On Construction of Quality Fault-Tolerant Virtual Backbone in Wireless Networks

Wei Wang, Donghyun Kim, Member, IEEE, Min Kyung An, Student Member, IEEE, Wei Gao, Xianyue Li, Zhao Zhang, and Weili Wu, Member, IEEE

Abstract—In this paper, we study the problem of computing quality fault-tolerant virtual backbone in homogeneous wireless network, which is defined as the $k$-connected $m$-dominating set problem in unit disk graph. This problem is NP-hard, and thus many efforts have been made to find a constant factor approximation algorithm for it, but never succeeded so far with arbitrary $k \geq 3$ and $m \geq 1$ pair. We propose a new strategy for computing a smaller size $3$-connected $m$-dominating set in unit disk graph with any $m \geq 1$. We show the approximation ratio of our algorithm is constant and its running time is polynomial. We also conduct a simulation to examine the average performance of our algorithm. Our result implies while there exists a constant factor approximation algorithm for the $k$-connected $m$-dominating set problem with arbitrary $k \leq 3$ and $m \geq 1$ pair, the $k$-connected $m$-dominating set problem is still open with $k > 3$.

Index Terms—Approximation algorithm design and analysis, graph theory, connected dominating set, virtual backbone, fault-tolerance.

I. INTRODUCTION

A wireless network consists a set of computing devices referred as wireless nodes, which are connected with each other using wireless communication technology. During recent years, two particular types of infrastructure-less wireless networks, ad-hoc network and wireless sensor network, drew a spotlight due to their wide range of applications [1], [2]. In the rest of this paper, we will denote these infrastructure-less wireless networks by “wireless networks” in short. Typically, each wireless node is powered by a battery, which is a limited energy source. In wireless communication, the energy consumed to send a message superlinearly grows as the communication distance linearly increases. Therefore, short distance multi-hop communication is the major communication pattern in wireless networks rather than long distance direct communication. However, due to the unique characteristics of wireless communication such as signal collision and interference, designing an efficient multi-hop routing algorithm in wireless networks is known to be very challenging.

Because of its simplicity, many existing routing protocols in wireless networks exploit flooding strategy, in which every node participates in the protocols by broadcasting the messages it received to its neighbors. However, it is known that such flooding-oriented protocols suffer from huge amount of collisions and redundancy, and thus are very energy-exhaustive [3]. It is widely believed that Ephremides et al. first introduced the idea of constructing a backbone-like structure to wireless networks [4], which is generally referred as virtual backbone nowadays. In this scheme, a subset $C$ of nodes is selected such that i) every node in a given wireless network is either in $C$ or adjacent to a node in $C$, and ii) the subgraph induced by $C$ is connected. Then, each node communicates with another through the connected subset $C$. This strategy has several apparent benefits to wireless networks. Above all, regardless from the specific routing algorithm used over this structure, only the nodes in $C$ will be involved in message routing, and therefore the number of routing-related control messages can be reduced and the amount of wireless signal collision and interference will be decreased. As a result, the routing protocol will work much faster and efficiently [5].

Clearly, the advantage of a virtual backbone can be magnified as its size becomes smaller. This motivated many people to investigate the problem of computing smaller virtual backbones in wireless networks. Guha and Koller modeled this problem as the minimum connected dominating set (MCDS) problem [6]. Since this is a very well-known NP-hard problem, they introduced approximation algorithms for it [7]. After all, due to the significant merit and potential of virtual backbone in wireless networks, extensive researches have been conducted on the MCDS problem and its variations [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33].

In many applications of wireless networks, the topology of the networks can be altered due to many reasons such as mobility of nodes, temporal communication disruption, and energy exhaustion of nodes. Unfortunately, a virtual backbone structure is fragile in such environment. That is, once the topology is changed, the structure may be disconnected and needed to be reconstructed. Or, a node may be disconnected from the virtual backbone structure due to the loss of its only neighboring virtual backbone node. In order to deal with these issues, Dai and Wu have introduced the concept of the fault-tolerant virtual backbone [14]. In their seminary work, a virtual backbone is said to be fault-tolerant if it satisfies following two
properties:

i) \textit{k-vertex-connectivity:} a graph $G$ is said to be $k$-connected only if after any $k-1$ nodes are removed from $G$, the graph is still connected. By enforcing such property to a virtual backbone for some $k$ greater than 1, the backbone can be still connected after the loss of at most $k-1$ nodes.

ii) \textit{m-domination:} a subset $C$ of a graph $G$ is an \textit{m-dominating set} of $G$ only if for every node $u \in (G \setminus C)$, $u$ has at least $m$ neighbors in $C$.

By enforcing these two properties, a virtual backbone structure is still operational even after the loss of any $\min\{m-1, k-1\}$ nodes. Generally, the problem of computing the minimum cardinality subset satisfying those two properties is called the \textit{k-connected m-dominating set problem}, the $(k,m)$-CDS problem in short. Apparently, this problem is NP-hard since its simplest case is the MCDS problem, in which $k = 1$ and $m = 1$. So far, many efforts concerning this problem have been made [17], [18], [19], [26], [27], [28], [33]. However, as we showed in our previous surveys [34], [35], there exists no constant factor approximation algorithm which always succeeds in computing a $(k,m)$-CDS with arbitrary $k \geq 3$ and $m \geq 1$ pair from a UDG with a feasible solution.

In this paper, we propose the first polynomial time constant factor approximation algorithm for the $(3,3)$-CDS problem. Then, we generalize this result to have the first polynomial time constant factor approximation for the $(3, m)$-CDS for any positive integer $m \geq 1$. We also conduct a simulation to exam the performance of our algorithm. Due to our consecutive efforts, the problem of designing a constant factor approximation algorithm for the $k$-connected $m$-dominating set problem in unit disk graph with arbitrary $k > 3$ and $m \geq 1$ pair is proven to be still open.

The rest of this paper is organized as follows. In Section II, we introduce some notations, assumptions, and definitions. The related works are introduced in Section III. Section IV provides preliminary information. Our major result, a polynomial time constant factor approximation algorithm for the $(3,m)$-CDS problem for any $m \geq 1$ is introduced in Section V. Our simulation result is given in Section VI. We conclude our paper and present future works in Section VII.

II. Notations, Assumptions, and Definitions

In this paper, we assume that the wireless networks of our interest are homogeneous (i.e. the wireless networks of identical wireless nodes), and use the unit disk graph, which

is defined below, to abstract the networks. Let $udist(u,v)$ be the euclidean distance between $u$ and $v$.

Definition 2.1: A graph $G = (V, E) = (V(G), E(G))$ is a unit disk graph (UDG) if $\forall u, v \in V, (u,v) \in E$ if and only if $udist(u,v) \leq 1$.

Definition 2.2: A subset $I \subseteq V$ is an independent set (IS) of $G$ if $\forall u, v \in I, (u,v) \notin E$.

Definition 2.3: A subset $M \subseteq V$ is a maximal independent set (MIS) of $G$ if $M$ is an IS of $G$ and $\forall u \in (V \setminus M), \{u\} \cup M$ is not an IS anymore.

Definition 2.4: A subset $D \subseteq V$ is a dominating set (DS) of $G$ if $\forall u \in V, \text{ either } u \in D \text{ or } \exists v \in D \text{ such that } (u,v) \in E$.

Definition 2.5: A subset $C \subseteq V$ is a connected dominating set (CDS) of $G$ if $C$ is a DS and the subgraph of $G$ induced by $C$ is connected.

Clearly, an MIS $M$ of $G$ is a DS of $G$. Based on this fact, one typical approach to compute a CDS $C$ is

i) compute an MIS $M$ using a simple 2-coloring algorithm,

ii) find a subset $A$ such that the subgraph induced by $M \cup A$ is connected.

Usually, a variation of minimum spanning tree algorithm or Steiner minimum tree algorithm is used to find $A$ such that $|A| \leq c|M|$ for some constant $c$. Based on this relationship and a known fact that $|M| \leq dOPT_{CDS}$ for some constant $d$, where $OPT_{CDS}$ is an optimal CDS, the above strategy is in fact an approximation of the MCDS problem whose performance ratio is $c \cdot d$. This approach is initially introduced by Guha and Kuller [6] and later becomes a standard approximation technique for the MCDS problem and its variations.

Definition 2.6: A DS $D$ is a $(m)$-DS of $G$ if $\forall u \in (V \setminus D), u$ is a neighbor of at least $m$ nodes in $D$.

Definition 2.7: A graph $G$ is $k$-vertex-connected if the subgraph induced by $V \setminus X$ is connected for any $X \subseteq V$ such that $|X| < k$.

In the rest of the paper, “$k$-vertex-connected” and “$k$-connected” will be used interchangeably. In addition, $G_k$ will imply a $k$-connected graph.

Definition 2.8: A subset $D$ is a $k$-connected $m$-dominating set of $G$ if the subgraph induced by $D$ is $k$-connected and $D$ is $m$-dominating set of $V$.

In the rest of paper, we will interchangeably use $(k,m)$CDS and $C_{k,m}$ to denote a $k$-connected $m$-dominating set. Also, we will use $k$-CDS to note $(k,k) - CDS$. Finally, $C_{k,m}^{\text{opt}}$ is an optimal $(k,m)$-CDS. This paper assumes that the UDG induced from the input WSN is $\max\{m,k\}$-connected. However, how to obtain such a WSN is out of scope of this paper, and we will focus on how to obtain quality $k$-connected $m$-dominating set. We also assume the connectivity of each pair of node remain same during the computation of a $(k,m)$-CDS.

