NuCDS: An Efficient Local Search Algorithm for Minimum Connected Dominating Set

Bohan Li[1,2], **Xindi Zhang**[1,2], **Shaowei Cai**[1,2*], **Jinkun Lin**[1], **Yiyuan Wang**[3*] and **Christian Blum**[4]

[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 Computer Science and Information Technology, Northeast Normal University, China
[4]Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Bellaterra, Spain
libohan19@mails.ucas.ac.cn, dezhangxd@163.com, caisw@ios.ac.cn, jkunlin@gmail.com, yiyuanwangjlu@126.com, christian.blum@iiia.csic.es

Abstract

The minimum connected dominating set (MCDS) problem is an important extension of the minimum dominating set problem, with wide applications, especially in wireless networks. Despite its practical importance, there are few works on solving MCDS for massive graphs, mainly due to the complexity of maintaining connectivity. In this paper, we propose two novel ideas, and develop a new local search algorithm for MCDS called NuCDS. First, a hybrid dynamic connectivity maintenance method is designed to switch alternately between a novel fast connectivity maintenance method based on spanning tree and its previous counterpart. Second, we define a new vertex property called *safety* to make the algorithm more considerate when selecting vertices. Experiments show that NuCDS significantly outperforms the state-of-the-art MCDS algorithms on both massive graphs and classic benchmarks.

1 Introduction

Given an undirected connected graph $G = (V, E)$, a set $D \subseteq V$ is called a dominating set if each vertex in V either belongs to D or is adjacent to at least one vertex from D. The minimum dominating set (MDS) problem is to find a dominating set with the minimum number of vertices in the given graph. An important generalization of MDS is the minimum connected dominating set (MCDS) problem, whose goal is to find a minimum size dominating set that forms a connected subgraph in the given graph. An important application of MCDS is generating a virtual backbone in wireless networks such as mobile ad hoc networks [Al-Karaki and Kamal, 2008], wireless sensors networks [Misra and Mandal, 2009] and vehicular ad hoc networks [Chinnasamy *et al.*, 2019]. Applications are also found in other fields. [Milenković *et al.*, 2011] Also, MCDS is equivalent to the maximum leaf spanning tree problem [Fernau *et al.*, 2011].

It is well known that MCDS is NP-hard [Kann, 1992]. Several exact algorithms [Fomin *et al.*, 2008; Simonetti *et al.*, 2011; Fan and Watson, 2012; Gendron *et al.*, 2014] and approximation algorithms [Cheng *et al.*, 2003; Ruan *et al.*, 2004; Khuller and Yang, 2019] have been designed for MCDS. Nevertheless, these algorithms are either too time-consuming or have poor performance in practice, especially in the context of massive graphs.

1.1 Related Work

Because of its NP-hardness, much of the research effort in the past decade concerned with solving MCDS has focused on heuristics with the aim of obtaining a good solution within a reasonable time. Two algorithms called MCDS/SA and MCDS/TS based on simulated annealing and tabu search were proposed [Morgan and Grout, 2007]. Hedar and Ismail [2012] designed a simulated annealing algorithm with stochastic local search for MCDS. Later, Jovanovic and Tuba [2013] designed an ant colony optimization algorithm with a so-called pheromone correction strategy. A greedy random adaptive search procedure that incorporated a local search procedure based on a greedy function and tabu search was described in [Li *et al.*, 2017]. Wu et al. [2017] used a restricted swap-based neighborhood to improve the tabu search procedure, resulting in the RNS-TS algorithm. Two meta-heuristics based on genetic algorithms and simulated annealing were designed to solve MCDS [Hedar *et al.*, 2019]. Li et al. [2019] presented a multi-start local search algorithm called MSLS based on three mechanisms including a vertex score, configuration checking, and vertex flipping. Finally, a meta-heuristic algorithm called ACO-RVNS [Bouamama *et al.*, 2019] was proposed, based on ant colony optimization and reduced variable neighborhood search. Experiments show that, for classic graphs with fewer than 5000 vertices, RNS-TS, MSLS, and ACO-RVNS obtain similar state-of-the-art performance.

1.2 Contributions and Paper Organization

Although previous MCDS algorithms have made progress in solving classic graphs, these algorithms still cannot handle massive graphs with millions of vertices. In this work, we focus on solving MCDS on massive graphs, and develop a local search algorithm called NuCDS. The algorithm is based

*Corresponding author

on two novel ideas that are important for dealing with the connectivity constraint and for vertex selection in local search respectively.

The first idea is a hybrid dynamic connectivity maintenance method called HDC. Previous connectivity maintenance methods make the algorithm explore rather large parts of the search space, but they become futile when faced with very large search space. To overcome this issue, we propose a novel tree-based connectivity maintenance (TBC for short) method, which is inspired by spanning trees. Compared to previous methods, the TBC method has a very low complexity, while limiting the algorithm to explore a relatively small part of the search space. To obtain a balance between diversification and intensification, we design the HDC method to alternately switch between previous connectivity maintenance methods and TBC during the search process.

The second idea is a safety-based vertex selection heuristic. We define a new vertex property called *safety*, which takes into account the differences among dominated vertices. We propose a novel vertex selection rule based on different scoring functions, one of which is safety-based. This is in contrast to previous MCDS algorithms which only use one scoring function. In some sense, the safety-based scoring function can be used as a supplement of traditional scoring functions.

Extensive experiments are carried out to evaluate NuCDS on classical benchmarks used in previous literature and on massive graphs from real-world applications. Experimental results indicate that NuCDS performs better than four state-of-the-art MCDS heuristic algorithms for most instances.

Section 2 introduces preliminary knowledge. In Sections 3 and 4, we describe the hybrid dynamic connectivity maintenance method and the safety-based vertex selection heuristic. Section 5 presents the NuCDS algorithm. Experiments are presented in Section 6, and Section 7 concludes the paper.

2 Preliminaries

An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set E. For an edge $e = \{u, v\}$, vertices u and v are the endpoints of the edge. For a vertex v, its neighborhood is $N_G(v) = \{u \in V | \{u, v\} \in E\}$, and its closed neighborhood is $N_G[v] = N_G(v) \cup \{v\}$. The degree of a vertex v, denoted as $d_G(v)$, is defined as $|N_G(v)|$, and Δ_G is the maximum number of $d_G(v)$ for $\forall v \in V$. Given a vertex set $S \subseteq V$, $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$ and $N_G[S] = \bigcup_{v \in S} N_G[v]$ stands for the neighborhood and closed neighborhood of S, respectively. $G[S] = (V_S, E_S)$ is a subgraph in G induced by S such that $V_S = S$ and E_S consists of all the edges in E whose endpoints are in S.

