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 a 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 a 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 that 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, connected dominating set, fault tolerance, graph theory, virtual backbone.**

I. Introduction

A WIRELESS network consists a set of computing devices referred to as *wireless nodes*, which are connected with each other using wireless communication technology. During recent years, two particular types of infrastructureless 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 infrastructureless wireless networks by "wireless networks" in short. 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 a 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: 1) every node in a given wireless network is either in C or adjacent to a node in C; and 2) 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 Kuller 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 research has been conducted on the MCDS problem and its variations [8]–[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 an environment. That is, once the topology is changed, the structure may be disconnected and needed to be reconstructed. Conversely, 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.

1) *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

Manuscript received August 02, 2011; revised May 01, 2012; September 10, 2012; and October 17, 2012; accepted October 30, 2012; approved by IEEE/ACM Transactions on Networking Editor S. Ramasubramanian. Date of publication December 11, 2012; date of current version October 11, 2013. This work was supported in part by the NSFC under Grants NSFC-11071191, NSFC-61073046, NSFC-11201208, and NSFC-61222201; the NSF CREST under Grant No. HRD-0833184; the US ARO under Grant No. W911NF-0810510; and the US NSF under Grants CNS-0831579, CNS-1016320, and CCF-0829993.

W. Wang and W. Gao are with the Department of Mathematics, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: wang_weiw@163.com; w_gao@foxmail.com).

D. Kim is with the Department of Mathematics and Computer Science, North Carolina Central University, Durham, NC 27707 USA (e-mail: donghyun.kim@nccu.edu).

M. K. An and W. Wu are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083 USA (e-mail: mka081000@utdallas.edu; weiliwu@utdallas.edu).

X. Li is with the School of Mathematics and Statistics, Lanzhou University, Lanzhou 730000, China (e-mail: lixianyue@lzu.edu.cn).

Z. Zhang is with the College of Mathematics and System Sciences, Xinjiang University, Urumqi 830046, China (e-mail: hxhzz@163.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2012.2227791

2) *m-domination*: A subset C of a graph G is an 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 *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]–[19], [26]–[28], [33]. However, as we showed in our previous surveys [34], [35], there exists no constant factor approximation algorithm that 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 a 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. 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 IV. Our simulation result is given in Section V. We conclude our paper and present future works in Section VI.

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 in the following, to abstract the networks. Let $\mathrm{udist}(u,v)$ be the euclidean distance between u and v. 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 $\mathrm{udist}(u,v) \leq 1$. A subset $I \subset V$ is an *independent set* (IS) of G if $\forall u, v \in I$, $(u,v) \notin E$. A subset $M \subset V$ 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. A subset $D \subseteq V$ is a *dominating set* (DS) of G if $\forall u \in V$, either $u \in D$ or $\exists v \in D$ such that $(u,v) \in E$. 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: 1) compute an MIS M using a simple 2-coloring algorithm; and 2) 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 $|M| \leq d|\mathrm{OPT}_{\mathrm{CDS}}|$ for some constant d, where $\mathrm{OPT}_{\mathrm{CDS}}$ is an optimal CDS, the above strategy is in fact an approximation of the MCDS problem whose performance ratio is $c \cdot d$.

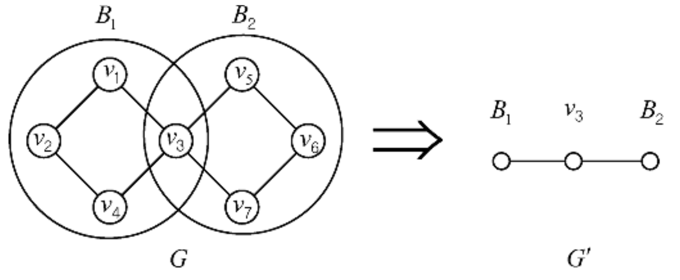


Fig. 1. This figure illustrates how *(right)* a leaf-block graph can be computed from *(left)* a given 2-connected graph.

This approach is initially introduced by Guha and Kuller [6] and later becomes a standard approximation technique for the MCDS problem and its variations.