Definition 2.9: A vertex $u \in V$ is a cut-vertex of $G$ only if the graph induced by $V \setminus \{u\}$ is disconnected.

Definition 2.10: A subgraph of $G$ is called a block only if it is a maximal connected subgraph of $G$ without any cut-vertex.

Proposition 2.1: Every block in a connected graph is either a maximal 2-connected subgraph or a bridge, which is an edge with two end points.
Definition 2.11: Given a connected subgraph \( G \) which can be decomposed of a set of blocks and cut-vertices, a leaf-block graph \( G' \) is an induced graph from \( G \) by the following construction rules.

i) For each block \( B \) of \( G \), add a corresponding node \( v_B \) to \( G' \).

ii) For each cut-vertex \( u \) of \( G \), add a node \( u \) to \( G' \).

iii) Add an edge between \( v_B \) in \( G' \), which is added by the first rule, and \( u \) in \( G' \), which is added by the second rule, only if \( u \in B \) in \( G \), where \( B \) is the block of \( G \) which corresponds to \( v_B \).

Figure 1 illustrates how a leaf-block graph can be constructed. In the rest of the paper, a sub-block refers to a block in a leaf-block graph of a block. A leaf block is a block which has degree one in the leaf-block graph. We would like to draw the attention to following three facts.

- **Fact 1**: Two different blocks of \( G \) share at most one cut-vertex in common. Since otherwise, they can be merged into one block by definition.

- **Fact 2**: Every edge of \( G \) lies in a unique block. That is, no two blocks share the same edge.

- **Fact 3**: \( G \) is the union of its blocks.

Based on the facts, following proposition naturally follows.

Proposition 2.2 ([38]): The leaf-block graph of a connected graph is a tree (i.e. a connected graph).

Definition 2.12: For a connected graph \( G \), a separator of \( G \) is pair of vertices \( \{u, v\} \subseteq V \) such that the subgraph induced by \( G \setminus \{u, v\} \) is disconnected.

Definition 2.13: For a 2-connected graph \( G_2, u \in V(G_2) \) is a good-point only if the subgraph induced by \( G_2 \setminus \{u\} \) is 2-connected. Otherwise, we call \( u \) a bad-point. (i.e. \( G_2 \setminus \{u\} \) is 1-connected)

Definition 2.14: An \( H \)-path \( P \) of an induced subgraph \( H \) of a graph \( G \) is a path between two distinct nodes in \( H \) such that no internal node of \( P \) is in \( H \). The length of an \( H \)-path \( P \) is just the number of edges of \( P \). Also, \( H_k \) means an \( H \)-path with length at most \( k \).

So far, we have introduced a series of important notations and definitions. Some other ones will be defined later if necessary. At last, we introduce several important lemmas and a theorem.

Lemma 2.3: A 2-connected graph without any bad-point is a 3-connected graph.

Proof: Suppose \( G \) is a 2-connected graph. If there is no bad-point in \( G \), for any \( u \in V \), the induced graph of \( V \setminus \{u\} \) is still 2-connected. Then, by definition, \( G \) is a 3-connected graph and this lemma holds true.

Lemma 2.4: Consider a 3-connected graph \( G_3 \) and a 2-connected 3-dominating set \( C_{2,3} \) of \( G_3 \). Now, suppose \( \{c_1, c_2\} \) is a separator of \( C_{2,3} \) such that \( C_{2,3} \setminus \{c_1, c_2\} \) is divided into several connected components, \( \{B_1, B_2, ..., B_i\} \). Then, there has to be two distinct components \( B_i \) and \( B_j \) having a shortest path between them with length at most three hops.

Proof: Suppose \( B_1 \) and \( B_2 \) are the two closest components. Since \( G_3 \) is 3-connected, there must exist a path in \( G_3 \setminus \{c_1, c_2\} \) connecting \( B_1 \) and \( B_2 \). Let \( \{w_0, w_1, ..., w_l\} \) be the shortest \( H \)-path between them. We show that \( l \leq 3 \).

For contradiction, suppose that \( l \geq 4 \). Note that every node in \( G_3 \setminus C_{2,3} \) is 3-dominated by \( C_{2,3} \), it follows that every \( w_i \) is 3-dominated by \( C_{2,3} \) for \( i = 1, 2, ..., l - 1 \) (see Figure 2).

Now, consider the node \( w_2 \). We find that \( w_2 \) can be dominated by neither \( B_1 \) nor \( B_2 \); for otherwise, the length between \( B_1 \) and \( B_2 \) can be shortened by at least one. Moreover, \( w_2 \) can be dominated by \( c_1 \) and \( c_2 \) at most twice. It follows that there must exist another connected component, say \( B_k \) \( (k \neq 1, 2) \), which dominates \( w_2 \). However, the distance between \( B_k \) and \( B_2 \) is at most \( l - 1 \), which contradicts the fact that \( B_1 \) and \( B_2 \) are the closest components. This shows that \( l \leq 3 \) and the lemma follows.

Lemma 2.5: Consider a 3-connected graph \( G_3 \), a \( C_{2,3} \) of \( G_3 \), and a separator \( \{c_1, c_2\} \) of \( C_{2,3} \). Then, \( C_{2,3} \setminus \{c_1, c_2\} \) is divided into at most five connected components.

Proof: It is known that in UDG, a node has at most five independent neighbors [15]. By definition, each connected component of \( C_{2,3} \setminus \{c_1, c_2\} \) has to be independent and connected to \( c_1 \) (and \( c_2 \)), their number cannot exceed more than five.

Theorem 2.6 (Menger’s Theorem [40]): Given a graph \( G \) and two vertices \( u, v \in V \), the minimum number of vertices to be removed from \( G \) so that \( u \) and \( v \) are separated in the remaining graph is equivalent to the maximum number of disjoint paths from \( u \) to \( v \) in \( G \).

III. Related Work

The problem of computing quality fault-tolerant virtual backbone is introduced by Dai and Wu [14]. In their seminar work, three localized heuristic algorithms for the \((k,k)\)-CDS problem are introduced. The first approximation algorithm to construct fault-tolerant virtual backbone was proposed by Wang et al. [28]. They specifically focused on the \((2,1)\)-CDS problem and they proved the performance ratio of their algorithm is 62.19. A constant factor approximation for the \((1,m)\)-CDS problem is introduced by Shang et al. [17] and Thai et.al. [18] independently. Shang et al. [17] introduced how Wang et al.’s idea for the \((2,1)\)-CDS problem can be incorporated with their approximation algorithm for the \((1,m)\)-CDS problem to achieve a constant factor approximation for the \((2,m)\)-CDS problem. As a part of the conclusion, they conjectured that it would be very difficult to design a constant factor approximation algorithm for the \((k,m)\)-CDS problem for any \( k \geq 3 \) and \( m \geq 1 \) pair.
During recent few years, extensive efforts have been made to find a constant factor approximation algorithm for the \((k, m)\)-CDS problem for arbitrary \(k \geq 3\) and \(m \geq 1\) pair. In most case, UDG is used to abstract a wireless network. Thai et al. tried to use a generalization of Wang et al.’s approximation scheme for the \((2, 1)\)-CDS problem to design a constant factor approximation [18]. Based on this result, Zhang et al. [26] designed an approximation algorithm for the \((k, m)\)-CDS problem in terms of the size of a \((k, m)\)-CDS as well as its diameter. Wu et al. [19] proposed the first distributed approximation algorithm for this problem. After then, several centralized/distributed approximation algorithms for the \((k, m)\)-CDS problem are introduced [27], [33]. However, as we showed in our previous surveys [34], [35], there is no constant factor approximation algorithm which always succeeds in computing a \((k, m)\)-CDS with arbitrary \(k \geq 3\) and \(m \geq 1\) pair from a UDG including a feasible solution.

IV. PRELIMINARIES

A. Approximation for the \((2, 1)\)-CDS Problem

Now, we introduce the CDS augmentation algorithm (CDSA) which is the first constant factor approximation algorithm to multi-connected CDS [28]. The input of this algorithm is a 2-connected UDG \(G\). The algorithm firstly employs an approximation algorithm such as [30] to generate 1-CDS of \(G\), which is denoted by \(C_{1,1}\). Then, the algorithm uses following procedures to add more nodes to \(C_{1,1}\) and make the union of them 2-connected.

i) Suppose \(C = C_{1,1}\). Identify a set \(L\) of all blocks in \(C_{1,1}\).

ii) Identify a leaf block \(B \in L\) as well as the shortest \(H\)-path \(P\) from \(u \in B\) to \(v \in C_{1,1} \setminus B\) such that \(u\) and \(v\) are non-cut-vertices. Then, add all of the nodes in \(P\) to \(C\), which makes two blocks in \(L\) becomes one block. This step is repeated until there is only one block left in \(L\).