An undirected graph $G = (V, E)$ is connected when it has at least one vertex and there is a path between every pair of vertices. For convenience, we use G to denote an undirected connected graph in the following parts.

Definition 1. *Given an undirected connected graph G, a vertex in G is an articulation vertex iff removing it, together with the edges connected to it, disconnects the graph. The articulation vertex set of G is denoted as* $art(G)$.

Given a vertex set $D \subseteq V$, a vertex $v \in V$ is dominated by D if $v \in N[D]$, and is non-dominated otherwise. We use $D \subseteq V$ to denote a candidate solution. If $G[D]$ is connected and D dominates all vertices, then D is a connected dominating set (CDS). For a given graph G, the aim of the minimum connected dominating set (MCDS) problem is to find the connected dominating set D with the smallest size.

3 The HDC Method

In order to solve the performance bottleneck problem caused by the connectivity constraint of MCDS, we introduce a hybrid dynamic connectivity maintenance method (HDC for short). After reviewing two main previous connectivity maintenance methods, we propose our novel tree-based connectivity maintenance method, and finally give the HDC method. For convenience of discussions on complexity, we will use notations $m = |V_D|$ and $n = |E_D|$, where $G[D] = (V_D, E_D)$ is the induced subgraph of current candidate solution D.

3.1 Previous Connectivity Maintenance Methods

Before introducing HDC, we review two previous methods to handle the connectivity constraint, namely the substraction-based and addition-based methods.

The substraction-based method is used by previous state-of-the-art MCDS algorithms such as RNS-TS [Wu *et al.*, 2017]. In order to keep the connectivity of candidate solution D, during each iteration of local search, the algorithm maintains the candidate removal vertex set, defined as $candRemoval(D)$=$D \setminus art(G[D])$. The traditional approach to computing $art(G[D])$ is called Tarjan's algorithm [Hopcroft and Tarjan, 1973]. It works as follow: depth first search (DFS for short) is used to determine whether the child vertex of a vertex u can access the ancestor vertex of u without passing through u.[1] If so, then u is not an articulation vertex, and otherwise it is an articulation vertex. The complexity of the substraction-based method is $O(m + n)$.

The addition-based method has two versions. The first version, used in ACO-RVNS [Bouamama *et al.*, 2019] and in MCDS/TS [Morgan and Grout, 2007], works as follow: during each iteration, the algorithm starts from an empty candidate solution D, and iteratively adds a vertex from $N_G(D)$ to D until D becomes a feasible solution. The second version, used in GRASP [Li *et al.*, 2017] and MSLS [Li *et al.*, 2019], works as follow: the algorithm allows removing articulation vertices of D, so D may become disconnected. Thus, before vertex u is selected to be added, DFS is used to calculate the number of connected subgraphs of D that $v \in N(u)$ belongs to, and the vertex with the largest number is preferred. The complexity of both versions is $O(m)$.

The complexity of the above two methods is at least $O(m)$. This indicates a high commutation time when applied to massive graphs, hindering the performance on masive graphs.

3.2 Tree-based Connectivity Maintenance Method

To lower the time complexity of connectivity maintenance, we present a novel tree-based connectivity maintenance (TBC for short) method, which is inspired by spanning trees.

[1] The child vertex of u is the vertex visited directly after u, while father and ancestor vertices of u are the vertices visited directly and indirectly before u by DFS.

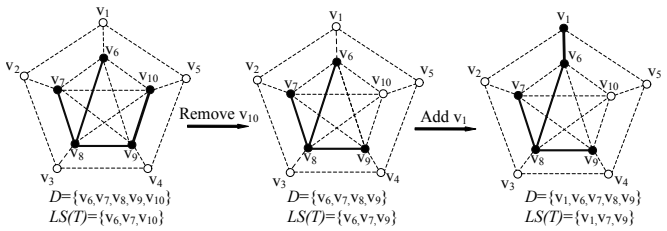

Figure 1: An example of the TBC method. (The solid nodes denote those vertices in the candidate solution D, while the solid edges compose the spanning tree T of $G[D]$.)

For this purpose, we first introduce the definitions of a spanning tree and a leaf vertex. Given a connected graph $G = (V, E)$, a spanning tree $T = (V', E')$ is defined as a connected subgraph of G with $V' = V$, $E' \subseteq E$, and without any cycles. Given a spanning tree $T = (V', E')$, a vertex $v \in V'$ is called a leaf vertex if $d_T(v) = 1$ and the leaf set is defined as $LS(T) = \{v | d_T(v) = 1, v \in V'\}$.

Proposition 1. *Given a spanning tree $T = (V', E')$ and its corresponding leaf set $LS(T)$ of graph $G = (V, E)$, $LS(T)$ is a subset of $V \setminus art(G)$.*

Proof: For any vertex $v \in LS(T)$, after removing v and relevant edges through v from T, the remaining graph is still a spanning tree of $G[V \setminus \{v\}]$. So $G[V \setminus \{v\}]$ remains connected, which means that v is not an articulation vertex of G, inducing that $v \in V \setminus art(G)$. Thus, we can conclude that $LS(T) \subseteq V \setminus art(G)$. □

Based on the definition and proposition above, we describe TBC as follows. Given a candidate solution D, we maintain a spanning tree T of $G[D]$ and its corresponding leaf set $LS(T)$ during the search process. Each vertex $v \in LS(T)$ is allowed to be removed from D. All other vertices are forbidden to be removed. In contrast to the previous substraction-based method, given a current solution D, TBC calculates an approximate $candRemoval(D) = LS(T)$, which is a subset of $D \setminus art(G[D])$. In order to dynamically update $candRemoval(D)$, three updating rules of TBC are described as follows.

Construction Rule: At the initialization phase, both T and $LS(T)$ with respect to $G[D]$ are constructed according to the initial candidate solution D by using breadth first search.

Removing Rule: When vertex v is selected to be removed, its father vertex u needs to be found. If $d_T(u) = 2$, meaning that u becomes a leaf vertex after removing v, $LS(T) = LS(T) \setminus \{v\} \cup \{u\}$. Otherwise, $LS(T) = LS(T) \setminus \{v\}$.

Adding Rule: When vertex v is added to the candidate solution D, one vertex $u \in N(v) \cap T$ needs to be selected as the father vertex of v. In order to render larger $candRemoval(D)$, we prefer not to select a leaf vertex in $LS(T)$, and thus we pick the vertex u with the maximum $d_T(u)$ among $\forall u \in N(v) \cap T$. If u is a leaf vertex, $LS(T) = LS(T) \setminus \{u\} \cup \{v\}$, else $LS(T) = LS(T) \cup \{v\}$. Lastly, v is set as the child of u in T.

