# Two-goal Local Search and Inference Rules for Minimum Dominating Set

**Shaowei Cai**[1,2*], **Wenying Hou**[3], **Yiyuan Wang**[4], **Chuan Luo**[5] and **Qingwei Lin**[5]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
[3]School of Software and Microelectronics, Peking University, China
[4]School of Computer Science and Information Technology, Northeast Normal University, China
[5]Microsoft Research, China
caisw@ios.ac.cn, houwyvenny@pku.edu.cn, yiyuanwangjlu@126.com

## Abstract

Minimum dominating set (MinDS) is a canonical NP-hard combinatorial optimization problem with applications. For large and hard instances one must resort to heuristic approaches to obtain good solutions within reasonable time. This paper develops an efficient local search algorithm for MinDS, which has two main ideas. The first one is a novel local search framework, while the second is a construction procedure with inference rules. Our algorithm named FastDS is evaluated on 4 standard benchmarks and 3 massive graphs benchmarks. FastDS obtains the best performance for almost all benchmarks, and obtains better solutions than state-of-the-art algorithms on massive graphs.

## 1 Introduction

Given an undirected graph $G = (V, E)$, a dominating set $D$ is a subset of vertices such that each vertex in $V \setminus D$ has at least one adjacent vertex in $D$. Dominating set is an important concept in graph theory, and have applications in diverse areas, especially in social networks [Wang *et al.*, 2009; Chalupa, 2018]. One typical problem is to find a smallest group of influential individuals or a set of initial seeds in a social network, so that all participants can be reached with only one hop from the seeds. This problem is equivalent to finding a minimum dominating set (MinDS) for the network.

The MinDS problem is a classic NP-hard problem, and cannot be approximated with a constant ratio under the assumption $P \neq NP$ [Raz and Safra, 1997]. Moreover, negative results have been proved for the approximation of MinDS even when limited to power law graphs [Gast *et al.*, 2015]. A number of works have been done on exact algorithms for MinDS, which mainly focus on improving the upper bound of running time. State-of-the-art exact algorithms for MinDS are based on the branch and reduce paradigm and can achieve a run time of $O(1.4969^n)$ [van Rooij and Bodlaender, 2011]. Fixed parameterized algorithms have allowed to obtain better complexity results [Karthik C. S. *et al.*, 2018]. The main focus of such algorithms is on theoretical aspects.

In practice, one usually resorts to heuristic approaches to obtain good solutions within reasonable time. Although greedy algorithms are fast, the obtained dominating sets are far from satisfactory. A comparison among several greedy heuristics for MinDS can be found in [Sanchis, 2002]. Heuristic search algorithms usually return very good or even optimal solutions within a reasonable time budget. Commonly used heuristic search methods, including genetic algorithm [Hedar and Ismail, 2010] and ant colony optimization [Jovanovic *et al.*, 2010; Potluri and Singh, 2011; Potluri and Singh, 2013], have been developed to solve MinDS. Hyper meta-heuristic algorithms combine different heuristic search algorithms and preprocessing techniques to obtain better performance [Potluri and Singh, 2013; Chaurasia and Singh, 2015; Bouamama and Blum, 2016; Lin *et al.*, 2016; Abu-Khzam *et al.*, 2017]. These algorithms were tested on standard benchmarks with up to thousand vertices. Recently, the configuration checking (CC) strategy [Cai *et al.*, 2011] has been applied to MinDS and led to two local search algorithms. Wang et al. proposed the CC$^2$FS algorithm for both unweighted and weighted MinDS [Wang *et al.*, 2017], and obtained better solutions than ACO-PP-LS [Potluri and Singh, 2013] on standard benchmarks. Afterwards, another CC-based local search named FastMWDS was proposed, which significantly improved CC$^2$FS on weighted massive graphs [Wang *et al.*, 2018]. Chalupa proposed an order-based randomised local search named RLS$_o$ [Chalupa, 2018], and achieved better results than ACO-LS and ACO-PP-LS [Potluri and Singh, 2011; Potluri and Singh, 2013] on standard benchmarks of unit disk graphs as well as some massive graphs. Very recently, Fan et. al. designed a local search algorithm named ScBppw [Fan *et al.*, 2019], based on two ideas including score checking and probabilistic random walk.

This work develops an efficient and robust algorithm for MinDS named FastDS, which achieves best results on almost all standard and massive benchmarks, and significantly outperforms state of the art algorithms on massive benchmarks. There are two main novel ideas in FastDS. The first one is a new local search framework. A typical local search framework for MinDS, is to solve the problem by iteratively solving the $k$-DS problem — given an integer $k$, searching for a $k$ sized dominating set [Fan *et al.*, 2019]. Specifically, when finding a $k$ sized feasible solution, the algorithm removes one vertex and goes on to search for a $(k-1)$ size solution, by

*Corresponding author

exchanging a pair of vertices in each step. We propose a new local search framework. When finding a $k$ size feasible solution, our algorithm goes on to search for a solution with $(k-1)$ or $(k-2)$ size. That is, our algorithm targets two goals during the search. This accelerates the convergence speed, and also expands the search region.

Moreover, we propose inference rules for MinDS, which are used to fix vertices that must be in some optimal solution. The inference rules are particularly useful for solving massive graphs. In order to solve combinatorial optimization problems on massive graphs, reduction rules have been implemented to reduce the size of the graphs. Examples include maximum clique [Cai and Lin, 2016], vertex cover [Cai *et al.*, 2017], k-plex [Gao *et al.*, 2018], as well as the weighted dominating set problem [Wang *et al.*, 2018]. Unfortunately, reduction rules become very restricted when coming to the MinDS problem, due to the special feature of the problem. For example, even if we can fix the value of a vertex to be 1 (being selected), we cannot delete it from the graph, as this vertex may be responsible for dominating other vertices in the remaining graph. Although there are some reduction rules for minimum weighted dominating set, they are based on the differentials among vertex weights [Wang *et al.*, 2018], and cannot be applied to MinDS as all vertices have the same weight there. We propose a compromise solution to design inference rules rather than reduction rules. We fix the value of some vertices using inference, without deleting them.

We conduct extensive experiments to compare FastDS with state-of-the-art heuristic algorithms on a broad range of benchmarks, including 4 standard benchmarks and 3 massive graphs benchmarks. FastDS obtains the best results for almost all benchmarks, and significantly outperforms state-of-the-art algorithms on the 3 massive benchmarks.

The paper is organized as follows. Section 2 presents basic definitions. Section 3 introduces the new local search framework. Sections 4 presents the inference rules. Section 5 describes our FastDS algorithm. Experimental results are shown in Section 6 and Section 7 gives concluding remarks.