Using a geometrical property of UDG, the authors proved that each path \(P\) can contain at most \(8\) nodes which are not in \(C\) yet. They also showed that by adding at most \(|C_{1,1}|\) such passes to \(C\), \(C\) becomes a \((2, 1)\)-CDS from a \((1, 1)\)-CDS. Therefore,

\[
|C_{2,1}| \leq |C_{1,1}| + 8|C_{1,1}| = 9|C_{1,1}|.
\]

Since \(|C_{1,1}| \leq c|C_{1,1}^{opt}|\) is guaranteed by the \(c\)-approximation algorithm employed for some \(c\) and \(|C_{1,1}^{opt}| \leq |C_{2,1}^{opt}|\) is clearly true, we have

\[
|C_{2,1}| \leq 9c|C_{1,1}^{opt}| \leq 9c|C_{2,1}^{opt}|.
\]

Originally, the performance ratio of this strategy is proven to be 62.19. However, by the most recent result by Li et al. [29], \(c = 6.075\) is achievable, and thus, the approximation ratio of this strategy for the \((2, 1)\)-CDS problem can be trivially improved to 54.675.

B. Approximation for the \((1, m)\)-CDS Problem

In this subsection, we discuss the strategy to approximate the \((1, m)\)-CDS problem in [17], [18], which consists of following two distinct steps.

i) Given a UDG \(G\), find a \((1,1)\)-CDS \(C_{1,1} = M_0 \cup A\), where \(M_0\) is an MIS of \(G\) and \(A\) is a subset of nodes in \(V \setminus M_0\).

This can be done by exploiting an existing approximation algorithm for the \((1,1)\)-CDS problem such as [30].

ii) Let \(C = C_{1,1}\). For each \(i = 1\) to \(m - 1\), repeat following:

- Find an MIS \(M_i\) of \((V \setminus (A \cup M_0 \cup \cdots \cup M_{i-1}))\) and add it to \(C\). After the completion of the repetitions, return \(C\).

After the first step, each node in \(V \setminus C_{1,1}\) has to be adjacent to at least one node in an MIS \(M_0 \subset C_{1,1}\) of \(G\). In the second step, after each \(M_i\) is computed for \(1 \leq i \leq m - 1\), each node in \(V \setminus (C_{1,1} \cup (\bigcup_{1 \leq j \leq i} M_j))\) must have at least one neighbor in each \(M_i\) for each \(0 \leq j \leq i\). Therefore, after the second step, \(C\) is a \((1, m)\)-CDS of \(G\).

Lemma 4.1 ([17]): Suppose \(G\) is a UDG, \(M\) is an MIS of \(G\), and \(D^{opt}_m\) is a minimum \(m\)-DS of \(G\). Then, \(|M| \leq \max\{\frac{5}{m}, 1\}|D^{opt}_m|\).

Theorem 4.2 ([17]): The performance ratio of this algorithm for the \((1, m)\)-CDS problem is \((5 + \frac{5}{m})\) for \(m \leq 5\) and \(7\) for \(m > 5\).

Proof: Suppose \(S_i = M_i \cap D^{opt}_m\). Then, each \(M_i \setminus S_i\) is an IS and \((M_i \setminus S_i) \cap D^{opt}_m = \emptyset\). Suppose \(m \leq 5\). Then, from Lemma 4.1, we have \(|M_i \setminus S_i| \leq \frac{5}{m}|D^{opt}_m \setminus S_i|\) and

\[
\Big| \bigcup_{1 \leq i \leq m-1} M_i \Big| = \sum_{i=0}^{m-1} |S_i| + \sum_{i=0}^{m-1} |M_i \setminus S_i| \leq \sum_{i=0}^{m-1} |S_i| + \sum_{i=0}^{m-1} \frac{5}{m}|D^{opt}_m \setminus S_i|
\]

\[
= \sum_{i=0}^{m-1} |S_i| + \sum_{i=0}^{m-1} \left( |D^{opt}_m| - |S_i| \right)
\]

\[
= \left( 1 - \frac{5}{m} \right) \sum_{i=0}^{m-1} |S_i| + 5|D^{opt}_m|.
\]

Therefor, we have \(|\bigcup_{0 \leq i \leq m-1} M_i| \leq 5|D^{opt}_m|\) for \(m \leq 5\). Similarly, \(|\bigcup_{0 \leq i \leq m-1} M_i| \leq 6|D^{opt}_m|\) for \(m > 5\) since by definition \(S_i = M_i \cap D^{opt}_m\) and thus \(S_i \cup S_{i+1} \cup \cdots \cup S_{m-1} \subseteq D^{opt}_m\), which implies \(1 - \frac{5}{m} \sum_{i=0}^{m-1} |S_i| \leq |D^{opt}_m|\) for \(m > 5\). From Lemma 4.1, we can conclude

\[
|A| \leq |M_0| \leq \max\\{\frac{5}{m}, 1\}\cdot |D^{opt}_m|,
\]

because \(|A| \leq |M_0|\) is true if a coloring algorithm is used to find \(M_0\). As a result, the size of \(|C|\), an output of this algorithm, is bounded by \((5 + \frac{5}{m})|D^{opt}_m|\) for \(m \leq 5\) and \(7|D^{opt}_m|\) for \(k > 5\).

C. Approximation for the \((2, m)\)-CDS Problem

Throughout Section IV-A and Section IV-B, two constant factor approximations, one for the \((2, 1)\)-CDS problem and another for the \((1, m)\)-CDS problem, are introduced, respectively. By combining these two strategies, an approximation for the \((2, m)\)-CDS problem can be obtained as follows [17].

i) Given a UDG \(G\), apply the approximation algorithm for the \((1, m)\)-CDS problem introduced in Section IV-B and obtain a \((1, m)\)-CDS of \(G\) which is denoted by \(C_{1,m}\). Let \(C = C_{1,m}\).
ii) Apply a variation of Wang et al.’s approximation strategy for the (2, 1)-CDS problem in Section IV-A to C. In detail, to remove a cut-vertex in C, which is a vertex shared by a leaf block B1 and another block B2 of C, an H-path P with maximum length 3 between them is identified and added to C1,m. Here, each end of P has to be connected to u ∈ B1 and v ∈ B2 such that u and v are not cut-vertices of C. After all cut-vertices in C are removed, the algorithm output C. Clearly, the output C of this algorithm is a (2, m)-CDS of G. The following theorem show that this strategy is an approximation of the (2, m)-CDS problem.

Theorem 4.3 ([17]): The performance ratio of this algorithm for the (2, m)-CDS problem is (5 + \frac{25}{m}) for 2 ≤ m ≤ 5 and 11 for m > 5.

Proof: Suppose Copt1,m and Copt2,m are an optimal (1, m)-CDS and an optimal (2, m)-CDS of G, respectively. Note that
i) There exists an approximation algorithm to generate a CDS |C1| = |M0| + |A|, where M0 is an MIS of G and A is a subset of V \ M0 such that the subgraph induced by M0 ∪ A is connected,
ii) |A| ≤ |M0| if a coloring strategy is used to generate a M0,
iii) |Dopt| ≤ |Copt1,m| and |M0| ≤ max{\frac{5}{m}, 1}|Dopt| (from Lemma 4.1),
iv) in this algorithm, a C2,m is obtained from a C1,m by adding at most 2|C1,m| nodes from V \ C1,m, and v) |Copt2,m| ≤ |Copt1,m|.

From the first two facts, we have |C1,m| ≤ 2|M0|, and by combining this inequality with the third and fourth facts, we can conclude

|C2,m| ≤ |C1,m| + 2|C1,m| ≤ |C1,m| + 4|M0|
≤ |C1,m| + 4 max{\frac{5}{m}, 1}|Copt1,m|.

Finally, by combining this with the last fact and Theorem 4.2, we have |C2,m| ≤ (5 + \frac{25}{m})|Copt2,m| for 2 ≤ m ≤ 5 and |C2,m| ≤ 11|Copt2,m| for m > 5, and this theorem is proved. □

V. A CONSTANT FACTOR APPROXIMATION ALGORITHM FOR COMPUTING 3-CONNECTED m-DOMINATING SETS IN UNIT DISK GRAPHS

In this section, we introduce our algorithm, Fault-Tolerant Connected Dominating Sets Computation Algorithm (FT-CDS-CA), which computes (3, m)-CDSs in UDGs. We first proceed this section with m = 3 and later explain how our result can be generalized for any m ≠ 3. Roughly, FT-CDS-CA works as follows: Given a G3, FT-CDS-CA first computes a C2,3 ⊆ G3 using a constant factor approximation algorithm in [17, 18] and set Y ← C2,3. Next, the algorithm identifies the set of all bad-points X in C2,3. Note that |X| ≤ |C2,3|. By Lemma 2.3, the 2-connected subgraph C2,3 is 3-connected if X = 0. The core strategy of FT-CDS-CA is that the algorithm repeatedly changes each bad-point v ∈ X into a good point by moving at most some constant number of nodes from G3 \ Y to Y without introducing any new bad-point into X. In this way, after changing all bad points in X into good, Y becomes 3-connected 3-dominating set, and accordingly, the total number of nodes newly added to C2,3 is bounded by a constant factor of |C2,3|.

Algorithm 1 FT-CDS-CA (G3)

1: Compute a C2,3 and set Y ← C2,3.
2: Identify the set of bad points X in Y.
3: while X ≠ ∅ do
4: i* Step 1: Multi-level Decomposition */
5: (T, v, l) ← MLD(X, Y, Y, v, 0), where v is one bad-point of Y.
6: if l = 0 then
7: T_l ← T_l.
8: else
9: i* Step 2: Merging Sub-Blocks */
10: T_l ← MSB(X, Y, T_l, v).
11: end if
12: i* Step 3: One Bad-Point Elimination */
13: (X, Y) ← OBPE(G3, X, Y, T_l, v)
14: end while
15: Return Y.