The complexity of the construction rule is $O(m + n)$. When adding or removing a vertex v based on the removing and adding rules, $N(v)$ has to be searched to update T and $LS(T)$, which has a complexity of $O(\Delta_G)$. To make the TBC method more comprehensive, we present its example in Figure 1 regarding to removing v_{10} and then adding v_1.

Algorithm 1: the HDC heuristic

Input: the current solution D
Output: candidate removal vertex set $candRemoval(D)$

```
if stepNoimp > NoImpr || stepOneCon > MaxNoImpr then
    if curMethod==SUB then
        curMethod :=TBC;
        construct a spanning tree T based on
          Construction Rule;
    else curMethod :=SUB ;
    update NoImpr by some tricks;
    stepOneCon := 0, stepNoimp := 0;
if curMethod==TBC then
    candRemovel(D) := LS(T);
else
    compute art(G[D]) based on Tarjan's algorithm;
    candRemovel(D) := D \ art(G[D])
return candRemoval(D);
```

3.3 The Main Framework of HDC Method

For large graphs, TBC is substantially faster than previous connectivity maintenance methods. However, since TBC does not consider all candidate removal vertices, it may miss some high-quality options. As an extreme example, a candidate solution D with a vertex $v \in D$ such that $D \setminus \{v\}$ is also a connected dominating set. TBC, however, does not consider this option if v is not a leaf vertex of the current spanning tree. In contrast, the substraction-based method using Tarjan's algorithm potentially considers more options for removal because it accurately determines the complete candidate removal vertex set, but its complexity is higher.

In order to take profit from the respective advantages of both methods, a balance must be achieved between the complexity and the accuracy of determining the candidate removal vertex set. We propose a heuristic called hybrid dynamic connectivity maintenance method (HDC) to switch between the two methods. We first define three parameters of the heuristic: 1) parameter $NoImpr$ denotes the maximum number of steps without improving the candidate solution, where $NoImpr \in [MinNoImpr, MaxNoImpr]$; 2) parameter MinNoImpr denotes the minimum value of NoImpr, and it is also adopted as the absolute value of the change on NoImpr in each step; 3) parameter $MaxNoImpr$ denotes the maximum number of steps using the current connectivity maintenance method.

Considering the difference of complexity between the two connectivity maintenance methods, two sets of parameters are respectively set for SUB and TBC.

A formal description of the HDC heuristic is given in Algorithm 1. $stepNoImpr$ and $stepOneCon$ denote the number of non-improving steps and the number of steps adopting one

current connectivity maintenance method, respectively.

When the candidate solution has not been improved for $Noimpr$ iterations or the current method has been used for $MaxNoImpr$ iterations (line 1), the algorithm switches from the current connectivity maintenance method to the other one (lines 2–5). In particular, after switching to TBC, a spanning tree T of D and its corresponding leaf set $LS(T)$ need to be constructed (line 4). After the switching operation, $NoImpr$ needs to be updated (line 6). Specifically, if $stepNoimp > NoImpr$, then the algorithm increases $NoImpr$ by $MinNoImpr$ so that it can search more exhaustively for better candidate solutions. Otherwise, if $stepOneCon > MaxNoImpr$, the algorithm decreases $NoImpr$ by $MinNoImpr$ in order to accelerate the search. Both $stepNoimpr$ and $stepOneCon$ should be reset to 0 (line 7). Lastly, the algorithm uses the selected connectivity maintenance method to calculate the candidate removal vertex set $candRemoval(D)$ and returns it (lines 8–12).

4 Safety-based Vertex Selection Heuristic

In this section, we introduce a novel vertex selection heuristic by considering the *safety* of vertices. Before proposing our selection rule, we introduce a previous scoring function.

4.1 Scoring Functions

Local search algorithms typically use scoring functions to choose an operation to execute in each step. In the context of the MCDS problem, a scoring function is used to choose a vertex $u \in D$ for removal and a vertex $v \in N(D)$ for addition. Our algorithm adopts the frequency based scoring function [Wang *et al.*, 2017] for this purpose. For each vertex $v \in V$, the frequency of v is denoted as $freq[v]$. It works as follow: 1) before the local search process, $freq[v] = 1$ for $\forall v \in V$; 2) at the end of each iteration of local search, $freq[v] = freq[v]+1$ for each non-dominated vertex $v \in V$.

The scoring function, denoted as $score$, is calculated as follow: If $u \in D$, $score(u) = -\sum_{v \in C_1} freq[v]$, and otherwise $score(u) = \sum_{v \in C_2} freq[v]$, where C_1 is the set of dominated vertices that would become non-dominated by removing u from D and C_2 is the set of non-dominated vertices that would become dominated by adding u to D.

4.2 The Safety-based Selection Rule

In addition to considering the score value, we also observe that some dominated vertices are more "endangered" than others, in the sense that they may become non-dominated more easily. An extreme case is a vertex that is dominated just by one vertex (either by itself or by one of its neighbors). Such vertices may become non-dominated due to an exchanging step. Based on this consideration, we define the *safety* of vertices, taking into account the differences among dominated vertices. We first give the definition of the domination degree.

Definition 2. *Given a connected graph $G = (V, E)$ and a candidate solution D, the domination degree of vertex v is defined as $dd_G(v) = |N_G[v] \cap D|$, for each vertex $v \in V$.*

This means that a vertex v with $dd_G(v) = d$ is dominated by d vertices. Thus, the larger the domination degree, the safer the corresponding vertex. Thus, we define the *safety* of a vertex v, denoted as $sf(v)$, which measures the accumulative effectiveness of the domination degree. For each $v \in V$, $sf(v) = -dd_G(v)$ if $v \in D$, or $sf(v) = dd_G(v)$ if $v \notin D$.

Different from $score$, the *safety* of a vertex measures the domination degree by removing or adding this vertex. For large graphs, ties w.r.t. $score$ frequently occur during vertex selection, that is, more than one vertex has the highest $score$. In such a situation, we can use the *safety* as a secondary criterion for further measurement. By doing this, the local search process can be more considerate in vertex selection. In addition, considering the property age [2] is usually used to break ties for the sake of diversification, we propose to use *safety* and age separately as the secondary criterion for vertex selection. The safety-based vertex selection heuristic in our algorithm is described as follows.

Selection Rule: Select the vertex v with the greatest $score$, breaking ties by preferring the one with the greatest $sf(v)$. Further ties are broken by picking the one with greatest age if more than one vertex has the greatest $score$ and $sf(v)$ values.

This idea is inspired by the concept of subscore for the SAT problem [Cai and Su, 2013] which considers the satisfaction degree of clauses. Moreover, as far as we know, it is the first time that the definition of *safety* is considered to be applied into solving a graph optimization problem.