## 2 Preliminaries

An undirected graph $G = (V, E)$ consists of a vertex set $V$ and an edge set $E$. Each edge $e = (v, u)$ connects two vertices $v$ and $u$, which are also called *endpoints* of $e$. Two vertices are *neighbors* if they belong to a same edge. For a vertex $v$, $N(v) = \{u \in V|(u, v) \in E\}$ is the set of neighbors of $v$, and the degree of $v$ is $degree(v) = |N(v)|$. The closed neighborhood of $v$ is $N[v] = N(v) \cup \{v\}$. Also, we use $N^2[v]$ to denote the set of $v$'s closed neighborhood and its neighbors' neighbors, i.e., $N^2[v] = N[v] \cup (\bigcup_{u \in N(v)} N(u))$. For a vertex set $S \subseteq V$, we use $N[S] = \bigcup_{v \in S} N[v]$ to denote the closed neighborhood of $S$.

A *candidate solution* (also known as partial solution) for MinDS is a subset of $V$. Given a candidate solution $D$, the value of a vertex $v$, denoted as $value(v)$, indicates whether it is selected in the candidate solution or not, which is 1 if $v \in D$, and 0 otherwise. A vertex $v$ is dominated by a candidate solution $D$ if $v \in N[D]$, and is non-dominated otherwise. The cost of a candidate solution $D$, denoted as $cost(D)$, is defined as the number of non-dominated vertices under $D$. A commonly used scoring function for MinDS is defined as

$$score(v) = cost(D) - cost(D')$$

where $D' = D \setminus \{v\}$ if $v \in D$, and $D' = D \cup \{v\}$ otherwise. Obviously, $score(v) \leq 0$ if $v \in D$, and $score(v) \geq 0$ if $v \notin D$.

We use $D$ to denote the current candidate solution, i.e., the set of vertices currently selected for dominating, and $U$ to denote the set of non-dominated vertices. $D^*$ is the best solution found so far, and $D^*$ at the end of the algorithm is returned by the algorithm. The *age* of a vertex $v$ is the number of steps that have occurred since $v$ last changed its state.

## 3 Two-goal Local Search

We propose a novel local search framework for MinDS, which allows the algorithm to target two different sizes during each period of the search.

For subset problems whose goal is to find a minimum set satisfying given constraints, such as MinDS, minimum vertex cover (MinVC) and minimum set cover, a popular paradigm of local search is to iteratively find a feasible solution of a fixed size [Musliu, 2006; Richter *et al.*, 2007; Cai *et al.*, 2013; Gao *et al.*, 2015; Fan *et al.*, 2019; Li *et al.*, 2019]. Specifically, when finding a $k$-sized solution, the algorithm focuses on searching for a ($k$-1)-sized solution by fixing the size of the partial solution to be $k$-1 at the end of each step. To this end, a popular local search framework is the two-stage exchanging framework [Cai *et al.*, 2013], where one vertex is removed from the partial solution in the first stage and another vertex is added in the second stage. Because of its good performance and low complexity, the two-stage exchanging framework has been applied to local search algorithms for the MinDS problem [Fan *et al.*, 2019].

We propose a new local search framework for MinDS (Algorithm 1). The local search starts from an initial feasible solution (line 2) and searches for a feasible solution as small as possible. Specifically, when $D$ becomes a feasible solution, $D^*$ is updated (line 5) and then a vertex is removed from $D$ (line 6). When $D$ is not a feasible solution, the algorithm iteratively exchanges vertices until $D$ becomes a new dominating set. It first removes vertices to make $|D| = |D^*| - 3$; in the adding phase, one vertex is added to $D$, and after that, if $D$ is still infeasible, another vertex is added with a certain probability. Thus, the size of $D$ can be $|D^*| - 1$ or $|D^*| - 2$ at the end of a search step. Finally, the best found dominating set $D^*$ is returned when the time limit is reached.

Previous local search algorithms focus on seeking a ($k$-1)-sized solution after finding a $k$-sized solution. Differently, our framework searches for a ($k$-1)-sized or ($k$-2)-sized solution after it finds a $k$-sized solution. This has two advantages. (1) It accelerates the converge of the search, as the algorithm sometimes directly finds a ($k$-2)-sized solution without finding a ($k$-1)-sized one. Note that if we set the goal to be ($k$-2) after finding a $k$-sized solution, we would fail to find the optimal solution if its size is ($k$-1). Our framework with two goals can improve the search efficiency without losing the ability to find the optimal solution. (2) It expands the search region for

**Algorithm 1:** A New Local Search Framework for MinDS

```
Input: An undirected graph $G = (V, E)$
Output: A dominating set $D$ of $G$
1  begin
2      $D \leftarrow ConstructDS()$;
3      while elapsed_time < cutoff do
4          if all vertices are dominated then
5              $D^* \leftarrow D$;
6              remove a vertex from $D$;
7              continue;
8          Remove vertices from $D$ until $|D| = |D^*| - 3$;
9          Add one vertex to $D$;
10         if $\exists$ non-dominated vertices then
11             if with certain probability then
12                 Add another vertex to $D$;
13     return $D^*$;
```

the adding stage by keeping a smaller candidate solution at the end of the removing stage. A smaller candidate solution leads to a larger $N[U]$ ($U$ is the set of non-dominated vertices), which implies more candidate adding vertices in the adding stage. Finally, we note that our local search framework can be applied to other subset problems.

## 4 Inference Rules for MinDS

To improve the performance of local search for MinDS on large instances, we propose three inference rules, which can fix a considerable portion of the vertices for massive graphs.

**Isolated vertex rule:** If a vertex $v$ is not adjacent to any other vertices, or all its neighbors have a fixed value 0 (not selected), then $v$'s value is fixed to 1.

This rule is obvious: in order to make the vertex $v$ dominated, there is only one option — including $v$ in the solution.

**Leaf rule:** If a vertex $v$ is adjacent to only one vertex $u$, or all its neighbors but $u$ have a fixed value 0, then $value(u)$ is fixed to 1 and $value(v)$ is fixed to 0.

The reasoning of this rule is as follows. In order to dominate vertex $v$, at least one of $u$ and $v$ should be selected, and we prove that the best way to do so is to choose $u$ and not choose $v$. (1) If both $u$ and $v$ are selected in a dominating set $D$, then $D \backslash \{v\}$ is a smaller dominating set; (2) If $v$ is selected and $u$ is not, i.e., $v \in D$ and $u \notin D$, then we can exchange $u$ and $v$ and the resulting set $D \setminus \{v\} \cup \{u\}$ is still a dominating set, as removing $v$ can only have impact on vertices $u$ and $v$, and by adding $v$ we dominate both of them.

**Triangle rule:** For a triangle $\{u, v, w\}$ in the graph, where $N(u) = \{v, w\}$, $N(v) = \{u, w\}$ and $N(w) \supset \{u, v\}$, $value(w)$ is fixed to 1 while $value(u)$ and $value(v)$ are fixed to 0.