A. How to Convert a Bad-Point to a Good-Point?

FT-CDS-CA is a round-based algorithm. In each round, it converts one bad-point in Y to a good-point as follows: Given a 2-connected subgraph Y (initially, this is a C2,3) and the set X of bad-points in Y, we select v ∈ X as a root and compute a leaf-block tree T_0 of Y \ {v}. Then, T_0 consists of a set of blocks {B_1, B_2, · · · , B_s} and a set of cut-vertices {c_1, · · · , c_t}. Clearly, v can constitute a separator only with another node in {c_1, · · · , c_t}. To simplify our presentation, we classify the nodes in each block B_i into the following two classes:

i) internal nodes I(B_i) := {w ∈ B_i | (w, u) ∈ E, ∀u ∈ Y \ B_i}, i.e., the nodes that cannot be adjacent to any node outside B_i;

ii) external nodes, i.e., the remaining nodes E(B_i) := B_i \ I_i, which consist of those spy-nodes S_i := {u ∈ B_i | (u, v) ∈ E} and those cut-vertices C_i := B_i \ {c_1, c_2, · · · , c_t} in each level of decompositions. Note that while S_i and C_i are two different sets, they may share some nodes in common. The external nodes are potential trouble-makers when we try to change a bad point into good.

Suppose there is no external node. If every B_i has no internal bad point for 1 ≤ i ≤ t, then FT-CDS-CA makes either one of c_k for 1 ≤ k ≤ t or v to be a good-point by adding constant number of H_3-paths such that Y \ {c_k} or Y \ {v} is still 2-connected (Algorithm 4). Otherwise, there must exist some B_i having an internal node w ∈ B_i which constitutes a separator {w, u} of Y with another node u ∈ Y. By Lemma 5.1, any internal bad point w in B_i can constitute a separator only with another node in B_k (while this may not be true for the external nodes such as cut-vertices and spy-nodes).
Algorithm 2 MLD \((X, Y, B, v, i)\)

1: Calculate a leaf-block tree \(T_i\) of \(B \setminus \{v\}\).
2: if all blocks in \(T_i\) are good blocks then
   3: Return \((T_i, v, i)\)
4: else
   5: Return MLD \((X, Y, B_j, u, i + 1)\), where \(B_j\) is a bad block of \(T_i\) and \(u\) is an internal bad point of \(B_j\).
6: end if

It follows that \(u \in B_i\). Note that by switching \(w\) to \(v\), we can make the original problem smaller. By repeating such process, we can keep making our problem smaller and eventually can find a block \(B_i\) in which our strategy works. Finally, to handle the external nodes (which constitutes ‘local separators’ with \(c_k\)), we simply add \(H_k\)-paths to eliminate these separators.

Now, we introduce the detailed description of our algorithm for the conversion, which consists of following three discrete steps, 1) Multi-Level Decomposition, 2) Merging Sub-Blocks, and 3) One Bad-Point Elimination.

Note that the first step takes a polynomial time since in each level, we can compute a leaf-block tree within a polynomial time and at most a polynomial number of trees have to be computed. The second step also takes a polynomial time since it will try to merge the limited number of blocks whose number is bounded by the number of nodes in the original graph. At last, the third step takes a polynomial time to make one bad-point to a good-point. Since at most a polynomial number of rounds are repeated to eliminate all bad-points and find a subgraph \(C_{3,3}\), the algorithm is clearly polynomial time executable (see Lemma 5.10 for details).

1) Multi-Level Decomposition (MLD): The purpose of this step is to find a subgraph in which the third step can be applied to convert at least one bad-point in \(X\) into a good point. We assume that \(X \neq \emptyset\), since otherwise \(Y\) is already 3-connected. Given a 2-connected block \(B \leftarrow Y\), FT-CDS-CA first picks one \(v \in X\) and starts the initial decomposition process (say level-0 decomposition). Then, \(B \setminus \{v\}\) is decomposed into a (level-0) leaf-block graph \(T_0\), which is a tree whose vertices consist of a set of blocks \(\{B_1, \ldots, B_s\}\) and a set of cut-vertices \(\{c_1, c_2, \ldots, c_l\}\). Observe that after the level-0 decomposition, \(s \geq 2\) and \(l \geq 1\).

Now, FT-CDS-CA examines each block in \(T_0\) to see if there is a block \(B_i\) having an internal bad point in it. By Lemma 5.1, we only need to check if there is an internal node \(w \in I_i \subset B_i\) which can constitute a separator of \(Y\) with another node in \(B_i\). We will call such block containing an internal bad point a bad block; otherwise it is called a good block. Once we find a bad block \(B_i\), we stop searching, set \(B \leftarrow B_i\), \(v \leftarrow w\), and start next level (level-1) decomposition process on \(B\). If such pair does not exist, we are done with this step. Note that this decomposition step never outputs an empty left-block tree since each block must contain a cut-vertex which is an external node. Once we found a bad-block-free leaf-block tree, we make the following modifications to the tree before starting the third step, “one bad-point elimination” on \(B\).

Algorithm 3 MSB \((X, Y, T_i, v)\)

1: Copy \(T_i\) to \(T'_i\).
2: if There is exactly one block having external nodes connecting to nodes outside \(B\) directly then
   3: We mark this as a virtual block \(V B\) (no virtual cut-vertex).
   4: Return \(T'_i\).
5: end if
6: repeat
7: boolMerged ← FALSE
8: for each pair of two different \(B_i\) and \(B_j\) of \(T'_i\) do
9: if i) each of \(B_i\) and \(B_j\) contains an external node \(x\) and \(y\) of \(T'_i\), respectively, and ii) \(x\) and \(y\) are connected through a path in \(Y \setminus (T_i \cup \{v\})\) then
10: if \(x\) and \(y\) are the cut-vertices of \(T'_i\) then
11: Merge all the blocks on the path from \(x\) to \(y\) in \(T'_i\) into one single block.
12: boolMerged ← TRUE
13: Get out of for loop */ break */
14: else if neither of \(x\) nor \(y\) is a cut-vertex of \(T'_i\) then
15: Merge all the blocks on the path from \(B_i\) to \(B_j\) in \(T'_i\) (including \(B_i\) and \(B_j\)) into one single block.
16: boolMerged ← TRUE
17: Get out of for loop */ break */
18: else
19: Without loss of generality, suppose \(x\) is a cut-vertex and \(y\) is not a cut-vertex. Then, merge all the blocks on the path from \(x\) to \(B_j\) in \(T'_i\) (including \(B_j\)) into one single block.
20: boolMerged ← TRUE
21: Get out of for loop */ break */
22: end if
23: end if
24: end for
25: until boolMerged is FALSE
26: Mark the merged big block as virtual block \(V B\) and virtual cut-vertex (or vertices) if applicable.
27: Return \(T'_i\)

2) Merging Sub-Blocks (MSB): Suppose after the level-\(l\) decomposition of \(B \setminus \{v\}\) for \(l > 0\), we have

\[ V(T_i) = \{B_1, B_2, \ldots, B_s\} \cup \{c_1, \ldots, c_l\} \quad (1) \]

and every \(B_i\) is a good block for \(1 \leq i \leq s\). Please observe that there exists at least one block \(B_i\) having external nodes that are adjacent to nodes in \(Y \setminus B\), since otherwise, \(B \) and \(Y \setminus B\) cannot be connected with each other. In addition, we would like to emphasize that there may exist more than one block having external nodes connected to some nodes outside \(B\). Suppose that \(B_i\) (resp. \(B_j\)) contains external node \(x\) (resp. \(y\)) and \(i \neq j, x \neq y\). Consider the following cases:

i) If neither \(x\) nor \(y\) is a cut-vertex, then every cut-vertex \(c_k\) lying on the path between \(B_i\) and \(B_j\) in \(T_i\) is actually not a cut-vertex of \(Y\) (though it is a cut-vertex in \(B \setminus \{v\}\)).
Thus, all blocks lying on the path between \( B_i \) and \( B_j \), should be merged to form a bigger block of \( B \setminus \{ v \} \).  

ii) If \( x = c_i \) and \( y = c_j \) are cut-vertices of \( B_i \) and \( B_j \) respectively, then the blocks lying on the path between \( c_i \) and \( c_j \) in \( T_1 \) should be merged, and the internal cut-vertices in the path do not constitute a separator with \( v \) in \( Y \) (while this not true in \( B \)).

iii) If exactly one of \( x \) and \( y \) is a cut-vertex, i.e. \( x = c_i \) is a cut-vertex and \( y \in B_j \) is not a cut-vertex, then the blocks lying on the path between \( c_i \) and \( B_j \) have to be merged.

After merging all possible blocks into one bigger block, we obtain a modified leaf-block tree \( T'_0 \) in which one bigger block \( VB \) (we call it a virtual block) is added representing all the the merged blocks, and all the cut-vertices \( c_i \) which do not constitute a separator with \( v \) will be removed (if there is exactly one block having external nodes that are adjacent to nodes outside \( B \), we simply choose this block as \( VB \)). Moreover, we mark every remaining cut-vertex of \( VB \) as a virtual cut-vertex, if it is adjacent to a node outside of \( B \). Note that by Lemma 5.2, there is a unique \( VB \) after the merging process, which can be connected to the outside of \( B \) directly without going through \( v \); see Fig. 3.