5 NuCDS Algorithm

Based on HDC and the safety-based vertex selection heuristic, we develop a local search algorithm for the MCDS problem named NuCDS. To avoid visiting previous candidate solutions, we use the CC^2 strategy [Wang *et al.*, 2017] in the adding process, and use the tabu strategy [Glover and Laguna, 1998] recording the adding operations to prevent removing a just added vertex for the next tt iterations. In our work, $tt = 5 + rand(10)$. The pseudo code of NuCDS is shown in Algorithm 2.

In the beginning, NuCDS initializes $score$ and sf (line 1), and sets $curMethod$ = TBC which means that the TBC method is selected as the initial connectivity maintenance method (line 2). The non-improvement step $stepNoimpr$ and the number of steps used by the current connectivity maintenance method $stepOneCon$ are both set to 0 (line 2). Then, the algorithm constructs the initial candidate solution D using the approximation algorithm in [Khuller and Yang, 2019] (line 3). According to the construction rule, the algorithm builds a spanning tree T of D (line 4), and then the candidate removal vertex set $candRemoval(D)$ is initialized to $LS(T)$ (line 5).

During the local search procedure (lines 6–19), the algorithm first uses the HDC heuristic to update $candRemoval(D)$ (line 7). If D is a feasible solution, which means that the algorithm has already found a connected dominating set of size $|D|$, D^* is updated to D and $stepNoimpr$ is set to 0 (line 9). Then, the algorithm continues to find a solution of size $(|D| - 1)$, i.e., by removing a vertex v from D using the BMS heuristic [Cai, 2015] (lines 10–11).

[2] The age of a vertex v is the number of steps that have occurred since v last changed its state.

Specifically, the algorithm randomly selects k vertices from $candRemoval(D)$ and picks from this set the removal vertex u according to the selection rule (in our work, k is set to 45). If the algorithm selects the TBC method as the current connectivity maintenance method, the corresponding spanning tree T and its leaf set $LS(T)$ should be updated based on the removing rule (line 12).

If D is still an infeasible solution, the algorithm tries to exchange two vertices (lines 14–17), i.e., removing a non-tabu vertex $u \in candRemoval(D)$ from D via the BMS heuristic and the selection rule, and then adding a vertex $v \in N(D) \cap N(G \setminus N[D])$ to D via the CC^2 strategy and the selection rule, where $G \setminus N[D]$ is the non-dominated vertex set and $N(D)$ contains vertices maintaining connectivity. Finally, the algorithm needs to update T, $LS(T)$, $stepOneCon$, and $stepNoimpr$ accordingly (lines 18–19). When the time limit is reached, the best solution found (D^*) will be returned. The complexity of each iteration in the local search process is O($|D|$) or O(Δ_G) when using SUB or TBC methods respectively.

Moreover, we make use of a trick for adopting the HDC heuristic. If $|D| < 100$ (that is, if the complexity of the SUB method is acceptable), using TBC becomes trivial and thus we only use the SUB method under such a situation.

Algorithm 2: the NuCDS algorithm

Input: An undirected graph $G = (V, E)$, the *cutoff* time
Output: An obtained best connected dominating set D^*

1 initialize $score(v)$, $sf(v)$, for $\forall v \in V$;
2 $curMethod$:=TBC, $stepNoimpr := stepOneCon := 0$;
3 $D := ConstructCDS(G)$, $D^* := D$;
4 build a spanning tree T according to **Construction Rule**;
5 $candRemoval(D) := LS(T)$;
6 **while** *elapsed time* $<$ *cutoff* **do**
7 $candRemoval(D) := HDC(D)$;
8 **if** *D is a connected dominating set* **then**
9 $D^* := D$, $stepNoimpr := 0$;
10 select a vertex $u \in candRemoval(D)$ by BMS according to **Selection Rule**;
11 $D := D \setminus \{u\}$;
12 **if** $curMethod$==*TBC* **then** update T and $LS(T)$ according to **Removing Rule**;
13 continue;
14 select a vertex $u \in candRemoval(D)$ by BMS according to **Selection Rule**;
15 $D := D \setminus \{u\}$;
16 choose a vertex $v \in N(D) \cap N(G \setminus N[D])$ according to **Selection Rule**;
17 $D := D \cup \{v\}$;
18 **if** $curMethod$==*TBC* **then** update T and $LS(T)$ according to **Removing and Adding Rules**;
19 $stepNoimpr$++, $stepOneCon$++;
20 **return** D^*;

6 Experimental Evaluation

We carry out extensive experiments to evaluate the performance of NuCDS. We compare NuCDS with four state-of-the-art heuristic algorithms: MSLS [Li *et al.*, 2019], ACO-RVNS [Bouamama *et al.*, 2019], ACO-efficient[3] and RNS-TS [Wu *et al.*, 2017]. The codes of all competitors were kindly provided by the authors. RNS-TS was implemented in Java while NuCDS and the other competitors were implemented in C++ and complied by g++ with '-O3'. Data structure was modified to handle massive graph. Moreover, ACO-efficient was modified by the author of ACO, specialized for massive graphs. All experiments were run on a server with Intel Xeon E7-4830 v4 2.00GHz with 256GB RAM under Ubuntu 16.04.5.

For our experiments, we adopt several popular benchmarks which are divided into two groups, including classical benchmarks and massive graphs. First we considered 121 classical benchmarks, which are mainly divided into five groups: the instances of the maximum leaf spanning tree problem [Lucena *et al.*, 2010], bus power flow test cases[4] , random geometric graphs [Jovanovic and Tuba, 2013], and random graphs from [Erdem *et al.*, 2009] and from [Bouamama *et al.*, 2019]. All these classical instances have already been used for testing the effectiveness of the competitors [2019].

We also evaluated NuCDS on massive graphs, including massive real-world graphs from the Network Data Repository (NDR) [Rossi and Ahmed, 2015] and Stanford Large Network Dataset Collection (SNAP)[5], as well as large instances from the 10th DIMACS implementation challenge (DIMACS10)[6]. As in previous work [Lin *et al.*, 2017], we only report the results on graphs from the SNAP and DIMACS10 benchmarks with at least 30,000 vertices (with the exception of a few cases with fewer vertices). Moreover, due to space limitations, we do not report the results on graphs from the NDR benchmark with fewer than 100,000 vertices or fewer than 1,000,000 edges. Hence, we picked 22, 31 and 65 instances from SNAP, DIMACS10, and NDR benchmarks respectively, leading to totally 118 massive graphs.

All algorithms were executed 10 times on each instance independently. The cutoff time was set to 1000 seconds for the classical benchmarks, and 3600 seconds for massive graphs. We report the the best size (min) and the average size (avg) of the solutions found. If an algorithm fails to find a solution within the time limit, this is indicated by 'N/A'.