The reasoning follows as this: For any dominating set $D$, in order to dominate vertices $u$ and $v$, at least one vertex from $\{u, v, w\}$ is in $D$. Now, since $N[u] = N[v] = \{u, v, w\} \subset N[w]$, we know that, any vertex that can be dominated by $u$ and $v$ can be dominated by $w$, but not vice versa.

We can generalize the Triangle rule as follows. However, due to the high computation cost, the generalized rule is not implemented in our algorithm.

**Generalized Triangle rule:** For a vertex $w$ and a vertex set $S \subseteq N(w)$, if $N[S] \subset N[w]$, then $value(w)$ is fixed to 1, and the value of any vertex in $S$ is fixed to 0.

## 5 The FastDS Algorithm

We develop a local search algorithm for MinDS named FastDS (Algorithm 3), which is based on our two-goal framework and employs inference rules.

### 5.1 Initial Construction

In the beginning, the algorithm constructs an initial solution by a greedy construction procedure, making use of the inference rules. A key concept of the algorithm is the non-dominated neighborhood and non-dominated degree.

**Definition 1.** *Given a graph $G = (V, E)$ and a vertex subset $D$, the non-dominated neighborhood of a vertex $v$ is $N_r[v] = \{u|u \in N[v], N[u] \cap D = \emptyset\}$, and the non-dominated degree of $v$ is $degree_r(v) = |N_r[v]|$.*

Intuitively, the set $N_r[v]$ includes all the vertices that are currently non-dominated and would be dominated if $v$ is added to the candidate solution $D$. Thus, $degree_r(v)$ measures the benefits of adding vertex $v$.

The construction procedure fist traverses all vertices and employs inference rules to fix the vertices satisfying at least one of the rules. Then, it utilizes a greedy strategy based on non-dominated degrees to iteratively add vertices until a dominating set is obtained. At the end of the construction, redundant vertices are removed from $D$ to make it minimal.

**Algorithm 2:** FastDS

```
Input: An undirected graph $G = (V, E)$
Output: A dominating set $D^*$ of $G$
1  begin
2      $D \leftarrow ConstructDS(G)$;
3      while elapsed_time < cutoff do
4          if $U = \emptyset$ then
5              remove redundant vertices from $D$;
6              $D^* \leftarrow D$;
7              remove a vertex with highest $score$ from $D$;
8          $u_1 \leftarrow$ a vertex in $D$ according to BMS heuristic;
9          $D \leftarrow D \setminus \{u_1\}$;
10         if $|D| = |D^*| - 2$ then
11             $u_2 \leftarrow$ a random vertex from $D$;
12             $D \leftarrow D \setminus \{u_2\}$;
13         $v_1 \leftarrow$ a vertex in $N[U]$ with highest $score$,
             breaking ties by $age$;
14         $D \leftarrow D \cup \{v_1\}$;
15         if $U \neq \emptyset$ then
16             if with probability $\alpha$ then
17                 $v_2 \leftarrow$ a vertex in $N[U]$ with highest $score$,
                     breaking ties by $age$;
18                 $D \leftarrow D \cup \{v_2\}$;
19     return $D^*$;
```

## 5.2 Local Search

The constructed initial solution is then fed to the local search phase for improving. During the local search process, whenever $D$ becomes a feasible solution, redundant vertices (if any) are first removed from $D$, and $D^*$ is updated to $D$. After that, a vertex with the maximum $score$ (i.e., the minimum loss) is removed. Then, the local search iteratively executes exchanging steps until $D$ becomes a feasible solution.

Each exchanging step consists of a removing stage and an adding stage. In the removing stage (lines 8-12), one or two vertices are removed from $D$. The first vertex $u_1$ is selected from $D$ according to a sampling heuristic named Best from Multiple Selections (BMS) [Cai, 2015], which randomly samples $t$ vertices in $D$ and picks the one with the maximum $score$ to remove ($t$ is suggested near 50 [Cai, 2015] and is set to 45 in this work). If there is more than one such vertex, the vertex with the largest $age$ is selected to remove. Additionally, after $u_1$ is removed from $D$, if $|D^*|-|D|$ equals 2 rather than 3, one more vertex is removed, to ensure $|D| = |D^*| - 3$ at the end of the removing stage.

In the adding stage (lines 13-18), one or two vertices are added into $D$. First, a vertex $v_1 \in N[U]$ ($U$ is the set of non-dominated vertices) with the highest $score$ is added to $D$, breaking ties by preferring the one with largest $age$[1]. Then, if there are non-dominated vertices, with a probability of $\alpha$, a second vertex $v_2$ is selected in the same way as the first one and added to $D$.

An important implementation detail is that, when removing or adding a vertex $x$, the $score$ of vertices in $N^2[x]$ may change, and should be updated accordingly.

| Instance | CC$^2$FS | FastMWDS | RLS$_o$ | ScBppw | FastDS |
|---|---|---|---|---|---|
| Family | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ |
| V100U150 | **17** | **17** | **17** | 17.3 | **17** |
| V100U200 | **10.4** | **10.4** | **10.4** | 10.6 | **10.4** |
| V250U150 | **18** | **18** | **18** | 19 | **18** |
| V250U200 | **10.8** | **10.8** | **10.8** | 11.5 | **10.8** |
| V500U150 | **18.5** | **18.5** | 18.6 | 20.1 | **18.5** |
| V500U200 | **11.2** | **11.2** | **11.2** | 12.4 | **11.2** |
| V50U150 | **12.9** | **12.9** | **12.9** | **12.9** | **12.9** |
| V50U200 | **9.4** | **9.4** | **9.4** | **9.4** | **9.4** |
| V800U150 | **19** | **19** | 19.1 | 20.9 | **19** |
| V800U200 | **11.7** | **11.7** | 11.9 | 12.6 | 11.8 |
| V1000U150 | **19.1** | **19.1** | 19.2 | 21.3 | **19.1** |
| V1000U200 | **12** | **12** | **12** | 13 | **12** |