Note that if \( l = 0 \), no block has to be merged since no block having an external node adjacent to \( Y \setminus B = \emptyset \). However, to unify our discussions, we make the convention that the virtual block \( VB \) of \( T_0 \) is a node in \( T_0 \) with degree larger than two (if exists), or an endpoint of \( T_1 \) (now \( T_1 \) is a path).

3) One Bad-Point Elimination (OBPE): At this point, we have a leaf-block tree \( T_1 \) as in Eq. (1), which is obtained from \( B \setminus v \). Suppose that \( T_1, l > 0 \) includes no bad block and the merging process above has been executed (unless \( l \) is 0). Then, we have one virtual block \( VB \) and zero to many virtual cut-vertices. Let us denote a virtual cut-vertex by \( c \). Note that we must have \( s \geq 2 \), since otherwise \( v \) is not a bad point.

In this step, we employ a simple process to make either \( v \) or one of the cut-vertices in \( \{ v_1, v_2, \ldots, v_l \} \cup C \) ( \( C \) is the set of remaining cut-vertices in the virtual block \( VB \) ) to be a good point. The key point here is that there always exists at least one leaf-block in \( T_1 \) which does not contain any external node that is adjacent to \( Y \setminus B \), i.e., it cannot be connected to the outside of \( B \) directly without going through \( v \), since otherwise \( v \) would be a good point. Motivated by this observation, now we explain our core strategy after introducing one definition which will make our writing more concise.

**Definition 5.1:** Consider a leaf-block tree \( T_1 \) obtained after the level \( l \) decomposition of the original graph. Then the leaf-root path is the path starting from a leaf-block of \( T_1 \) to \( VB \), the root of \( T_1 \).

Now, we pick the longest leaf-root path

\[
P = \{ \tilde{B}_0, \tilde{c}_1, \tilde{B}_1, \ldots, \tilde{c}_i, \tilde{B}_i, \ldots, \tilde{c}_k, VB \},
\]

Algorithm 4 OBPE \((G_3, X, Y, T_1, V)

1: Suppose \( P = \{ B_0, \tilde{c}_1, \tilde{B}_1, \ldots, \tilde{c}_i, \tilde{B}_i, \ldots, VB \} \) is the longest leaf-root path of \( T_1 \).

2: if There is only one cut-vertex \( c \) in \( P \) then

3: if \( c \) is the only cut-vertex in \( T_1 \) then

4: \( T_i \) is a star centered at \( c \) and \( Y \setminus \{ \tilde{c}, v \} \) can be partitioned into at most five parts. Employ (move nodes from \( G_3 \setminus Y \) to \( Y \)) at most four \( H_3 \)-paths to make \( v \) to be good.

5: else

6: \( T_i \) has at most five leaf blocks connected to \( v \). Also each leaf block \( B_i \) is connected to the root of \( T_1 \), \( VB \), via one cut-vertex \( c_i \). Employ at most five \( H_3 \)-path such that none of \( \{ v, c_i \} \) for \( 1 \leq i \leq 5 \) can form a separator.

7: end if

8: else

9: Elect a new root \( R \) of \( T_1 \) (see the main text for details) and modify the leaf-root path \( P \) to \( \{ B_0, \tilde{c}_1, \ldots, R \} \). Let \( P_1 = P \setminus \{ B_0, \ldots, \tilde{c}_i \} \). Let \( LB(\tilde{c}_1) \) be the set of leaf-blocks being adjacent with \( \tilde{c}_1 \) in \( T_i \).

10: if \( \tilde{B}_1 = R \) then

11: Find an \( H_3 \)-path \( \Gamma_i \) connecting \( LB_1 \setminus \{ \tilde{c}_1 \} \) with \( Y \setminus (LB_1 \cup \{ v \}) \) for each \( LB_1 \in LB(\tilde{c}_1) \) (\( 1 \leq i \leq q \leq 4 \)), and add the nodes in the \( H_3 \)-paths to \( Y \).

12: Apply Procedure-EEN and convert \( \tilde{c}_1 \) into a good point (remove \( \tilde{c}_1 \) from \( X \)).

13: else

14: In this case, there are at least two cut-vertices in \( P \). Then, the first two cut-vertices \( \tilde{c}_1 \) and \( \tilde{c}_2 \) constitute a pair of separator of \( Y \), and thus there must exist an \( H_3 \)-path \( \Gamma \) starting from \( B_1 \setminus \{ \tilde{c}_1, \tilde{c}_2 \} \) ending at \( Y \setminus B_1 \) with endpoint \( x \).

15: if \( x \) does not lie in \( P \) then

16: Add \( H_3 \) paths \( \Gamma_1 \) for each \( i \) and \( \Gamma \) to \( Y \).

17: Apply Procedure-EEN and convert \( \tilde{c}_1 \) into a good point (remove \( \tilde{c}_1 \) from \( X \)).

18: else

19: Apply Procedure-ECP and convert one of \( \{ \tilde{c}_2, \ldots, \tilde{c}_p \} \) into a good point by adding a constant number of \( H_3 \)-paths.

end if

21: end if

22: end if
where \( k \geq 1 \) is the number of cut-vertices in \( P \). In what follows, we will make either \( v \) (if \( k = 1 \)) or one of \( \tilde{c}_i \) (1 \( \leq i \leq k - 1 \), if \( k \geq 2 \)) to be good (but remember we will never try to make any cut-vertex to be good if it is a virtual cut-vertex or adjacent to a virtual block). We distinguish the following cases:

**Case 1:** The number of cut-vertices in \( P \) is equal to one. (See Fig. 4(c) for Subcase 1-1 and Fig. 4(a) for Subcase 1-2)

- **Subcase 1-1:** There is exactly one cut-vertex in \( T_i \). Then \( T_i \) is a star centered at \( v \) and \( Y \backslash \{v, c_i\} \) can be partitioned into at most five parts. Thus, we can find at most four \( H_3 \)-paths and add them to \( Y \) to make \( v \) to be a good point.

- **Subcase 1-2:** There are at least two cut-vertices in \( T_i \). Then \( T_i \) is a tree such that every leaf-block is connected with \( V B \) by a cut-vertex. Let \( c \) be the cut-vertex for \( 1 \leq i \leq q \). Add several \( H_3 \)-paths to remove the pair of separator \( \{v, c_1\} \). Then check whether \( v \) and \( c_2 \) is still a separator, if so, add some \( H_3 \)-paths, \( \cdots \), continuing this process until \( v \) and \( c_q \) is no longer a separator. Since \( G_3 \) is a UDG, we have \( q \leq 5 \). By adding at most five \( H_3 \)-paths, \( v \) can be a good point (see Lemma 5.3).

**Case 2:** The number of cut-vertices in \( P \) is at least two. In this case, we first examine the leaf-root path \( P \) from \( B_1 \) to the root \( V B \). Let \( B \) be the first node (a block or a cut-vertex) in \( P \) with degree larger than two in \( T_i \). If such a node does not exist, we try to use a virtual cut-vertex \( c \) on the path as \( R \) (if exists). Otherwise, we just keep \( V B \) as \( R \). Therefore, \( R \) can be a block \( B_m \) (1 \( \leq m \leq k \)) or a cut-vertex \( c_m \) (2 \( \leq m \leq k \)) in \( P \), or can coincide with the root \( V B \) or a virtual cut-vertex \( c \) lying in \( V B \). After finding the new root \( R \), we modify the leaf-root path \( P \) given in Eq. (2) into a new leaf-root path with endpoint \( R \), still denoted as \( P \) with understood \( P \) ends at \( R \). Let \( P_i = P \backslash \{B_0, \cdots, c_i\} \).

Let \( LB(c_i) \) be the set of leaf-blocks being adjacent with \( \tilde{c}_i \) in \( T_i \). Note that the number \( q \) of leaf-blocks in \( LB(c_i) \) is at most four. Since \( v \) and \( \tilde{c}_1 \) constitutes a separator of \( Y \), \( Y \backslash \{v, \tilde{c}_1\} \) breaks into at most five parts, and there must exist an \( H_3 \)-path \( \Gamma_i \) connecting \( LB_i \backslash \{\tilde{c}_i\} \) with \( Y \backslash (LB_i \cup \{v\}) \) for each \( LB_i \in LB(\tilde{c}_i) \) (1 \( \leq i \leq q \)). Now consider \( B_1 \).

i) If \( R = B_1 \) (note \( R \) cannot be a virtual block since \( k \geq 2 \)), then adding \( \Gamma_i \) and applying Procedure-EEN below makes \( \tilde{c}_1 \) to be good.

ii) Next, suppose \( m \geq 2 \). Since \( \tilde{c}_1 \) and \( \tilde{c}_2 \) constitute a pair of separator of \( Y \), there must exist an \( H_3 \)-path \( \Gamma \) starting from \( B_1 \backslash \{\tilde{c}_1, \tilde{c}_2\} \) ending at \( Y \backslash B_1 \) with endpoint \( x \).

- **Subcase 2-1:** If \( x \) does not lie in \( P_2 \), then \( \tilde{c}_1 \) becomes good by adding \( H_3 \)-paths \( \Gamma \) (1 \( \leq i \leq q \)) and \( \Gamma \) then applying Procedure-EEN (Eliminating the External Nodes); see Lemma 5.6.