In preliminary experiments we found that NuCDS is not very sensitive to parameter settings. According to these experiemnts we set $MinNoImpr$=100 when using SUB, and $MinNoimpr$=100000 when using TBC. $NoImpr$ and $MaxNoImpr$ were set as $2\times MinNoImpr$ and $10\times MinNoImpr$, respectively.

[3]We contacted the author of ACO-RVNS to get an improved version called ACO-efficient, which performs better than ACO-RVNS on most large instances.

[4]http://labs.ece.uw.edu/pstca

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

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

Instance	NuCDS min(avg)	MSLS min(avg)	ACO-efficient min(avg)	ACO-RVNS min(avg)	RNS-TS min(avg)
v150_d10	**14(14)**	**14**(14.6)	**14**(14.6)	**14**(14.5)	**14(14)**
ieee_300_bus	130(130.9)	**129**(129.2)	**129**(129.2)	**129(129)**	**129(129)**
n1000_200_r100	**38(38)**	39(39.6)	39(39.1)	**38**(38.2)	**38**(38.5)
n1000_200_r110	**34(34)**	**34**(34.1)	**34**(34.1)	**34(34)**	**34(34)**
n1000_200_r120	**29(29)**	**29**(30.3)	30(30.2)	**29(29)**	**29(29)**
n1000_200_r130	**26(26)**	**26**(26.8)	**26**(26.7)	**26(26)**	**26(26)**
n1000_200_r140	**23(23)**	**23**(23.1)	**23**(23.4)	**23(23)**	**23(23)**
n1000_200_r150	**21(21)**	**21**(21.1)	**21**(21.1)	**21(21)**	**21(21)**
n1000_200_r160	**19(19)**	**19**(19.6)	36(36.3)	**19(19)**	**19**(19.1)
n1500_250_r130	**49(49)**	**49**(50)	**49**(49.7)	**49**(49.1)	**49(49)**
n1500_250_r140	**43(43.7)**	44(44.3)	44(44)	**43**(43.9)	**43**(43.9)
n1500_250_r150	**40(40)**	41(41.6)	41(41.6)	**40**(40.7)	**40**(40.1)
n1500_250_r160	**36(36)**	37(37.7)	**36**(36.3)	**36(36)**	**36(36)**
n2000_300_r200	**41(41.5)**	43(43.1)	42(42.5)	42(42.1)	**41**(41.6)
n2000_300_r210	**38(38)**	**38**(38.6)	**38**(38.5)	**38(38)**	**38(38)**
n2000_300_r220	**35(35)**	36(36.3)	**35**(36)	**35**(35.1)	**35(35)**
n2000_300_r230	**33(33)**	34(34.8)	34(34.5)	**33(33)**	**33**(33.2)
n2500_350_r200	**60(60)**	61(61.3)	61(62.3)	**60**(60.7)	**60**(60.3)
n2500_350_r210	**54(54.9)**	56(57.0)	57(58.2)	55(56.1)	55(56.0)
n2500_350_r220	**51(51.1)**	52(54.3)	54(54.7)	**51**(52.7)	**51**(51.4)
n2500_350_r230	49(49.1)	50(50.6)	50(51.7)	**48**(49.5)	49(49.0)
n3000_400_r210	**74(74)**	**74**(76.3)	76(76.6)	**74**(75.5)	75(75.1)
n3000_400_r220	**70(70)**	71(71.9)	71(71.6)	**70**(70.8)	**70**(70.3)
n3000_400_r230	**64(64.8)**	66(67.0)	65(67.1)	65(65.9)	65(65.1)
n3000_400_r240	**60(60.9)**	61(62.5)	62(62.8)	61(61.7)	61(61.2)
n600_100_r110	**14(14)**	**14**(14.5)	**14**(14.6)	**14(14)**	**14(14)**
n700_200_r100	**22(22)**	**22**(22.5)	**22**(22.9)	**22(22)**	**22(22)**
n700_200_r110	**20(20)**	**20(20)**	**20**(20.5)	**20(20)**	**20(20)**
n700_200_r120	**17(17)**	**17(17)**	**17**(17.8)	**17(17)**	**17(17)**
n700_200_r70	**38(38.1)**	39(39.6)	39(39)	**38(38.1)**	**38**(38.5)
n700_200_r80	**32(32)**	33(33)	33(33)	**32(32)**	**32(32)**
n1000_ep0007	**179(179)**	191(194)	187(189)	185(186.6)	189(190.7)
n1000_ep0014	**98(98)**	105(105.5)	104(105.3)	101(103.1)	103(105.3)
n1000_ep0028	**59(59.9)**	62(62.5)	62(62.8)	61(62.8)	63(63.6)
n1000_ep0056	**37(37)**	**37(37)**	**37**(37.8)	**37**(37.8)	38(38.2)
n1000_ep0112	**22(22)**	**22(22)**	**22(22)**	**22(22)**	**22**(22.1)
n1000_ep0224	**12**(12.5)	**12(12)**	**12**(12.7)	**12(12)**	**12**(12.2)
n1000_r0048	**275**(277.3)	276(276.5)	**275(275.2)**	**275**(275.8)	280(283.7)
n1000_r0070	**125**(126.7)	**125(125)**	128(129.3)	127(128.4)	132(134.4)
n1000_r0100	**61(61.8)**	62(62.4)	64(65.4)	64(65.4)	65(66.1)
n1000_r0140	**32(32.3)**	33(33)	33(33.3)	**32**(32.9)	34(34.6)
n1000_r0207	**15(15.1)**	16(16)	16(16.2)	**15**(15.7)	16(16.6)
n1000_r0308	**7(7)**	**7(7)**	**7(7)**	**7(7)**	**7**(7.7)
n5000_ep0007	**264(265.6)**	277(277.4)	277(278.2)	277(278.2)	392(424.8)
n5000_ep0014	163(164.2)	**161(162.1)**	164(164.2)	165(166.2)	206(220)
n5000_ep0028	**94(94)**	95(95.1)	96(96)	95(95.8)	121(25.8)
n5000_ep0056	56(56)	**55(55)**	**55**(55.6)	56(56.1)	76(100)
n5000_ep0112	32(32)	**31(31.4)**	**31**(31.9)	**31**(31.7)	2707(2934.0)
n5000_ep0224	18(18)	**17(17)**	**17(17)**	**17**(17.1)	3888(4045)
n5000_r0048	**268(270)**	275(275.8)	280(284.6)	280(284.6)	319(336.1)
n5000_r0070	**127(128.4)**	133(133.4)	132(136.4)	132(136.2)	147(150.3)
n5000_r0100	**63(64.4)**	68(68.42)	70(71.5)	72(72.5)	74(78.2)
n5000_r0140	**33(33.2)**	36(36.71)	37(37.4)	36(36.9)	39(41)
n5000_r0207	**16(16)**	17(17)	18(18.9)	16(16.8)	2099(2797.3)
n5000_r0308	**7(7.6)**	8(8.28)	9(9.5)	8(8)	3652(3876)

Table 1: Results of NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on classical benchmarks.

