which CC^2FS 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.

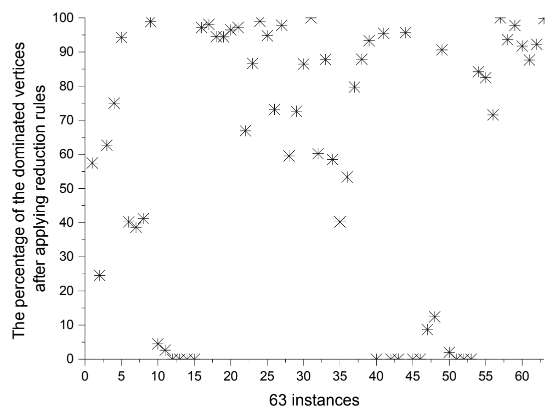

Figure 2: The percentage of the dominated vertices after applying reduction rules

The WPMS algorithms MaxHS, OpenWBO and WPM3-

Instance	MaxHS	OpenWBO	WPM3-2015-in	CC^{2}FS		FastMWDS	
	wmin	*wmin*	*wmin*	*wmin*	*time*	*wmin*	*time*
sc-ldoor	6305998	5799129	6777213	5459575	922.24	**5442220**	980.49
sc-msdoor	1991510	1639855	2218936	1578611	972.83	**1562389**	993.82
sc-pwtk	415218	311361	450291	278128	963.3	**274527**	914.19
sc-rel9	17250188	16591326	16746750	N/A	N/A	**11912921**	993.71
sc-shipsec1	975929	664599	1019460	389617	776.63	**380343**	742.31
sc-shipsec5	1354538	810243	1430233	516034	935.94	**503750**	783.93
soc-delicious	5632727	5647129	5647129	4929139	998.69	**4896072**	844.12
soc-digg	6650265	6758798	6758798	5500630	995.02	**5439126**	866.09
soc-dogster	2081739	1894341	2725541	1909172	995.17	**1893048**	987.54
soc-flickr	9889280	9937487	9937487	7945690	237.05	**7874744**	862.02
soc-flickr-und	29827231	29831830	29831830	N/A	N/A	**24592651**	963.27
soc-flixster	9142886	9142886	9142886	8743064	986.35	**8602570**	918.18
soc-FourSquare	6127205	6151890	6151890	5416484	594.87	**5390875**	986.52
soc-lastfm	6746555	6981300	6981300	6079423	329.57	**6058609**	913.03
soc-livejournal	81529479	79826229	81818399	N/A	N/A	**60672724**	106.55
soc-livejournal -user-groups	108356093	108211843	108211843	N/A	N/A	**86247425**	209.41
soc-ljournal-2008	102894556	122980315	124862057	N/A	N/A	**92171228**	201.53
soc-orkut	13759507	13767087	13767087	N/A	N/A	**7164654**	998.03
soc-orkut-dir	11040456	10934518	10934518	N/A	N/A	**6374315**	996.4
soc-pokec	22716203	20864032	22779370	N/A	N/A	**14731523**	993.91
soc-sinaweibo	N/A	N/A	N/A	N/A	N/A	**20002246**	947.87
soc-twitter-higgs	1124102	927158	3123822	909896	997.5	**906236**	893.13
soc-youtube	9147519	8743841	9224143	6993952	206.26	**6905158**	970.03
soc-youtube-snap	21504191	21492126	21492126	17291372	980.68	**16760036**	988.61
socfb-A-anon	20440160	20452282	20452282	N/A	N/A	**17041237**	997.91
socfb-B-anon	18866563	18875391	18875391	N/A	N/A	**16103964**	981.02
socfb-uci-uni	N/A	N/A	N/A	N/A	N/A	**84054669**	982.69
tech-as-skitter	19822652	19771180	19822631	N/A	N/A	**13173037**	998.68
tech-ip	13975	3510	2808	2427	744.53	**2321**	13.5
twitter_mpi	57042991	57032003	57032003	N/A	N/A	**50521218**	10.47
web-arabic-2005	1637957	1625020	2114247	1580428	993.98	**1572856**	974.85
web-baidu-baike	28300817	28245789	28274603	N/A	N/A	**23785438**	998.04
web-it-2004	2747602	2619362	4653739	2621892	998.21	**2582786**	989.66
web-uk-2005	**93183***	**93183***	**93183***	**93183**	1.28	**93183**	0.53
web-wikipedia_link_it	34761238	36041177	37120690	N/A	N/A	**29162579**	155.11
web-wikipedia2009	35830702	37674803	37674803	N/A	N/A	**27120007**	967.74
web-wikipedia-growth	12498617	11404055	12509126	N/A	N/A	**8301996**	948.26
wikipedia_link_en	N/A	N/A	N/A	N/A	N/A	**2408988048**	982.71

Table 2: Experiment results of MaxHS, OpenWBO, WPM3-2015-in, CC^{2}FS and FastMWDS on massive graphs II.