- **Subcase 2-2:** If \( x \) lies in \( P_2 \), say \( x = \tilde{c}_p \) or \( x \in B_2 \backslash \{\tilde{c}_p, \tilde{c}_{p+1}\} \) for some \( 2 \leq p \leq k \), then applying the Procedure-ECP (Eliminating a Cut-vertex on a Path) makes one of \( \tilde{c}_2, \cdots, \tilde{c}_{p-1} \) to be good by adding at most a constant number of \( H_3 \)-paths; see Lemma 5.5.

**Procedure-ECP.** Suppose that we are given a leaf-root path \( P \) as in Eq. (2), and there exists an \( H_3 \)-path \( \Gamma \) starting from \( B_1 \backslash \{\tilde{c}_1, \tilde{c}_2\} \) and ending at \( x \) which lies in \( P_2 \), we will use an iterative process to make one of the cut-vertices to be a good point.

i) If \( \Gamma \) ends at \( B_p \backslash \{\tilde{c}_p\} \), then we make one of \( \tilde{c}_2, \cdots, \tilde{c}_{p-1} \) to be good. Since \( \{\tilde{c}_2, \tilde{c}_3\} \) is a pair of separator of \( Y \), there exists an \( H_3 \)-path \( \Gamma' \) connecting \( B_2 \backslash \{\tilde{c}_2, \tilde{c}_3\} \) and \( Y \backslash B_2 \). If the endpoint \( y \) of \( \Gamma' \) does not lie on the path \( P_3 \backslash P_2 = \{B_3, \cdots, \tilde{c}_p\} \), then \( \tilde{c}_2 \) can be changed into a good point by adding \( \Gamma \) and \( \Gamma' \) and then applying Procedure-EEN. Otherwise we have \( y \) lies in \( P_3 \backslash P_2 \). Now set \( B_1 \leftarrow B_2 \) and \( x \leftarrow y \). Repeat the same process until we have changed one of the cut-vertices into good; see Fig. 5 and Lemma 5.5.

ii) The similar arguments can be applied if \( \Gamma \) ends at \( \tilde{c}_p \) or the virtual block \( V B \) (or \( c \)); we omit the details.

Finally, after connecting some good blocks \( B_i \) (which share a common cut-vertex \( \tilde{c} \)) by adding some \( H_3 \)-paths, we still worry about external nodes, which can constitute separators with the cut-vertex \( \tilde{c} \). The following Procedure-EEN (Eliminating the External Nodes) is for this purpose.

**Procedure-EEN.** Let \( B_i \) (1 \( \leq i \leq r \)) be good blocks in \( T_i \) with cut-vertex \( \tilde{c} \) connecting them. Suppose that each of \( B_i \) is not a virtual block and they can be connected by some paths without going through \( v \) after adding some \( H_3 \)-paths. For each external node \( u \in \cup_{i=1}^r E(B_i) \), check whether \( \{u, \tilde{c}\} \) is a separator of \( Y \), if so, adding an \( H_3 \)-path to eliminate this pair of separator, until any external node \( u \) cannot constitute a separator of \( Y \) with \( \tilde{c} \). It can be shown that at most \( H_3 \)-paths is needed to finish this step; see Lemma 5.4.

Algorithm 1 is the formal description of our algorithm. In the subsequent sections, we show the correctness proof, performance analysis, and time complexity of this algorithm.
We show \( H_{B} \) is 2-connected and the running time of our algorithm is polynomial. Solution, FT-CDS-CA can correctly generate a path contained in \( B_{m} \) connecting \( c_{m-1} \) and \( c_{m} \). If \( u \) and \( w \) is not contained in any of \( B_{m} \), then our assertion clearly holds. Now, suppose that \( u \) is contained in, say, \( B_{m'} (= B_{1}) \). Since \( B_{m'} \) is 2-connected, there is a path \( P_{m'} \) (which is independent of \( P_{m''} \)) in \( B_{m'} \) connecting \( c_{m'-1} \) and \( c_{m'} \). If \( w \) is also contained in some of \( B_{m''} \) (\( m' \neq m'' \)), then there exists a path a path \( P_{m''} \) (which is independent of \( P_{m''} \)) in \( B_{m''} \) connecting \( c_{m''-1} \) and \( c_{m''} \). Replace \( P_{m'} \) with \( P_{m''} \) in the original path. Then \( u \) and \( w \) is connected to each other in \( H \setminus \{v\} \). Hence, the lemma holds.

**Lemma 5.2:** After the merging process, there are exactly one virtual block left in \( T'' \) for \( l > 0 \).

**Proof:** Suppose after level-\( l \) decomposition, we get a leaf-block tree \( T_{l} \) with all blocks being good, and the vertices of which are given as in Eq. (1).

Construct a sub-tree \( T' \) as follows: Let \( B'_1, B'_2, \ldots, B'_{m} \) be the blocks each of which has external nodes that are adjacent to \( Y \setminus B \). If \( m = 1 \), i.e., there is exactly one block having external nodes connected to nodes outside \( B \) directly, then \( V_{B} \) is just block \( B' \). Next, suppose \( m \geq 2 \). Then \( V(T_{l}) \) consists of those blocks and cut-vertices which lying on the path between any two \( B'_i \) and \( B'_j \) for \( i \neq j \).

Examine the leaf-blocks of \( T_{l} \). They can be partitioned into two classes \( L_{1} \) and \( L_{2} \): \( L_{1} \) consists of those having exactly one cut-vertex as external node that are adjacent to \( Y \setminus B \) (i.e., virtual cut-vertex); \( L_{2} \) consists of those having at least one external node that are not cut-vertex and adjacent to \( Y \setminus B \). Then, according the rules of merging process, after removing all leaf-blocks in \( L_{1} \) from the blocks in \( T_{l} \), the remaining blocks can be merged into a unique block \( V_{B} \).

**Lemma 5.3:** After Line 2 of Algorithm 4 are executed, is executed, then \( v \) becomes a good-point by adding at most five \( H_{3} \)-paths.

**Proof:** Suppose after level-\( l \) decomposition, and we obtain a leaf-block tree \( T_{l} \) from \( B \setminus \{v\} \) with vertex set given as in Eq. (1) whose blocks \( B_{i} \) are all good. \( P \) is the longest leaf-root path in \( T_{l} \) with root which may ends at the virtual cut-vertex \( c \) or the virtual block \( V_{B} \). In any case, since \( P \) is the longest leaf-root path with one cut-vertex, \( T_{l} \) must be a tree like a star (centered at the virtual cut-vertex), or a tree (centered at the virtual block \( V_{B} \)) with all blocks being leaf-
blocks except for the center $VB$.

In the former case, since $\{v, c\}$ is a pair of separator of $Y$, $Y \setminus \{v, c\}$ breaks into at most five parts, and hence at most four $H_3$-paths are needed to add to $Y$ to reconnect them.

In the later case, let $c_i (1 \leq i \leq q)$ be the cut-vertices in $T_i$ which are adjacent to $l_i$ leaf-blocks. Then $B \setminus \{v, c_i\}$ breaks into at most $l_i + 1$ components and $l_i$ $H_3$-paths are needed to remove the separator $v$ and $c_i$. Note that $v$ is adjacent to all the leaf-blocks, which are pairwise disjoint. Since $G_3$ is a UDGs, every nodes have at most five independent neighbors. It follows that $l_1 + l_2 + \cdots + l_q \leq 5$. Thus, totally at most five $H_3$-paths are needed to remove all separators $\{v, c_i\}$.

Note that $v$ is an internal node, it follows from Lemma 5.1 that $v$ can constitute a separator only with the cut-vertices $c_i$ inside the block $B$. After adding at most five $H_3$-paths, $v$ cannot constitute separators of $Y$ with any $c_i$, therefore $v$ becomes good.

**Lemma 5.4:** At most five $H_3$-paths are needed to finish Procedure-EEN.

**Proof:** After adding several $H_3$-paths, $\tilde{c}$ can constitute separators only with nodes in $B_i$ for $1 \leq i \leq r$.

Since after the multi-decomposition process finished, $B_i$ has no bad internal points, $\tilde{c}$ can constitute separators only with those nodes in $\cup_{e=1}^r E(\tilde{B}_i)$. Therefore, remove every pair of separators $\{\tilde{c}, u\}$ for all $u \in \cup_{e=1}^r E(\tilde{B}_i)$ makes $\tilde{c}$ be good.

Now we show at most five $H_3$-paths are needed in this step. Let $\tilde{T}_i$ be the leaf-block tree of $B_i \setminus \{\tilde{c}\}$, then $\tilde{c}$ can constitute separators only with those cut-vertices in $\tilde{T}_i$. However, we note that every cut-vertex $\tilde{c}$ in $\tilde{T}_i$ except for those that are adjacent to the leaf-blocks, cannot constitute a separator with $\tilde{c}$. This is because $\tilde{c}$ are external nodes which are adjacent to nodes in $Y \setminus B_i$. Thus, the number of nodes in $E(\tilde{B}_i)$ which makes a pair of separator with $\tilde{c}$ is no more than the number of leaf-blocks in $B_i$.