Instance	NuCDS min(avg)	MSLS min(avg)	ACO-efficient min(avg)	ACO min	RNS min
Amazon0302	**45818(46419.8)**	48225(48625.6)	47969(47969)	N/A	N/A
Amazon0312	**52181(52696.7)**	N/A	N/A	N/A	N/A
Amazon0505	**54045(54544.5)**	N/A	N/A	N/A	N/A
Amazon0601	**48001(48487.6)**	N/A	N/A	N/A	N/A
Cit-HepPh	**3268(3281.4)**	3395(3407.2)	3363(3380.4)	3417	31018
Cit-HepTh	**3203(3220.3)**	3289(3303.3)	3264(3279.6)	3320	24036
cit-Patents	**734156(734603)**	N/A	N/A	N/A	N/A
Email-EuAll	2371(2371)	**2368(2368.6)**	**2368**(2369.3)	2373	224163
p2p-Gnutella04	**2268(2270.7)**	2294(2295.6)	2270(2271.5)	2279	6526
p2p-Gnutella24	5471(5471.2)	5475(5477.1)	**5469(5469.9)**	5470	23819
p2p-Gnutella25	4558(4558.2)	4561(4562.7)	**4556(4556.6)**	4557	19467
p2p-Gnutella30	**7229(7229.2)**	7238(7240.5)	**7229**(7231.7)	7231	34578
p2p-Gnutella31	**12675(12675.1)**	12677(12678.7)	12677(12679.4)	12683	61061
Slashdot0811	**14990(14992.3)**	14993(14994.5)	14994(14996.3)	N/A	76148
Slashdot0902	**16160(16163)**	16170(16173.81)	16169(16171.6)	N/A	81006
soc-Epinions1	**16667(16669.5)**	16668(16668.8)	16671(17762.3)	N/A	74677
web-BerkStan	**30961(31085.1)**	31308(31332.3)	N/A	N/A	N/A
web-Google	**86954(86973.8)**	N/A	N/A	N/A	N/A
web-NotreDame	25665(25679.7)	**25406(25462)**	25686(25698)	25641	N/A
web-Stanford	11586(11588)	**11541(11600.3)**	11706(11715.6)	11742	N/A
WikiTalk	**35038(35038.2)**	N/A	N/A	N/A	N/A
Wiki-Vote	**1101(1101)**	**1101(1101)**	**1101(1101)**	**1101**	3093
333SP	**1233411(1233510.8)**	N/A	N/A	N/A	N/A
as-22july06	**2059(2059)**	**2059(2059)**	**2059(2059)**	**2059**	19917
audikw1	**11055(12776.2)**	N/A	11864(11864)	N/A	N/A
belgium	**1201658(1201705.1)**	N/A	N/A	N/A	N/A
cage15	**679723(681679.2)**	N/A	N/A	N/A	N/A
caida*Level	**47993(48085.8)**	48361(48707.8)	48422(48422)	N/A	N/A
citationCiteseer	**49675(49870)**	50488(51109)	50434(50434)	N/A	N/A
cnr-2000	**24769(24779.8)**	25177(25190.5)	24880(24880)	N/A	N/A
coAu*Citeseer	**38124(38202.9)**	38203(38453.8)	38265(38265)	N/A	N/A
coAuthorsDBLP	**48809(48881.9)**	49103(49141.5)	48963(48963)	N/A	N/A
cond-mat-2005	**5125(5129.2)**	5165(5169.3)	5168(5171.9)	5175	33782
coPapersCiteseer	**34790(35282.3)**	N/A	N/A	N/A	N/A
coPapersDBLP	**46433(47181.5)**	N/A	N/A	N/A	N/A
ecology1	**400417(404621.5)**	N/A	N/A	N/A	N/A
eu-2005	**35143(38694)**	N/A	N/A	N/A	N/A
G_n_pin_pout	**14278(14474)**	15124(15154.4)	15040(15065.3)	15351	N/A
in-2004	**86846(86859.11)**	N/A	N/A	N/A	N/A
kron*logn16	**3886(3889.125)**	3888(3889.2)	3887(3888.4)	N/A	53980
ldoor	**34128(35520.2)**	N/A	N/A	N/A	N/A
luxembourg	**101556(101570.8)**	102284(102305.8)	101968(101968)	N/A	113387
prefer*chment	**8551(8586)**	9100(9116.3)	9022(9048.6)	9155	N/A
rgg_n_2_17_s0	**23158(23285.4)**	23631(23662.3)	23621(23621)	N/A	N/A
rgg_n_2_19_s0	**84112(84843.1)**	N/A	N/A	N/A	N/A
rgg_n_2_20_s0	**162669(163039.11)**	N/A	N/A	N/A	N/A
rgg_n_2_21_s0	**315925(317382)**	N/A	N/A	N/A	N/A
rgg_n_2_22_s0	**620349(621002.11)**	N/A	N/A	N/A	N/A
rgg_n_2_23_s0	**1199405(1199630.66)**	N/A	N/A	N/A	N/A
rgg_n_2_24_s0	**2325059(2325227.55)**	N/A	N/A	N/A	N/A
smallworld	**14363(14801.3)**	15354(15374.6)	15609(15645.7)	15646	N/A
uk-2002	**1184695(1184712)**	N/A	N/A	N/A	N/A
wave	**16343(16660.7)**	17504(17547.8)	17792(17815)	N/A	N/A

Table 2: Results on SNAP and DIMACS10 benchmarks. To save space, we denote ACO-RVNS and RNS-TS as ACO and RNS.

6.1 Results on Classical Benchmarks

Most instances of classical benchmarks are so easy that all algorithms obtain the same solution quality very quickly. We ignore these instances, but we report the average run time when all algorithms obtain the same minimal and average values in Figure 2, which shows the effectiveness of NuCDS.

We present the results for the remaining 55 classical instances in Table 1. NuCDS obtains better solutions than MSLS, ACO-efficient, ACO-RVNS and RNS-TS for 31, 33, 19 and 25 instances, respectively, while NuCDS fails to match the solutions obtained by some competitors only on 6 instances. For the instances where NuCDS generates a solution with the same value as the best competitor, NuCDS obtains better average size for 8 instances with 3 exceptions.

6.2 Results on Massive Graphs

Tables 2 and 3 show the results of NuCDS and all competitors on massive graphs. Once again, the results of NuCDS are significantly better than those of all competitors for most of the considered graphs. This is with the exception of only five graphs. Moreover, NuCDS can solve all these 118 instances within the time limit, while MSLS, ACO-efficient, ACO-RVNS and RNS-TS can only solve 45, 48, 18, and 26 instances, respectively. Among all those solvable instances, the best solution values obtained by NuCDS are on average 1.79%, 7.38%, 0.51%, and 85.77% smaller than these found by MSLS, ACO-efficient, ACO-RVNS, and RNS-TS, respectively. The excellent results of NuCDS on massive graphs can mainly be attributed to the power of the HDC heuristic.