Table 1: Experiment results on UDG benchmark

| Instance | CC$^2$FS | MWDS | RLS$_0$ | ScBpw | FastDS | Instance | CC$^2$FS | MWDS | RLS$_0$ | ScBpw | FastDS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Family | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ | Family | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ | $D_{min}$ |
| V100E100 | **33.6** | **33.6** | **33.6** | **33.6** | **33.6** | V250E5000 | **11** | **11** | 11.2 | 11.5 | **11** |
| V100E1000 | **7.5** | **7.5** | **7.5** | 7.8 | **7.5** | V250E750 | **44** | **44** | 44.3 | 44.9 | **44** |
| V100E2000 | **4.1** | **4.1** | **4.1** | 4.3 | **4.1** | V300E1000 | **48.6** | **48.6** | 49.1 | 49.5 | **48.6** |
| V100E250 | **19.9** | **19.9** | **19.9** | 20.1 | **19.9** | V300E2000 | **29.4** | **29.4** | 30 | 30.6 | **29.4** |
| V100E500 | **12.2** | **12.2** | **12.2** | 12.5 | **12.2** | V300E300 | **100** | **100** | **100** | 100.1 | **100** |
| V100E750 | **9** | **9** | **9** | 9.5 | **9** | V300E3000 | **22** | **22** | 22.8 | 22.9 | **22** |
| V150E1000 | **15** | **15** | 15.1 | 15.4 | **15** | V300E500 | **77.7** | **77.7** | 77.8 | 78.4 | **77.7** |
| V150E150 | **50** | **50** | **50** | **50** | **50** | V300E5000 | **15.1** | **15.1** | 15.7 | 15.9 | **15.1** |
| V150E2000 | **9** | **9** | **9** | 9.5 | **9** | V300E750 | **59.6** | **59.6** | **59.6** | 60 | **59.6** |
| V150E250 | **39.1** | **39.1** | 39.3 | 39.3 | **39.1** | V500E1000 | **114.7** | **114.7** | 114.8 | 116.3 | **114.7** |
| V150E3000 | **6.9** | **6.9** | **6.9** | **6.9** | **6.9** | V500E10000 | **22** | 22.1 | 23.2 | 23.3 | 22.2 |
| V150E500 | **24.6** | **24.6** | 24.7 | 24.9 | **24.6** | V500E2000 | **71.2** | **71.2** | 72.5 | 72.9 | **71.2** |
| V150E750 | **18.3** | **18.3** | 18.5 | 18.8 | **18.3** | V500E500 | **167** | **167** | **167** | **167** | **167** |
| V200E1000 | **24.4** | **24.4** | 24.7 | 25 | **24.4** | V500E5000 | **36.9** | **36.9** | 38.4 | 38.7 | **36.9** |
| V200E2000 | **15** | **15** | **15** | 15.3 | **15** | V800E100 | **267** | **267** | **267** | **267** | **267** |
| V200E250 | **61.1** | **61.1** | **61.1** | 61.7 | **61.1** | V800E1000 | **242.5** | **242.5** | **242.5** | 246.5 | **242.5** |
| V200E3000 | **11** | **11** | 11.1 | 11.3 | **11** | V800E10000 | 50.4 | 50.3 | 52.7 | 53.1 | **50.2** |
| V200E500 | **39.6** | **39.6** | **39.6** | 40 | **39.6** | V800E2000 | **158.3** | **158.3** | 159.4 | 162.1 | **158.3** |
| V200E750 | **30** | **30** | 30.1 | 30.4 | **30** | V800E5000 | **82.6** | **82.6** | 85.9 | 86.3 | **82.6** |
| V250E1000 | **36** | **36** | 36.3 | 36.6 | **36** | V1000E1000 | **333.7** | 334 | **333.7** | **333.7** | **333.7** |
| V250E2000 | **21.6** | **21.6** | 22.1 | 22.4 | **21.6** | V1000E10000 | 74.2 | 74.1 | 77.8 | 77.6 | **74** |
| V250E250 | **83.3** | **83.3** | **83.3** | **83.3** | **83.3** | V1000E15000 | 55.5 | 55.1 | 58.2 | 58.2 | **55** |
| V250E3000 | **16** | **16** | 16.5 | 16.6 | **16** | V1000E20000 | 45.2 | 45.4 | 47.1 | 47.4 | **45** |
| V250E500 | **57.8** | **57.8** | 57.9 | 58.1 | **57.8** | V1000E5000 | 121.2 | **121.1** | 125.3 | 126 | **121.1** |

Table 2: Experiment results on T1 benchmark. To save space, we denote FastMWDS as MWDS.

## 6 Experiments

We evaluate FastDS on 7 benchmarks, including 4 standard benchmarks in the literatures and 3 massive benchmarks.

**UDG**: This is a widely used standard benchmark for MinDS [Hedar and Ismail, 2010; Potluri and Singh, 2011; Potluri and Singh, 2013; Chalupa, 2018]. UDG contains 120 instances generated by the topology generator[2]. UDG is divided into 12 families, each of which has 10 instances.

**T1**: This data set consists of 520 instances where each instance has two different weight functions [Jovanovic *et al.*, 2010]. We select these original graphs where the weight of each vertex is set to 1. There are 52 families, each of which contains 10 instances with the same size. To save space, we do not report the results on graphs with 50 vertices.

**DIMACS**: This benchmark is from the Second DIMACS Implementation Challenge (1992-1993)[3], including problems from real applications and randomly generated graphs.

**BHOSLIB**: This benchmark are generated based on the RB model [Xu *et al.*, 2007] near the phase transition. It is a popular benchmark for graph theoretic problems.

**SNAP**: This benchmark is from Stanford Large Network Dataset Collection[4]. It is a collection of real world graphs from $10^4$ vertices to $10^7$ vertices.

**DIMACS10**: This benchmark is from the 10th DIMACS implementation challenge (2010)[5], which aims to provide large challenging instances for graph theoretic problems.

**Network Repository**: The Network Data Repository [Rossi and Ahmed, 2015] collects massive graphs from various areas. Many of the graphs have 100 thousands or millions of vertices. This benchmark has been widely used for graph theoretic problems including vertex cover, clique, coloring, and dominating set problems.

For SNAP and DIMACS10 benchmarks, as in previous literatures [Lin *et al.*, 2017; Verma *et al.*, 2015], we only report the results on graphs with at least 30 000 vertices. For Repository benchmark, we choose the graphs with at least $10^5$ vertices. This results in 22, 31 and 65 graphs in SNAP, DIMACS10, and Repository, respectively.

### 6.1 Experiment Setup

FastDS is implemented in C++ and complied by g++ with '-O3' option, and the parameter $\alpha$ is set to 0.6. All experiments are run on a server with Intel Xeon E5-2640 v4 2.40GHz with 128GB RAM under CentOS 7.5.

[1]In practice, acceleration can be gained for the classic benchmarks by picking a random non-dominated vertex with a small probability such as 0.005 here.