Since $v$ can be adjacent to at most five leaf-blocks, the total number of separators $\{\tilde{c}, u\}$ for all $u \in \cup_{e=1}^r E(\tilde{B}_i)$ is at most five. Removing a pair of separator needs at most one $H_3$-path, and hence five $H_3$-paths are needed to finish this step. ■

**Lemma 5.5:** After Procedure-EEN is executed, at least one cut-vertex $\tilde{c}_1$ can be changed into a good point by adding at most seven $H_3$-paths.

**Proof:** We prove the lemma by induction on the number of blocks $p$. If $p = 2$, then $\tilde{c}_2$ can be changed into a good point by adding $H_3$-path $\Gamma$ and applying Procedure-EEN, which needs at most $1 + 5 = 6$ $H_3$-paths. Suppose that the lemma is true if the number of blocks is no more than $p - 1$.

If $B_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ and $B_p$ are connected by an $H_3$-path $\Gamma$, then $\tilde{c}_2$ can constitute a separator of $Y$ with only $\tilde{c}_3, \ldots, \tilde{c}_p$, and some of those external nodes of $B_1$ and $B_p$ (which we have ignored in the multi-decomposition process). Now, since $\tilde{c}_2$ and $\tilde{c}_3$ constitute a pair of separator of $Y$, there exists an $H_3$-path $\Gamma'$ connecting $B_2 \setminus \{\tilde{c}_2, \tilde{c}_3\}$ and $Y \setminus B_2$. If the endpoint $y$ of $\Gamma'$ does not lie on the path $P_3 \setminus P_p = \{B_3, \ldots, B_p\}$, then $\tilde{c}_2$ cannot constitute a separator of $Y$ with any of $\tilde{c}_3, \ldots, \tilde{c}_p$, then after applying Procedure-EEN, $\tilde{c}_2$ cannot constitute a separator of $Y$ with those external nodes of $B_1$ and $B_2$. Therefore, $\tilde{c}_2$ can be changed into a good point by adding two $H_3$-paths $\Gamma$ and $\Gamma'$ and then applying Procedure-EEN, which needs at most $2 + 5 = 7$ $H_3$-paths in total.

Otherwise we have $y$ lies in $P_3 \setminus P_p$. By induction hypothesis, we get that one of the cut-vertices $\tilde{c}_3, \ldots, \tilde{c}_{p-1}$ can be changed into a good point.

Similarly, if $B_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ and $\tilde{c}_p$ (or $VB$ or $c$) are connected by an $H_3$-path $\Gamma$, we can still get either $\tilde{c}_2$ or one of $\tilde{c}_3, \ldots, \tilde{c}_{p-1}$ can be changed into a good point.

**Lemma 5.6:** After Line 13 of Algorithm 4 are executed, at least one $\tilde{c}_i$ becomes a good point by adding at most ten $H_3$-paths.

Moreover, Applying Procedure-EEN removes all separators $\{\tilde{c}_1, u\}$ for every external nodes $u$ lying the blocks being adjacent to $\tilde{c}_1$. Therefore, at most $4 + 1 + 5 = 10$ $H_3$-paths are needed to make $\tilde{c}_1$ to be good.

Similarly, if we confront with Case 2 Subcase 2-1. Note $\tilde{c}_1$ can constitute a separator only with $v$, the cut-vertices lying in the leaf-root path $P$ and the external nodes of those blocks being adjacent to $\tilde{c}_1$. However, after adding at most four $H_3$-paths $\Gamma_i$, $\tilde{c}_1$ and $v$ cannot constitute a separator of $Y$. Adding $\Gamma$ with endpoint lying outside path $P_2$, $\tilde{c}_1$ cannot constitute a separator with the cut-vertices lying in the leaf-root path $P$ any more.

Theorem 5.8 ([17]): There exists a polynomial time approximation algorithm $A$ which can generate an approximated solution for $(2, m)$-CDS with performance ratios $(5 + \frac{23}{m})$ for $2 \leq m \leq 5$ and $11$ for $m > 5$.

**Lemma 5.9:** Given a $G_3$, let $H$ be a connected subgraph containing a $C_{2,3}$ of the $G_3$. Then, adding a $H_3$-path $P$ to $H$ does not introduce a new bad-point into $H$.

**Proof:** To prove this, we consider following two cases in which the length of $P$ is two or three hops. In the first case, $P$ includes only one new node $u$. Then, no node in $H$ can constitute a separator with $u$ in $H \cup \{u\}$ since $H$ is 2-connected. Therefore, all nodes in $P$ are good-points in $H \cup \{u\}$. Now, we consider the second case in which $P$ has two new points $u$ and $v$. Now, we claim that $u$ is not a bad-point in $H \cup \{u, v\}$. Suppose $u$ is a bad-point and it constitute a separator with another node $w \in H \cup \{v\}$. Clearly, $w \neq v$ since $H$ is 2-connected. On the other hand, since $H$ is 2-connected and $v$ has at least three neighbors in $H$, $H \cup \{v\}$ is at least 2-connected. Therefore, $(H \cup P) \setminus \{u, w\}$ is connected for any $w \in H$. Therefore, $u$ is not a bad-point and by the same argument, $v$ is not a bad-point. In conclusion, the lemma is true.

**Lemma 5.10:** The time complexity of Algorithm 1 is at most $O(n^4)$, where $n$ is the order of the input graph $G_3$. 
Proof: By [17], \( Y = C_{2,3} \) can be computed in \( O(n^3) \). Given \( Y \), all of its bad points \( X \) can be computed in \( O(|Y|^3) = O(n^3) \), since for every vertex \( v \) we can determine whether \( Y \setminus \{v\} \) is 2-connected in time \( O(n^2) \). Now we estimate the time of eliminating one bad point from \( X \).

In order to eliminating one bad point, we have to find a block \( B \) with cut vertex \( v \in B \) such that \( B \setminus \{v\} \) can be decomposed into good blocks \( \{B_1, B_2, \ldots, B_z\} \) and cut vertices \( \{c_1, c_2, \ldots, c_y\} \). This can be achieved by multi-level decomposition. Since it takes time \( O(n^2) \) to construct a leaf-block tree at each level [41], in the worst case, we have at most \( n \) levels and need time at most \( O(n^3) \) to finish this step. The merging process also takes time at most \( O(n^3) \) by Floyd’s algorithm for computing shortest path between any pair of nodes.

Once \( B \) is found. We have to change either one of the bad points in \( \{c_1, c_2, \ldots, c_y\} \) or \( v \) into a good point by adding several \( H_3 \)-paths. This step takes time \( O(n^3) \) since finding a \( H_3 \)-path between two connected components of a graph takes time \( O(n^2) \) and we have to try at most \( O(\log(n)) \) times to find such \( H_3 \)-paths to eliminate one bad point.

There are \(|X| = O(n)| \) bad points in total. Therefore, the complexity of Algorithm 1 is at most \( O(n^4) \).

Theorem 5.11: Algorithm 1 is a 280-approximation for 3-Connected 3-Dominating Set problem. Proof: From Theorem 5.8, we have an \( r \)-approximation algorithm for computing a \( C_{2,3} \), where \( r = \frac{40}{3} \). Then, we can have a \( C_{2,3} \) such that \(|C_{2,3}| \leq r|^{opt}_{C_{2,3}}|\). In Algorithm 1, since \( X \subseteq C_{2,3} \), \(|X| \leq |C_{2,3}|\). From Lemma 5.7, Algorithm 1 will use at most \(|20 |X| \) nodes to augment the \( C_{2,3} \) to a \( Y = C_{3,3} \). As a result, the size of final \( Y \) is bounded by \(|Y| = |C_{2,3}| + 20|X| \leq r|C^{opt}_{2,3}| + 20|C_{2,3}| \leq r|C^{opt}_{2,3}| + 20r|C_{2,3}| \leq 21r|C^{opt}_{2,3}| \leq 21r|^{opt}_{C_{3,3}}|\). C. Generalization for any \( m \neq 3 \)

When \( m > 3 \), we first compute a \( C_{2,m} \) using the existing algorithm in [17],[18]. Then, we augment \( C_{2,m} \) to \( C_{3,m} \) using Algorithm 1. Now, we prove that the size of the outputs by this strategy is within a constant factor from an optimal solution even in the worst case.

Theorem 5.12: The approximation ratio of this strategy is \( 21r \) for \( m > 3 \), where \( r = \frac{5 + \frac{22}{m}}{3} \) for \( 3 \leq m \leq 5 \) and 11 for \( m > 5 \).

Proof: For \( m > 3 \), it is easy to show that \(|Y| \leq 21r|^{opt}_{C_{3,m}}|\) using the argument in the proof of Theorem 5.11.

When \( m = 1,2 \), We start from a \( C_{1,3} \), then augment \( C_{1,3} \) to \( C_{2,3} \). Both can be computed by the existing method in [17],[18]. Finally, augment \( C_{2,3} \) to \( C_{3,3} \) using Algorithm 1. Now, we evaluate the worst case quality of outputs of this approach.

Theorem 5.13: The approximation ratio of this strategy is \( 17 \cdot 2 \cdot 21 \) for \( m = 1,2 \).