Two reasons account for why competitors cannot output feasible solutions on some instances: 1) The construction processes of MSLS, ACO and ACO-efficient have a complexity of $O(|V|^2)$. They are essential parts of multi-restart and ACO framework, so we reserve them; 2) The complexity of each step is $O(|V|^2)$ in RNS, often trapping in the first iteration. Because $|V|$ of some instances reaches to 10^7, competitors fail to output feasible solution.

Instance	NuCDS min(avg)	MSLS min	ACO_e min	ACO min	RNS-TS min
bn-human*1-bg	**4400(4418.3)**	4683	4692	NA	NA
bn-human*2-bg	**3450(3464.7)**	3836	3704	NA	NA
ca-coauthors-dblp	**46399(46428.1)**	NA	NA	NA	NA
ca-dblp-2012	**51009(51021.3)**	51072	51193	NA	316258
ca-hollywood-2009	**53090(53199)**	NA	NA	NA	NA
channel*b050	**653930(654224.11)**	NA	NA	NA	NA
dbpedia-link	**1561222(1561222)**	NA	NA	NA	NA
delaunayn22	**1188366(1188453.2)**	NA	NA	NA	NA
delaunayn23	**2367669(2368135.5)**	NA	NA	NA	NA
delaunayn24	**4792433(4792748.7)**	NA	NA	NA	NA
friendster	**659165(659324.8)**	NA	NA	NA	NA
hugebubbles-00020	**12536446(12566644.3)**	NA	NA	NA	NA
hugetrace-00010	**6979613(6989126.1)**	NA	NA	NA	NA
hugetrace-00020	**9344012(9357095.1)**	NA	NA	NA	NA
inf-europeosm	**43785185(43785235.8)**	NA	NA	NA	NA
inf-germanyosm	**9497489(9497535.3)**	NA	NA	NA	NA
inf-roadNet-CA	**1016026(1016095.6)**	NA	NA	NA	NA
inf-roadNet-PA	**539337(539932.2)**	NA	NA	NA	NA
inf-road-usa	**14394770(14394881.9)**	NA	NA	NA	NA
rec-dating	**11747(11749.3)**	11748	11754	NA	NA
rec-epinions	**9083(9084.8)**	NA	9100	NA	NA
rec-libimseti-dir	**12988(13008.7)**	NA	13031	NA	NA
rggn223s0	**1199405(1199630.6)**	NA	NA	NA	NA
rggn224s0	**2325049(2325222)**	NA	NA	NA	NA
rt-retweet-crawl	**83116(83118.4)**	NA	NA	NA	NA
sc-ldoor	**34062(34101.2)**	NA	NA	NA	NA
sc-msdoor	**15060(15079)**	15279	15391	NA	NA
sc-pwtk	**8732(8772.9)**	8819	8931	NA	NA
sc-rel9	**123682(123963.2)**	NA	NA	NA	NA
sc-shipsec1	**10901(10944.9)**	11791	11954	NA	13766
sc-shipsec5	**13798(13919)**	14839	15082	NA	176869
soc-buzznet	**128(128)**	**128**	**128**	NA	99847
soc-delicious	**57684(57686.8)**	NA	NA	NA	535742
soc-digg	**70654(70659)**	NA	NA	NA	NA
soc-dogster	**27286(27289.6)**	27359	27391	NA	NA
socfb-A-anon	**205956(206047.6)**	NA	NA	NA	NA
socfb-B-anon	**191997(192075.6)**	NA	NA	NA	NA
socfb-uci-uni	**1542466(1542478)**	NA	NA	NA	NA
soc-flickr	**105649(105659.2)**	NA	NA	NA	513557
soc-flickr-und	**296975(297059.7)**	NA	NA	NA	NA
soc-flixster	**91545(91545)**	NA	NA	NA	NA
soc-FourSquare	**60982(60983.7)**	NA	NA	NA	638426
soc-lastfm	**67429(67429)**	NA	NA	NA	NA
soc-livejournal	**854544(854551.88)**	NA	NA	NA	NA
soc-live*-groups	**1128481(1128481)**	NA	NA	NA	NA
soc-LiveMocha	**1427(1428.9)**	1430	1454	1476	103049
soc-ljournal-2008	**1071992(1072003.5)**	NA	NA	NA	NA
soc-orkut	**120565(120616.7)**	NA	NA	NA	NA
soc-orkut-dir	**100505(100618.7)**	NA	NA	NA	NA
soc-pokec	**221632(221703.2)**	NA	NA	NA	NA
soc-sinaweibo	**201634(201634)**	NA	NA	NA	NA
soc-twitter-higgs	**15201(15177.3)**	15335	15375	NA	NA
soc-youtube	**101653(101659.4)**	NA	NA	NA	495497
soc-youtube-snap	**235584(235590.2)**	NA	NA	NA	NA
tech-as-skitter	**199541(199671.5)**	NA	NA	NA	NA
tech-ip	**153(153.3)**	NA	170	NA	NA
twittermpi	**727931(727931)**	NA	NA	NA	NA
web-arabic-2005	**20666(20670.3)**	20737	20730	NA	162433
web-baidu-baike	**273777(273856.7)**	NA	NA	NA	NA
web-it-2004	**34548(34548.9)**	34551	34557	NA	NA
web-uk-2005	**1728(1728)**	**1728**	**1728**	**1728**	NA
web-wiki**2009	**395316(395447.5)**	NA	NA	NA	NA
web-wiki**-growth	**118963(118981)**	NA	NA	NA	NA
web-wikipedlink	**190986(190999)**	NA	NA	NA	NA
wikipedialinken	**212240(212241.3)**	NA	NA	NA	NA

Table 3: Results on the NDR benchmarks. To save space, we denote ACO-efficient and ACO-RVNS as ACO_e and ACO, respectively. Moreover, since min and avg of all competitors are no better than that of NuCDS on all instances, we only report min of competitors.

6.3 Effectiveness of Proposed Strategies

As shown in Table 4, three modified versions of NuCDS are proposed to verify the effectiveness of HDC and $safety$, especially on massive graphs. We compared NuCDS with $NuCDS_1$ to show the effectiveness of $safety$, and with $NuCDS_2$ and $NuCDS_3$ to show the effectiveness of HDC. The results in Table 5 confirm that both strategies are effective. Moreover, when compared with SUB, HDC improves the average best solution by 17.285% on massive graphs.