2015-in fail on 4 graphs, and FastMWDS finds better solutions than MaxHS, OpenWBO and WPM3-2015-in on the remaining massive graphs, except two instances inf-germany_osm and rec-epinions.

6.1 The Effectiveness of ConstructDS

Figure 1 plots the initialization run time of CC^{2}FS versus FastMWDS, clearly showing the superiority of FastMWDS. For all massive graphs, the initialization run time of FastMWDS is always shorter than that of CC^{2}FS. Especially, for 37 graphs, the initialization of FastMWDS is 1000 $\times$ faster than CC^{2}FS. For only 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%. Figures 1 and 2 intuitively display the effectiveness of our construction procedure.

6.2 The Effectiveness of the CC^{2}V3 Strategy

In this subsection, to further study the effectiveness of the CC^{2}V3 strategy, we compare FastMWDS with its alternative version named FastMWDS+CC^2, which uses the CC^2 strategy instead of our CC^{2}V3 strategy. We select 26 instances where CC^{2}FS fails to obtain the solutions within the time limit (3600 seconds). Table 3 shows that FastMWDS finds better solutions than FastMWDS+CC^2 on these instances, except for one instance soc-ljournal-2008. Experimental results indicate that the CC^{2}V3 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 mas-

Instance	FastMWDS+CC^2 $wmin$	FastMWDS $wmin$	δ_{min}
bn-human-BNU_1_0025865_session_1-bg	119322869	119322667	**202**
bn-human-BNU_1_0025865_session_2-bg	155703116	155703082	**34**
channel-500x100x100-b050	28262180	27826097	**436083**
dbpedia-link	133741307	133739180	**2127**
delaunay_n22	55587000	55243217	**343783**
delaunay_n23	112278364	112013949	**264415**
delaunay_n24	226368604	226251102	**117502**
friendster	55513437	55506341	**7096**
hugebubbles-00020	516502746	516458602	**44144**
hugetrace-00010	307418821	307261619	**157202**
hugetrace-00020	400194448	400022798	**171650**
inf-europe_osm	1820450708	1820445787	**4921**
inf-germany_osm	403925678	403783914	**141764**
inf-roadNet-CA	53130369	53077217	**53152**
inf-road-usa	646193278	646087601	**105677**
rgg_n_2_23_s0	19086086	19032060	**54026**
rgg_n_2_24_s0	36012055	35956171	**55884**
sc-rel9	11985406	11912921	**72485**
socfb-uci-uni	84055494	84054669	**825**
soc-livejournal	60694863	60672724	**22139**
soc-livejournal-user-groups	86250344	86247425	**2919**
soc-ljournal-2008	**92169964**	92171228	-1264
soc-sinaweibo	20002259	20002246	**13**
twitter_mpi	50524864	50521218	**3646**
web-wikipedia_link_it	29164627	29162579	**2048**
wikipedia_link_en	2408988216	2408988048	**168**

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.

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^{2}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 reasonable time limit (1000 seconds), while CC^{2}FS 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 Funds for the Central Universities 2412018QD022, NSFC (under Grant Nos. 61370156, 61502464, 61503074), China National 973 Program 2014CB340301. Shaowei Cai is also supported by Youth Innovation Promotion Association, Chinese Academy of Sciences.

References

[Abramé *et al.*, 2017] André Abramé, Djamal Habet, and Donia Toumi. Improving configuration checking for satisfiable random k-sat instances. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):5–24, 2017.

[Ansótegui *et al.*, 2013] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[Ansótegui *et al.*, 2015] Carlos Ansótegui, Frederic Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in maxsat. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 283–289, 2015.

[Aoun *et al.*, 2006] Bassam Aoun, Raouf Boutaba, Youssef Iraqi, and Gary Kenward. Gateway placement optimization in wireless mesh networks with QoS constraints. *IEEE Journal on Selected Areas in Communications*, 24(11):2127–2136, 2006.