[2]http://www.michelemastrogiovanni.net/software/download/

[3]ftp://dimacs.rutgers.edu/pub/challenges

[4]http://snap.stanford.edu/data

[5]https://www.cc.gatech.edu/dimacs10/

|  | CC$^2$FS | FastMWDS | RLS$_0$ | ScBppw | FastDS |
|---|---|---|---|---|---|
| Instance | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ |
| frb40-19-1 | **14** | **14** | 14(15) | 15(15.8) | **14** |
| frb40-19-2 | **14** | **14** | 15(15.4) | 16(16.5) | **14**(14.3) |
| frb40-19-3 | 14(14.8) | **14(14.6)** | 15(15.6) | 16(16.5) | 14(14.7) |
| frb40-19-4 | **14** | **14** | 15(15.8) | 15(16.2) | **14** |
| frb40-19-5 | **14** | **14** | 15(15.1) | 15(15.2) | **14** |
| frb45-21-1 | **16** | **16** | 16(17.4) | 17(18) | **16** |
| frb45-21-2 | 16(16.2) | **16** | 17(17.8) | 18(18.1) | **16** |
| frb45-21-3 | **16** | **16** | 17(17.5) | 17(17.8) | **16** |
| frb45-21-4 | **16** | **16** | 17(17.8) | 17(18) | **16** |
| frb45-21-5 | **16** | **16** | 17(17.7) | 18(18.2) | **16** |
| frb50-23-1 | **18** | **18** | 19(19.6) | 19(19.9) | **18** |
| frb50-23-2 | **18** | **18** | 19(20.1) | 19(19.8) | **18** |
| frb50-23-3 | **18** | **18** | 19(20) | 20 | **18** |
| frb50-23-4 | **18** | **18** | 20(20.1) | 19(20) | **18** |
| frb50-23-5 | **18** | **18** | 19(19.8) | 19(19.9) | **18** |
| frb53-24-1 | **19** | **19** | 20(21.1) | 21(21.1) | **19** |
| frb53-24-2 | **19** | 19(19.1) | 20(21.4) | 21(21.2) | **19** |
| frb53-24-3 | **19** | **19** | 20(20.5) | 20(20.4) | **19** |
| frb53-24-4 | **18(18.7)** | 19 | 19(20) | 19(20.2) | **18(18.7)** |
| frb53-24-5 | 19(19.1) | **19** | 20(21.3) | 20(21.5) | **19** |
| frb56-25-1 | **20** | **20** | 21(21.8) | 21(22.2) | **20** |
| frb56-25-2 | **20** | 20(20.1) | 21(22.2) | 21(22.5) | **20** |
| frb56-25-3 | **20** | **20** | 22(22.7) | 21(22.2) | **20** |
| frb56-25-4 | 20(20.5) | 20(20.9) | 22(22.7) | 22(22.5) | **20** |
| frb56-25-5 | **20** | **20** | 21(21.5) | 21(21.8) | **20** |
| frb59-26-1 | 21 | 21 | 22(23) | 22(22.6) | **20(20.5)** |
| frb59-26-2 | **21** | **21** | 22(22.8) | 22(22.6) | **21** |
| frb59-26-3 | 21(21.7) | **21**(21.5) | 22(23.1) | 23(23.1) | **21** |
| frb59-26-4 | 21(21.1) | **21**(21.1) | 23(23.6) | 23(23.6) | **21** |
| frb59-26-5 | 21(21.9) | 21(21.8) | 24(24.3) | 23(24) | **21** |
| frb100-40 | 37(37.9) | 38(38.1) | 40(40.7) | 39(40.1) | **36(36.4)** |

Table 3: Experiment results on BHOSLIB benchmark

| Benchmark | CC$^2$FS | FastMWDS | FastDS |
|---|---|---|---|
| UDG | 0.21s | 0.83s | 22.19s |
| T1 | 4.88s | 8.35s | 11.23s |
| BHOSLIB | 96.68s | 101.44s | 95.62s |

Table 4: Averaged run time on standard benchmarks

We compare FastDS with four state-of-the-art heuristic algorithms, including CC$^2$FS [Wang *et al.*, 2017], FastMWDS [Wang *et al.*, 2018], RLS$_o$ [Chalupa, 2018] and ScBppw [Fan *et al.*, 2019]. CC$^2$FS is good at solving standard benchmarks, while FastMWDS and ScBppw are designed to solve massive graphs, and RLS$_o$ is a recent algorithm that outperforms previous ant optimization and hyper meta-heuristic algorithms. The codes of CC$^2$FS, FastMWDS and RLS$_o$ were kindly provided by their authors while the codes of ScBppw were open online[6]. All of them are implemented in C++ and complied by g++ with '-O3' option.

For each instance, all algorithms are executed 10 times with random seeds 1,2,3...10. The time limit of each run is 1000 seconds. For each instance, we report the best size ($D_{min}$) and the average size ($D_{avg}$) of the solutions found over the 10 runs. **When $D_{avg} = D_{min}$, we do not report $D_{avg}$.** For UDG and T1, we report for each family the averaged value of $D_{min}$, denoted as $\overline{D_{min}}$. When failing to find a solution in time limit, the cell is marked as 'N/A'.

## 6.2 Results on Standard Benchmarks

Results on UDG, T1 and BHOSLIB benchmarks are reported in Tables 1, 2 and 3 respectively. The DIMACS instances are so easy that CC$^2$FS, FastMWDS and FastDS find the same solutions on all the instances quickly, but RLS$_o$ and ScBppw are worse. For all standard benchmarks, FastDS has the best performance in terms of solution quality, except slightly worse than CC$^2$FS and FastMWDS on UDG benchmark. Over all standard benchmarks, CC$^2$FS, FastMWDS and FastDS are essentially better than the other algorithms. We compare the averaged run time of these three algorithms on these benchmarks (Table 4), where the run time of each run of an algorithm is the time to reach the final solution.