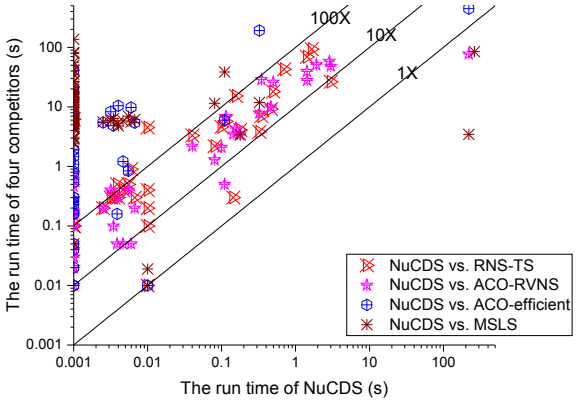

Figure 2: Average running time of NuCDS and competitors.

	$NuCDS_1$	$NuCDS_2$	$NuCDS_3$
HDC	+	-	-
SUB	-	+	-
TBC	-	-	+
$safety$	-	+	+

Table 4: Three modified versions of NuCDS, where "+" indicates that the version uses the corresponding strategy while "-" means not.

Benchmark		vs. $NuCDS_1$	vs. $NuCDS_2$	vs. $NuCDS_3$
SNAP(22)	#Better	14	10	21
	#Worse	2	5	1
DIMACS10(31)	#Better	28	28	23
	#Worse	1	2	5
NDR(65)	#Better	46	59	56
	#Worse	8	1	2

Table 5: Comparing NuCDS with three modified versions on massive graphs. #Better and #Worse respectively represent the number of instances where NuCDS achieves better and worse result.

7 Conclusion

We proposed two new algorithmic components, namely the hybrid dynamic connectivity maintenance heuristic and the safety-based vertex selection heuristic, for MCDS. Both components were used to develop an efficient local search algorithm named NuCDS. We conducted extensive benchmarks to evaluate the performance of NuCDS and the experimental results showed that our algorithm significantly outperforms its competitors on almost all the instances of any size.

Acknowledgements

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

References

[Al-Karaki and Kamal, 2008] Jamal N Al-Karaki and Ahmed E Kamal. Efficient virtual-backbone routing in mobile ad hoc networks. *Computer Networks*, 52(2):327–350, 2008.

[Bouamama *et al.*, 2019] Salim Bouamama, Christian Blum, and Jean-Guillaume Fages. An algorithm based on ant colony optimization for the minimum connected dominating set problem. *Applied Soft Computing*, 80:672–686, 2019.

[Cai and Su, 2013] Shaowei Cai and Kaile Su. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204:75–98, 2013.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 747–753, 2015.

[Cheng *et al.*, 2003] Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks: An International Journal*, 42(4):202–208, 2003.

[Chinnasamy *et al.*, 2019] A Chinnasamy, B Sivakumar, P Selvakumari, and A Suresh. Minimum connected dominating set based rsu allocation for smartcloud vehicles in vanet. *Cluster Computing*, 22(5):12795–12804, 2019.

[Erdem *et al.*, 2009] Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors. *Proceedings of 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 5753 of *Lecture Notes in Computer Science*, 2009.

[Fan and Watson, 2012] Neng Fan and Jean-Paul Watson. Solving the connected dominating set problem and power dominating set problem by integer programming. In *International conference on combinatorial optimization and applications*, pages 371–383, 2012.

[Fernau *et al.*, 2011] Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Daniel Raible, and Peter Rossmanith. An exact algorithm for the maximum leaf spanning tree problem. *Theoretical Computer Science*, 412(45):6290–6302, 2011.

[Fomin *et al.*, 2008] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. Solving connected dominating set faster than 2 n. *Algorithmica*, 52(2):153–166, 2008.

[Gendron *et al.*, 2014] Bernard Gendron, Abilio Lucena, Alexandre Salles da Cunha, and Luidi Simonetti. Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. *INFORMS Journal on Computing*, 26(4):645–657, 2014.

[Glover and Laguna, 1998] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[Hedar and Ismail, 2012] Abdel-Rahman Hedar and Rashad Ismail. Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics*, 3(2):97–109, 2012.

[Hedar *et al.*, 2019] Abdel-Rahman Hedar, Rashad Ismail, Gamal A El-Sayed, and Khalid M Jamil Khayyat. Two meta-heuristics designed to solve the minimum connected dominating set problem for wireless networks design and management. *Journal of Network and Systems Management*, 27(3):647–687, 2019.

[Hopcroft and Tarjan, 1973] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[Jovanovic and Tuba, 2013] Raka Jovanovic and Milan Tuba. Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. *Comput. Sci. Inf. Syst.*, 10(1):133–149, 2013.

[Kann, 1992] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

[Khuller and Yang, 2019] Samir Khuller and Sheng Yang. Revisiting connected dominating sets: An almost optimal local information algorithm. *Algorithmica*, 81(6):2592–2605, 2019.

[Li *et al.*, 2017] Ruizhi Li, Shuli Hu, Jian Gao, Yupeng Zhou, Yiyuan Wang, and Minghao Yin. Grasp for connected dominating set problems. *Neural Computing and Applications*, 28(1):1059–1067, 2017.

[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.*, 2017] Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In *IJCAI*, pages 517–523, 2017.

[Lucena *et al.*, 2010] Abilio Lucena, Nelson Maculan, and Luidi Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7(3):289–311, 2010.

[Milenković *et al.*, 2011] Tijana Milenković, Vesna Memišević, Anthony Bonato, and Nataša Pržulj. Dominating biological networks. *PloS one*, 6(8):e23016, 2011.

[Misra and Mandal, 2009] Rajiv Misra and Chittaranjan Mandal. Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks. *IEEE Transactions on parallel and distributed systems*, 21(3):292–302, 2009.

[Morgan and Grout, 2007] Mike Morgan and Vic Grout. Meta-heuristics for wireless network optimisation. In *AICT*, pages 15–15, 2007.

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

[Ruan *et al.*, 2004] Lu Ruan, Hongwei Du, Xiaohua Jia, Weili Wu, Yingshu Li, and Ker-I Ko. A greedy approximation for minimum connected dominating sets. *Theoretical Computer Science*, 329(1-3):325–330, 2004.

[Simonetti *et al.*, 2011] Luidi Simonetti, Alexandre Salles Da Cunha, and Abilio Lucena. The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm. In *International Conference on Network Optimization*, pages 162–169. Springer, 2011.

[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. *Journal of Artificial Intelligence Research*, 58:267–295, 2017.

[Wu *et al.*, 2017] Xinyun Wu, Zhipeng Lü, and Philippe Galinier. Restricted swap-based neighborhood search for the minimum connected dominating set problem. *Networks*, 69(2):222–236, 2017.