A DS D is a *m-dominating set* (m-DS) of G if $\forall u \in (V \setminus D)$, u is a neighbor of at least m nodes in D. 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 this paper, "k-vertex-connected" and "k-connected" will be used interchangeably. In addition, G_k will imply a k-connected graph. A subset D is a *k-connected m-dominating set* of G if the subgraph induced by D is k-connected and D is an m-dominating set of V. In the rest of this 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 notate (k,k)--DCS. Finally, $C_{k,m}^{\mathrm{opt}}$ is an optimal (k,m)-CDS. This paper assumes that the UDG induced from the input wireless network is $\max\{m,k\}$-connected. However, how to obtain such a wireless network 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 nodes remains the same during the computation of a (k,m)-CDS. A vertex $u \in V$ is a *cut-vertex* of G only if the graph induced by $V \setminus \{u\}$ is disconnected. A subgraph of G is called a *block* only if it is a maximal connected subgraph of G without any cut-vertex. Every block in a connected graph is either a maximal 2-connected subgraph or a *bridge*, which is an edge with two endpoints. Given a connected subgraph G that 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: 1) For each block B of G, add a corresponding node v_B to G'; 2) for each cut-vertex u of G, add a node u to G'; and 3) 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. Fig. 1 illustrates how a leaf-block graph can be constructed. In the rest of this paper, a *subblock* refers a block in a leaf-block graph of a block. A *leaf block* is a block that has degree one in the leaf-block graph. We would like to draw the attention to following three facts: 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: 2) every edge of G lies in a unique block. That is, no two blocks share the same edge; and 3) G is the union of its blocks. Based on the facts, the following proposition naturally follows.

Proposition 2.1 ([38]): The leaf-block graph of a connected graph is a tree (i.e., a connected graph).

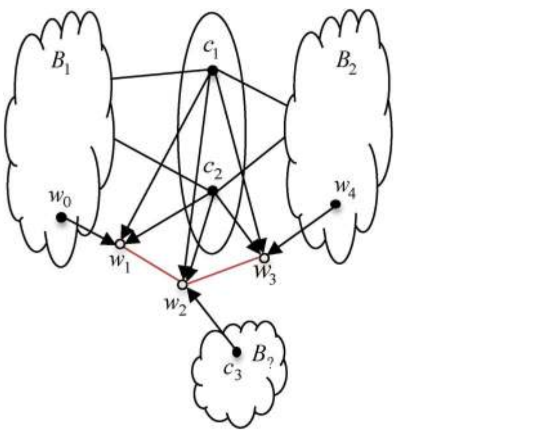


Fig. 2. Given a G_3, a $C_{2,3} \subset G_3$, and a separator (c_1, c_2) of $C_{2,3}$, suppose $C_{2,3} \setminus \{c_1, c_2\}$ is divided into several connected components, $\{B_1, B_2, \cdots\}$. Then, there have to be two different components B_i and B_j such that the length of the shortest H-path between them is at most three hops.

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. 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). 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 2.2: 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 2.3: 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, $\{B_1, B_2, \cdots\}$. 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 $\{w_0, w_1, \cdots, w_l\}$ 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 $i = 1, 2, \cdots, l-1$ (see Fig. 2).

Now, consider the node w_2. We find that w_2 can be dominated by neither B_1 nor B_2; 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 2.4: 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 2.5 (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 seminary 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 cases, 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. Afterwards, 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 that 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. Constant Factor Approximation for Computing (3,m)-CDS in UDGs

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 in this section with $m = 3$ and later explain how our result can be generalized for any $m \neq 3$. Roughly, FT-CDS-CA works as follows. Given a G_3, FT-CDS-CA first computes a $C_{2,3} \subseteq G_3$ using a constant factor approximation algorithm in [17], [18] and set $Y \leftarrow C_{2,3}$. Next, the algorithm identifies the set of all bad-points X in $C_{2,3}$. Note that $|X| \leq |C_{2,3}|$. By Lemma 2.2, the 2-connected subgraph $C_{2,3}$ is 3-connected if $X = \emptyset$. The core strategy of FT-CDS-CA is that the algorithm repeatedly changes each bad-point $v \in X$ into a good-point by moving at most some constant number of nodes from $G_3 \setminus Y$ to Y without introducing any new bad-point into X. In this way, after changing all bad-points in X into good, Y becomes a 3-connected 3-dominating set, and accordingly, the total number of nodes newly added to $C_{2,3}$ is bounded by a constant factor of $|C_{2,3}|$.