[6]https://github.com/Fan-Yi/

|  | FastMWDS | RLS$_0$ | ScBppw | FastDS |
|---|---|---|---|---|
| Instance | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ | $D_{min}(D_{avg})$ |
| Amazon0302 | 35616(35619.7) | 38735(38771.1) | 36199(36212.5) | **35593(35602.7)** |
| Amazon0312 | 45531(45538) | 49326(49362.7) | 45914(45931.9) | **45490(45493.7)** |
| Amazon0505 | 47362(47374.6) | 51281(51306.9) | 47734(47748.4) | **47310(47317.7)** |
| Amazon0601 | 42319(42325.5) | 45952(45989.9) | 42717(42732.8) | **42289(42294.2)** |
| Cit-HepPh | **3078(3078.6)** | 3192(3209.3) | 3087(3088.9) | **3078**(3078.9) |
| Cit-HepTh | **2935(2935.7)** | 2985(2993.4) | 2944(2948.6) | 2936(2936.1) |
| cit-Patents | 650610(651187.7) | 667418(667594.3) | 622886(622999.5) | **621722(621742.3)** |
| Email-EuAll | **18181** | 18181(18181.6) | **18181** | **18181** |
| p2p-Gnutela04 | **2227** | **2227** | **2227** | **2227** |
| p2p-Gnutela24 | **5418(5418)** | **5418**(5418.5) | **5418** | **5418** |
| p2p-Gnutela25 | **4519** | **4519**(4519.1) | **4519** | **4519** |
| p2p-Gnutela30 | **7169** | **7169**(7169.4) | **7169**(7169.2) | **7169** |
| p2p-Gnutela31 | **12582** | 12593(12596.1) | **12582** | **12582** |
| Slashdot0811 | **14312(14312)** | 14333(14337.3) | **14312(14312)** | **14312(14312)** |
| Slashdot0902 | **15305** | 15334(15338.4) | **15305** | **15305** |
| soc-Epinions1 | **15734** | 15742(15746.3) | **15734** | **15734** |
| web-BerkStan | 28434 | 30119(30152.4) | 28615(28628.9) | **28432(28436.1)** |
| web-Google | 79700(79704.3) | 81106(81122.1) | 79756(79761.8) | **79699** |
| web-NotDame | **23733(23734.4)** | 23950(23957.5) | 23746 | 23735 |
| web-Stanford | **13198(13199.1)** | 14099(14117.6) | 13298(13314.2) | 13199(13200.7) |
| WikiTalk | **36960** | **36960**(36961.4) | **36960** | **36960** |
| Wiki-Vote | **1116** | **1116** | **1116** | **1116** |
| as-22july06 | **2026** | **2026** | **2026** | **2026** |
| caidaR*Level | **40522(40522.8)** | 41647(41663.4) | 40573(40580.2) | 40523 |
| citattionCit*r | **43412** | 44936(44958.3) | 43430(43434.3) | **43412** |
| cnr-2000 | **22006(22008.5)** | 22313(22332.1) | 22027(22032.6) | 22011(22012.2) |
| coAuthorCit*r | **33197** | 33790(33818.7) | 33343(33358.1) | **33197(33197)** |
| coAuth*DBLP | **43978** | 44640 | 44084(44094.9) | **43978** |
| cond-mat2005 | **650** | 6514(6518.7) | 6531(6537.5) | **6508** |
| coPapersCit*r | 26090(26093.1) | 28693(28716.3) | 26989(27008) | **26082** |
| coPaperDBLP | 35629(35636.4) | 39883(39939.1) | 37060(37090.3) | **35597(35599.2)** |
| eu-2005 | 32292(32300.2) | 32812(32831.8) | 32322(32330.8) | **32284(32287.7)** |
| in-2004 | 77835(77838.7) | 79016(79036.8) | 77894(77903.2) | **77822(77827.4)** |
| kron*500log16 | 14105(14107.1) | 14103(14105.2) | **14100** | **14100** |
| rgg_n_2_17_s0 | 12378(12385.7) | 14705(14745) | 13827(13849.4) | **12334(12335.9)** |
| rgg_n_2_19_s0 | 44702(44732.6) | 55482(55549.2) | 50375(50437.4) | **44423(44436.7)** |
| rgg_n_2_20_s0 | 87222(89920.5) | 106700(106754.8) | 96649(96703.5) | **84708(84729.6)** |
| rgg_n_2_21_s0 | 180777(181900) | 204553(204696) | 185387(185478) | **162266(162313)** |
| rgg_n_2_22_s0 | 369858(370458) | 393433(393551) | 356868(356945) | **312350(312462)** |
| rgg_n_2_23_s0 | 740096(740750) | 757817(758064) | 687552(687945.8) | **605278(605592)** |
| rgg_n_2_24_s0 | 1448537 (1449017.6) | N/A | 1328020 (1328330) | **1189207 (1190047.3)** |
| uk-2002 | 1085564 (1085668.2) | N/A | 1077327 (1077391.7) | **1075943 (1075951)** |
| 333SP | 676146(676829.7) | 698206(698363.9) | 628797(629054) | **563599(563738)** |
| audikw1 | 9496(9510.3) | 11389(11416.7) | 10534(10568.5) | **9460(9471.5)** |
| belgium.osm | 505880(506566) | 537618(537898) | 476671(476809.9) | **468852(468863)** |
| cage15 | 518491(518875) | 504722(504881) | 476574(476878.5) | **456062(456340)** |
| ecology1 | 241117(241345) | 269422(269552) | 238052(238250.4) | **201498(201669)** |
| G_n_pin_pout | 13303(13320.4) | 14121(14160.4) | 12638(12670.2) | **12255(12275.7)** |
| ldoor | 19798(19807.1) | 24010(24065.9) | 21527(21546.4) | **19722(19730.9)** |
| luxem*g.osm | 37753(37754.4) | 41229(41333.1) | 38421(38444.8) | **37751(37752.1)** |
| preferAttach* | 8147(8151.2) | 8961(8987.4) | 8167(8180.1) | **8146(8150)** |
| smallworld | 10756(10767.6) | 12744(12769.3) | 11733(11770.8) | **10451(10456.3)** |
| wave | **11549(11562.2)** | 15007(15048.8) | 13910(13927.4) | 11619(11675.7) |

Table 5: Experiment results on SNAP and DIMACS10 benchmarks

## 6.3 Results on Massive Benchmarks

Results on SNAP and DIMACS10 benchmarks are reported in Table 5, and results on Repository benchmarks are reported in Table 6. CC$^2$FS is dominated by FastMWDS on all the instances and not reported in Tables 5 and 6. FastDS performs best for all the massive benchmarks. It obtains the best solutions for 19 (out of 22) SNAP instances, 28 (out of 31) DIMACS10 instances and 63 (out of 65) Repository