Proof: First, we focus on the case \( m = 1 \), and show that the obtained \( C_{3,3} \) is within a constant factor from \( C^{opt}_{3,3} \). In fact, by the Algorithms in [10], [11], we have \( C_{1,3} = I_1 \cup I_2 \cup I_3 \cup C \), where \( I_1 \) is the maximal independent set obtained sequentially and \( C \) is some additional nodes added to make \( I_1 \) to be connected. Since \(|I_i| \leq 5|^{opt}_{C_{1,3}}| \) (\( i = 1, 2, 3 \)) and \(|C| \leq 2|I_1| \), we have \(|C_{1,3}| \leq 17|^{opt}_{C_{1,3}}|\). Moreover, by the proofs in [10], [11], we have \(|C_{2,3}| \leq 2|C_{1,3}|\). By Algorithm 1, \(|C_{3,3}| \leq 21|C_{2,3}| \leq 2 \cdot 21|C_{1,3}| \leq 17 \cdot 2 \cdot 21|^{opt}_{C_{1,3}}|\). Note that \(|C_{1,3}| \leq |^{opt}_{C_{3,3}}|\). It follows that \(|C_{3,3}| \leq 17 \cdot 2 \cdot 21|^{opt}_{C_{3,3}}|\).

For \( m = 2 \), the same algorithm as above can be applied. And the approximation ratio can be obtained similarly by noting that \(|^{opt}_{C_{1,3}}| \leq |^{opt}_{C_{3,3}}|\).

By combining Theorem 5.11, Theorem 5.12, and Theorem 5.13, we have the following conclusion.

Theorem 5.14: There exists an \( O(1) \)-approximation algorithm for computing \((3,m)\)-CDS in UDGs for any \( m \).

VI. Simulation Results

In this section, we study the average performance and characteristics of our proposed algorithm via simulations. For the simulation, we randomly generate a set of nodes over a virtual space. The number of nodes varies from 40 to 200 increased by 20. The size of the virtual space is 25 by 25, 50 by 50, and 75 by 75. For each graph instance, we check if the graph is 3-connected. Otherwise, we discard it and generate a new one. For each parameter setting, we create 100 (3-connected) graph instances, and apply the constant factor approximation algorithm for \((2,3)\)-CDS by Shang et al. [17]
and our (3,3)-CDS computation algorithm, FT-CDS-CA, and calculate the average size of CDSs. Each of Fig. 6(a), Fig. 6(b), and Fig. 6(c) is the result of our simulation performed over a 25 by 25, 50 by 50, and 75 by 75 virtual space, respectively. Largely speaking, from all of the three figures, we can notice that the average size of (3,3)-CDS is roughly 20%-30% greater than that of (2,3)-CDS. In addition, we can observe that the ratio is getting smaller as the number of nodes in the network increases (roughly 30% with n = 40 and 20% with n = 200). Therefore, as the network size grows, our algorithm becomes more efficient to increase the connectivity of CDS. The figures also show that as n increases, we need more nodes to augment a (2,3)-CDS to (3,3)-CDS. This is because if the nodes are deployed within a smaller space, a large part of a (2,3)-CDS is already connected. From this observation, we can also expect that our algorithm will work more efficiently in a very dense network.

In Fig. 7, we compare the average size of (3,3)-CDS generated by FT-CDS-CA under various virtual space size and number of nodes. In general, when the size of the virtual space is smaller, the size of (3,3)-CDS is smaller. This is natural since with the same number of nodes within a smaller space, the density and connectivity of the network is usually higher, and we consequently will need less nodes to form a (3,3)-CDS. However, the simulation result also implies the affect of the size of network is not significant. In Fig. 7, we compare the average size of (3,3)-CDS generated by FT-CDS-CA under various virtual space size and number of nodes. In general, when the size of the virtual space is smaller, the size of (3,3)-CDS is smaller. This is natural since with the same number of nodes within a smaller space, the density and connectivity of the network is usually higher, and we consequently will need less nodes to form a (3,3)-CDS. However, the simulation result also implies the affect of the size of network is not significant.

Running Time Analysis. Fig. 8 illustrates the running time of FT-CDS-CA to compute (3,3)-CDS under various virtual space size. In Fig. 8, we compare the running time of FT-CDS-CA under various virtual space size. The running time of FT-CDS-CA is increased proportionally. Our simulation result also indicates that the running time of our algorithm is mainly affected by the number of nodes, but not significantly by the size of the virtual space.

Optimality Analysis. Finally, we evaluate the performance of Shang et al.’s algorithm in [17] and FT-CDS-CA against the optimal (2,3)-CDS and (3,3)-CDS, respectively. Note that since the problems of computing optimal (2,3)-CDS and (3,3)-CDS are NP-hard, we were able to perform these comparisons only in very small size networks. In detail, we prepare a 10 by 10 virtual space, deploy 15 nodes, and generate a 3-connected unit disk graph. Then, we exhaustively search the optimal (2,3)-CDS and (3,3)-CDS, and compare them with the (2,3)-CDS by Shang et al.’s algorithm and the (3,3)-CDS by FT-CDS-CA, respectively. After 30 repetition, we compute the average.

Fig. 9 shows the result of this simulation. In this figure, “Approx” is the average size of the output of Shang et al.’s algorithm in case of (2,3)-CDS computation and that of FT-CDS-CA’s in case of (3,3)-CDS computation, respectively. Also, “OPT” means the average size of optimal solutions for each problem. From the simulation, we found the average size of optimal (2,3)-CDS is 4.8 and the average size of (2,3)-CDS by Shang et al.’s is 7.10. In addition, we found the average size of optimal (3,3)-CDS is 5.27 and the average size of (3,3)-CDS by FT-CDS-CA is 10.57. Our simulation result indicates that the average (experimental) performance ratio of our algorithm, based on this simulation, is 2, which is much smaller than the performance ratio of FT-CDS-CA that we were able to prove.

VII. Conclusion

This paper investigate the problem of constructing fault-tolerant CDS in homogeneous wireless networks, which is abstracted as the minimum k-connected m-dominating set problems. In our recent surveys, we pointed out that each of existing approximation algorithm for this problem is flawed for k ≥ 3. Therefore, we propose a constant factor polynomial time approximation algorithm to compute (3, m)-CDSs. Our algorithm works for any abstract graph without the information...
of geometric coordinates of the input graphs, and we only use the property of UDG in the analysis part to get a constant approximation. As a future work, we are interested in generalizing our algorithm for any $k \geq 4$ and $m \geq 1$ pair. We will also investigate a constant factor approximation algorithm for $k \geq 3$ and $m \geq 1$ in disk graph. In addition, we plan to continue study the problem of computing quality fault-tolerant virtual backbone in more realistic network abstractions as our future work.

VIII. ACKNOWLEDGEMENT

The research was supported in part by NSFC-11071191. This work was supported in part by NSF CREST No. HRD-0833184. This work was supported in part by US ARO No. W911NF-0810510. The research was also supported in part by NSFC-61073046 and NSFC-11201208. This research was also supported in part by US NSF under Grant CNS-0831579, CCF-0829993. Finally, the authors would like to appreciate the time and effort of the editors and reviewers for their comments to improve the quality of this paper.

REFERENCES

Wei Gao is currently an undergraduate student at Department of Information and Computational Science at Xi’an Jiaotong University, China. His research interests include algorithms for routing in wireless ad hoc networks.

Wei Gao

Xianyue Li received the BS and PhD degrees from Lanzhou University, China, in 2003 and 2009, respectively. He is currently an assistant professor in the School of Mathematics and Statistics at Lanzhou University, China. His research interests include wireless networks, graph theory and approximation algorithm design and analysis.

Xianyue Li

Zhao Zhang is a full professor in College of Mathematics and System Sciences at Xinjiang University. She received her PhD in 2003 from Xinjiang University. Her current research interests are in combinatorial optimization and graph theory.

Zhao Zhang

Wei Li received the PhD degree in computer science from the University of Minnesota in 2002. Her main research interest is in database systems. She has produced a number of research papers in spatial data mining, distributed database systems, and algorithm design. She was also a co-editor of a book on clustering and information retrieval. She is currently an associate professor in the Department of Computer Science at the University of Texas at Dallas. She is a member of the IEEE.

Wei Li

Wei Wang received the BS degree from Zhejiang University, China, in 1991 and MS degree and PhD degree from Xi’an Jiaotong University, China, in 1994 and 2006 respectively. He is currently an assistant professor in the Department of Mathematics at Xi’an Jiaotong University. His research interests include algebraic graph theory and approximation algorithm design and analysis.

Wei Wang

Donghyun Kim received the BS degree in electronic and computer engineering from the Hanyang University, Ansan, Republic of Korea in 2003, and the MS degree in computer science and engineering from Hanyang University, Republic of Korea in 2005. He received the PhD degree in computer science from the University of Texas at Dallas, Richardson, USA in 2010 under the supervision of Prof. Ding-Zhu Du and Prof. Weili Wu. His research interests include wireless networks, mobile computing, distributed computing, approximation algorithm design and analysis. He is currently an assistant professor in the Department of Mathematics and Computer Science at the North Carolina Central University, Durham, USA. He is a member of the IEEE.

Donghyun Kim

Min Kyung An received her MS in Computer Science from the University of Texas at Arlington, Arlington, USA in 2007. She began her PhD studies in the Department of Computer Science at the University of Texas at Dallas, Richardson, USA in 2008. Her major areas or interests are wireless ad hoc and sensor networks, design and analysis of approximation algorithm, and graph theory.

Min Kyung An

Wei Li

Wei Li

Wei Li

Wei Li