[Bacchus, 2017] Fahiem Bacchus. Maxhs v3.0 in the 2017 maxsat evaluation. *MaxSAT Evaluation 2017*, page 8, 2017.

[Bautista and Pereira, 2006] Joaquín Bautista and Jordi Pereira. Modeling the problem of locating collection areas for urban waste management. an application to the metropolitan area of barcelona. *Omega*, 34(6):617–629, 2006.

[Bertsimas *et al.*, 1993] Dimitris Bertsimas, John Tsitsiklis, et al. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[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.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

[Cai *et al.*, 2017] Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

[Caprara *et al.*, 1997] Alberto Caprara, Matteo Fischetti, Paolo Toth, Daniele Vigo, and Pier Luigi Guida. Algorithms for railway crew management. *Mathematical programming*, 79(1-3):125–141, 1997.

[Ceria *et al.*, 1998] Sebastián Ceria, Paolo Nobili, and Antonio Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, 1998.

[Chalupa, 2018] David Chalupa. An order-based algorithm for minimum dominating set with application in graph mining. *Information Sciences*, 426(1):101–116, 2018.

[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.

[Davies, 2013] Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, PhD thesis, University of Toronto, 2013.

[Eubank *et al.*, 2004] Stephen Eubank, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, and Nan Wang. Structural and algorithmic aspects of massive social networks. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 718–727, 2004.

[Fomin *et al.*, 2009] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5):25, 2009.

[Gao *et al.*, 2014] Chao Gao, Thomas Weise, and Jinlong Li. A weighting-based local search heuristic algorithm for the set covering problem. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 826–831, 2014.

[Gao *et al.*, 2017] Wanru Gao, Tobias Friedrich, Timo Kötzing, and Frank Neumann. Scaling up local search for minimum vertex cover in large graphs by parallel kernelization. In *Australasian Joint Conference on Artificial Intelligence*, pages 131–143, 2017.

[Hedetniemi *et al.*, 2003] Sandra M Hedetniemi, Stephen T Hedetniemi, David P Jacobs, and Pradip K Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Computers & Mathematics with Applications*, 46(5):805–811, 2003.

[Jiang *et al.*, 2017] Hua Jiang, Chu-Min Li, and Felip Manya. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 830–838, 2017.

[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, 2010.

[Lin *et al.*, 2016] Geng Lin, Wenxing Zhu, and Montaz M Ali. An effective hybrid memetic algorithm for the minimum weight dominating set problem. *IEEE Transactions on Evolutionary Computation*, 20(6):892–907, 2016.

[Lin *et al.*, 2017] Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 517–523, 2017.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.

[Martins *et al.*, 2014] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 438–445, 2014.

[Martins *et al.*, 2017] Ruben Martins, Miguel Terra-Neves, Saurabh Joshi, Mikoláš Janota, Vasco Manquinho, and Inês Lynce. Open-wbo in maxsat evaluation 2017. *MaxSAT Evaluation 2017*, page 17, 2017.

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

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

[Qian *et al.*, 2015] Chao Qian, Yang Yu, and Zhi-Hua Zhou. Subset selection by pareto optimization. In *Advances in Neural Information Processing Systems*, pages 1774–1782, 2015.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. 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, 2010.

[Subhadrabandhu *et al.*, 2004] Dhanant Subhadrabandhu, Saswati Sarkar, and Farooq Anjum. Efficacy of misuse detection in ad hoc networks. In *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 97–107, 2004.

[Wang *et al.*, 2016] 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.*, 2017a] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, 58:267–295, 2017.

[Wang *et al.*, 2017b] Yiyuan Wang, Dantong Ouyang, Liming Zhang, and Minghao Yin. A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. *Science China Information Sciences*, 60(6):062103, 2017.

[Wu *et al.*, 2006] Ping Wu, Ji-Rong Wen, Huan Liu, and Wei-Ying Ma. Query selection techniques for efficient crawling of structured web sources. In *Proceedings of the 22nd International Conference on Data Engineering*, pages 47–47. IEEE, 2006.

[Zhu *et al.*, 2012] Xu Zhu, Wei Wang, Shan Shan, Zhong Wang, and Weili Wu. A PTAS for the minimum weighted dominating set problem with smooth weights on unit disk graphs. *Journal of combinatorial optimization*, 23(4):443–450, 2012.