| Instance | FastMWDS $D_{min}(D_{avg})$ | RLS$_0$ $D_{min}(D_{avg})$ | ScBppw $D_{min}(D_{avg})$ | FastDS $D_{min}(D_{avg})$ |
|---|---|---|---|---|
| bn-human-B*1 | 1189968 (1189990) | 1190728 (1190752) | 1190244 (1190256) | **1189874 (1189877.3)** |
| bn-human-B*2 | 1550899 (1550903.4) | 1551502 (1551519) | 1551107 (1551119.2) | **1550863 (1550865.6)** |
| ca-co*-dblp | 35629(35638.7) | 39880(39938.6) | 37060(37090.3) | **35597(35599.2)** |
| ca-dblp-2012 | **46138** | 46801(46818.5) | 46248(46262) | **46138** |
| ca-hollyw*d-09 | 51045(51099) | 53073(53109.2) | 50334(50364.5) | **48740(48748.3)** |
| channel*-b050 | 407609(408691.8) | 421739(421938.9) | 392298(392409.6) | **342645(342886.3)** |
| dbpedia-link | 1537033(1537076) | N/A | **1536656**(1536657) | **1536656** |
| delaunay_n22 | 747758(748471.6) | 759903(760115.2) | 689061(689178.8) | **640941(641227.5)** |
| delaunay_n23 | 1516416 (1517199) | 1519638 (1520217.4) | 1378243 (1378425.1) | **1288297 (1289304.2)** |
| delaunay_n24 | 3044962 (3046047.8) | N/A | 2756088 (2756512.2) | **2627137 (2632372)** |
| friendster | 657039(657107.2) | 661305(661382.7) | 656464(656466.4) | **656463** |
| hugebubs-*20 | 6782951 (6783747.3) | N/A | 5984446 (5984906.2) | **5926263 (5953412.2)** |
| hugetrace-*10 | 3850232 (3850634.6) | N/A | 3392580 (3392947.6) | **3219420 (3230203.8)** |
| hugetrace-*20 | 5112192 (5113235.7) | N/A | 4508924 (4509570) | **4363372 (4380777.8)** |
| inf-euro_osm | 18854928 (18855875.7) | N/A | 17006807 (17012959.9) | **17111963** (17156686.5) |
| inf-germ*_osm | 4237300 (4238386.6) | N/A | 3846494 (3846917.4) | **3797838 (3799558.6)** |
| inf-roadNet-CA | 633952(634444.4) | 662093(662374.9) | 595726(595798.5) | **586513(586550.2)** |
| inf-roadNet-PA | 345733(345831.6) | 369943(370080.6) | 332079(332142.9) | **326934(326951.8)** |
| inf-road-usa | 8400667 (8401988) | N/A | 7852540 (7853213.5) | **7814911 (7818726.7)** |
| rec-dating | 11740(11742.3) | 11770(11776.4) | 11736(11737.6) | **11735(11735.7)** |
| rec-epinions | **9598** | 9655(9660) | 9599 | **9598** |
| rec-libimset-dir | 12973(12980.4) | 13131(13139.9) | **12955**(12956) | **12955(12955.7)** |
| rgg_n_2_23_s0 | 740939(741675.6) | 757817(758064) | 687552(687946) | **605922(606268.9)** |
| rgg_n_2_24_s0 | 1448992 (1449583.4) | N/A | 1328020 (1328333.3) | **1190963 (1194498.1)** |
| rt-retw*-crawl | **75740** | 75816(75822) | **75740**(75740.1) | **75740** |
| sc-ldoor | 62474(62486.9) | 66589(66625.1) | 66244(66309.2) | **62411(62422.8)** |
| sc-msdoor | 19697(19710.1) | 21408(21438) | 21372(21393.2) | **19678(19683.3)** |
| sc-pwtk | **4194(4197.8)** | 5341(5367.7) | 5479(5504.5) | 4200(4201.8) |
| sc-rel9 | **123050(125726.7)** | 143998(148659.7) | 124845(129947) | 127548(127632.2) |
| sc-shipsec1 | 7713(7727.7) | 9309(9338.9) | 8787(8800.1) | **7662(7672.5)** |
| sc-shipsec5 | 10381(10394.6) | 12560(12577.7) | 11873(11894.1) | **10300(10306.9)** |
| soc-buzznet | **127** | 130(131.6) | **127** | **127** |
| soc-delicious | **55722** | 55969(55984.3) | 55726(55727.8) | **55722** |
| soc-digg | **66155** | 66641(66671.1) | **66155**(66156.6) | **66155** |
| soc-dogster | 26260(26264.5) | 26540(26569.1) | 26255(26258) | **26253** |
| socfb-A-anon | 201774(201793.6) | 203194(203218.6) | **201690**(201690.5) | **201690** |
| socfb-B-anon | 187031(187036) | 187829(187850.7) | **187030**(187030.1) | **187030** |
| socfb-uci-uni | 865675(865675.1) | N/A | **865675** | **865675** |
| soc-flickr | 98062(98062.5) | 98640(98658.7) | 98063(98064.9) | **98062** |
| soc-flickr-und | 295773(295790.9) | 296979(296991.9) | 295702(295705.1) | **295700** |
| soc-flixster | **91019** | 91127(91131.9) | **91019** | **91019** |
| soc-FourSq* | 60982(60985.8) | 61301(61316.3) | **60979** | **60979** |
| soc-lastfm | **67226** | 67264(67271.4) | **67226** | **67226** |
| soc-livejour* | 803448(803626.7) | 815040(815178.6) | 793988(793997.1) | **793887(793887.9)** |
| soc-live*groups | 1072271 (1072315.9) | 1074058 (1074107.7) | 1071123 (1071124) | **1071123** |
| soc-LiveMocha | **1424** | 1475(1482) | **1424** | **1424** |
| soc-ljour*2008 | 1015864 (1016080.1) | 1025928 (1026043.5) | 1005983 (1005988.6) | **1005858** |
| soc-orkut | 120334(120416.1) | 124516(124581.3) | 111087(111117.8) | **110547(110585.8)** |
| soc-orkut-dir | 100971(101010.1) | 104751(104867.9) | 93974(94013.3) | **93630(93644.8)** |
| soc-pokec | 213149(213242) | 221665(221816.5) | 207385(207397) | **207308(207312.3)** |
| soc-sinaweibo | **201328** | N/A | **201328** | **201328** |
| soc-twit*-higgs | 14751(14763.3) | 15235(15257.8) | **14689**(14690.6) | **14689(14689.4)** |
| soc-youtube | 89732 | 90525(90558.2) | 89734(89735.3) | **89732** |
| soc-yout*-snap | 213167(213174.6) | 214014(214041.8) | **213122**(213123.3) | **213122** |
| tech-as-skitter | 183324(183378.8) | 186588(186648.6) | 181852(181869.8) | **181717(181717.8)** |
| tech-ip | 160(162.9) | 174(176.1) | 155(157.9) | **154(154.8)** |
| twitter_mpi | 566378(566401.4) | 567050(567079.3) | **566315**(566315.5) | **566315(566315)** |
| web-arab*05 | **16901(16901.3)** | 17176(17191.3) | 16907(16909.5) | **16901(16901.4)** |
| web-*-baike | 278191(278204.7) | 279702(279727.1) | 277849(277850.4) | **277847** |
| web-it-2004 | **32997** | 33085(33093.6) | 32998(32999.3) | **32997** |
| web-uk-2005 | **1421** | **1421** | **1421** | **1421** |
| web-wik*_link | 338691(338753.2) | 339651(339679) | 336688(336690.9) | **336682(336682.9)** |
| web-wik*2009 | 348003(348024.5) | 352396(352492.8) | 346676(346682.5) | **346581(346584)** |
| web-wiki*grow | 117626(117663) | 118917(118967.5) | 116817(116818.7) | **116814(116814.6)** |
| wikip*_link_en | 23995929 (23995935.9) | N/A | 23995924 | **23995924** |

Table 6: Experiment results on Repository benchmark

| benchmark | #total | CC$^2$FS #min | FastMWDS #min | RLS$_0$ #min | ScBppw #min | FastDS #min |
|---|---|---|---|---|---|---|
| UDG | 12 | **12** | **12** | 8 | 2 | 11 |
| T1 | 48 | 43 | 42 | 19 | 7 | **47** |
| DIMACS | 61 | **60** | **60** | 51 | 45 | **60** |
| BHOSLIB | 31 | 29 | 28 | 2 | 0 | **31** |
| SNAP | 22 | 10 | 15 | 7 | 11 | **19** |
| DIMACS10 | 31 | 1 | 9 | 1 | 2 | **28** |
| Repository | 65 | 5 | 17 | 1 | 19 | **63** |

Table 7: Summary results of all benchmarks. We report for each algorithm the number of instances (or families) where it finds the best $D_{min}$ among the algorithms in experiments.

| Benchmark | fix_ratio | ctime_rules | ctime_no_rules |
|---|---|---|---|
| SNAP | 28.59% | 0.98s | 1.07s |
| DIMACS10 | 7.21% | 4.57s | 4.42s |
| Repository | 23.82% | 35.98s | 38.43s |

Table 8: The fix ratio of the inference rules and construction time with and without rules

instances. On average, the best solutions found by FastDS have 1320 and 23732 fewer vertices than the strongest competitor FastMWDS for SNAP and DIMACS10 benchmarks, and have 14150 fewer vertices than the strongest competitor ScBppw for Repository benchmark. Table 7 summarizes the comparison results on all the benchmarks.

## 6.4 Analysis of Underlying Ideas

We calculate the portion of vertices fixed by inference rules among all vertices, and compare the run time of the construction procedure with and without the rules (Table 8). The inference rules are effective on the massive benchmarks with little overhead. FastDS without the rules degrades obviously on the massive benchmarks. It gets worse solutions on half Repository instances. However, the rules have little impact on the standard benchmarks.

We also test a variant of FastDS (denoted as FastDS_g1) that uses the one-goal framework (targeting size of ($k$-1) after finding a $k$ sized solution) to replace our two-goal framework. FastDS finds better solutions than FastDS_g1 on 12, 48 and 31 instances for UDG, T1 and BHOSLIB benchmarks respectively, and worse on none of the instances. For massive benchmarks, FastDS finds better solutions on 11, 25 and 39 instances for SNAP, DIMACS10 and Repository respectively, while worse than FastDS_g1 only on 3 and 9 instances in DIMACS10 and Repository benchmarks respectively.

## 7 Conclusions

We proposed a two-goal local search framework for MinDS and proposed three inference rules. The resulting algorithm FastDS is robust and efficient on standard benchmarks and massive benchmarks, and significantly outperforms state-of-the-art algorithms on massive benchmarks. We would like to study the ideas for other subset problems.

## Acknowledgements

This work is supported by Beijing Academy of Artificial Intelligence (BAAI), Youth Innovation Promotion Association, Chinese Academy of Sciences [No. 2017150], and NSFC Grant 61806050.

## References

[Abu-Khzam *et al.*, 2017] Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-charging dominating set with an FPT subroutine: Further improvements and experimental analysis. In *Proceedings of TAMC 2017*, pages 59–70, 2017.

[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 and Lin, 2016] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of IJCAI 2016*, pages 568–574, 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. *Artif. Intell.*, 175(9-10):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. *J. Artif. Intell. Res.*, 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. *J. Artif. Intell. Res.*, 59:463–494, 2017.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of IJCAI 2015*, pages 747–753, 2015.

[Chalupa, 2018] David Chalupa. An order-based algorithm for minimum dominating set with application in graph mining. *Inf. Sci.*, 426:101–116, 2018.

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

[Fan *et al.*, 2019] Yi Fan, Yongxuan Lai, Chengqian Li, Nan Li, Zongjie Ma, Jun Zhou, Longin Jan Latecki, and Kaile Su. Efficient local search for minimum dominating sets in large graphs. In *Proceedings of DASFAA 2019*, pages 211–228, 2019.

[Gao *et al.*, 2015] Chao Gao, Xin Yao, Thomas Weise, and Jinlong Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.

[Gao *et al.*, 2018] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An exact algorithm for maximum k-plexes in massive graphs. In *Proceedings of IJCAI 2018*, pages 1449–1455, 2018.

[Gast *et al.*, 2015] Mikael Gast, Mathias Hauptmann, and Marek Karpinski. Inapproximability of dominating set on power law graphs. *Theor. Comput. Sci.*, 562:436–452, 2015.

[Hedar and Ismail, 2010] Abdel-Rahman Hedar and Rashad Ismail. Hybrid genetic algorithm for minimum dominating set problem. In *Proceedings of ICCSA 2010*, pages 457–467, 2010.

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

[Karthik C. S. *et al.*, 2018] Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. In *Proceedings of STOC 2018*, pages 1283–1296, 2018.

[Li *et al.*, 2019] Ruizhi Li, Shuli Hu, Huan Liu, Ruiting Li, Dantong Ouyang, and Minghao Yin. Multi-start local search algorithm for the minimum connected dominating set problems. *Mathematics*, 7(12):1173, 2019.

[Lin *et al.*, 2016] Geng Lin, Wenxing Zhu, and M. Montaz Ali. An effective hybrid memetic algorithm for the minimum weight dominating set problem. *IEEE Trans. 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 IJCAI 2017*, pages 517–523, 2017.

[Musliu, 2006] Nysret Musliu. Local search algorithm for unicost set covering problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 302–311. Springer, 2006.

[Potluri and Singh, 2011] Anupama Potluri and Alok Singh. Two hybrid meta-heuristic approaches for minimum dominating set problem. In *Proceedings of SEMCCO 2011*, pages 97–104, 2011.

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

[Raz and Safra, 1997] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of STOC 1997*, pages 475–484, 1997.

[Richter *et al.*, 2007] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proceedings of KI 2007*, pages 412–426, 2007.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of AAAI 2015*, pages 4292–4293, 2015.

[Sanchis, 2002] Laura A. Sanchis. Experimental analysis of heuristic algorithms for the dominating set problem. *Algorithmica*, 33(1):3–18, 2002.

[van Rooij and Bodlaender, 2011] Johan M. M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011.

[Verma *et al.*, 2015] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[Wang *et al.*, 2009] Feng Wang, Erika Camacho, and Kuai Xu. Positive influence dominating set in online social networks. In *Proceedings of COCOA 2009*, pages 313–321, 2009.

[Wang *et al.*, 2017] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *J. Artif. Intell. Res.*, 58:267–295, 2017.

[Wang *et al.*, 2018] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *Proceedings of IJCAI 2018*, pages 1514–1522, 2018.

[Xu *et al.*, 2007] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial intelligence*, 171(8-9):514–534, 2007.