Algorithm 1: FT-CDS-CA (G_3)

```
 1: Compute a $C_{2,3}$ and set $Y \leftarrow C_{2,3}$.
 2: Identify the set of bad-points $X$ in $Y$.
 3: while $X \neq \emptyset$ do
 4:     /* Step 1: Multilevel Decomposition */
 5:     $(T_l, v, l) \leftarrow \mathrm{MLD}(X, Y, Y, v, 0)$, where $v$ is one
        bad-point of $Y$.
 6:     if $l = 0$ then
 7:         $T'_l \leftarrow T_l$.
 8:     else
 9:         /* Step 2: Merging SubBlocks */
10:         $T'_l \leftarrow \mathrm{MSB}(X, Y, T_l, v)$.
11:     end if
12:     /* Step 3: One Bad-Point Elimination */
13:     $(X, Y) \leftarrow \mathrm{OBPE}(G_3, X, Y, T'_l, v)$
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 $C_{2,3}$) and the set X of bad-points in Y, we select $v \in X$ as a root and compute a leaf-block tree T_0 of $Y \setminus \{v\}$. Then, T_0 consists of a set of blocks $\{B_1, B_2, \cdots, B_s\}$ and a set of cut-vertices $\{c_1, \cdots, c_t\}$. Clearly, v can constitute a separator only with another node in $\{c_1, \cdots, c_t\}$. To simplify our presentation, we classify the nodes in each block B_i into the following two classes: 1) *internal nodes* $I(B_i) := \{w \in B_i | (w, u) \notin E, \forall u \in Y \setminus B_i\}$, i.e., the nodes that cannot be adjacent to any node outside B_i, and 2) external nodes, i.e., the remaining nodes $E(B_i) := B_i \setminus I_i$, which consist of those *spy-nodes* $S_i := \{u \in B_i | (u, v) \in E\}$ and those cut-vertices $C_i := B_i \cap \{c_1, c_2, \cdots, c_t\}$ 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 troublemakers 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 $1 \leq i \leq s$, then FT-CDS-CA makes either one of c_k for $1 \leq k \leq t$ or v to be a good-point by adding constant number of H_3-paths such that $Y \setminus \{c_k\}$ or $Y \setminus \{v\}$ is still 2-connected (Algorithm 4). Otherwise, there must exist some B_i having an internal node $w \in B_i$ which constitutes a separator $\{w, u\}$ of Y with another node $u \in Y$. By Lemma 4.1, any internal bad-point w in B_i can constitute a separator only with another node in B_i (while this may not be true for the external nodes such as cut-vertices and spy-nodes). It follows that $u \in B_i$. Note that by switching w to v, we can make the original problem smaller. By repeating such a 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 constitute "local separators" with c_k), we simply add H_3-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) multilevel decomposition; 2) merging subblocks; and 3) one bad-point elimination.

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
```

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 4.10 for details).

1) Multilevel 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 $\{B_1, \cdots, B_s\}$ and a set of cut-vertices $\{c_1, c_2, \cdots, c_t\}$. Observe that after the level-0 decomposition, $s \geq 2$ and $t \geq 1$.

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 4.1, we only need to check if there is an internal node $w \in I_i \subset B_i$ that 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 the next level (level-1) decomposition process on B. If such a 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 that 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_l, v)

```
1: Copy $T_l$ to $T'_l$.
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 $VB$ (no virtual cut-vertex).
4:     Return $T'_l$.
5: end if
6: repeat
7:     boolMerged ← FALSE
8:     for each pair of two different $B_i$ and $B_j$ of $T'_l$ do
9:         if: 1) each of $B_i$ and $B_j$ contains an external node $x$ and $y$ of $T'_l$, respectively, and 2) $x$ and $y$ are connected through a path in $Y \setminus (T_l \cup \{v\})\}$ then
10:            if $x$ and $y$ are the cut-vertices of $T'_l$ then
11:                Merge all the blocks on the path from $x$ to $y$ in $T'_l$ 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 $T'_l$ then
15:                Merge all the blocks on the path from $B_i$ to $B_j$ in $T'_l$ (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 $T'_l$ (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 $VB$ and virtual cut-vertex (or vertices) if applicable.
27: Return $T'_l$
```

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

$$V(T_l) = \{B_1, B_2, \cdots, B_s\} \cup \{c_1, \cdots, c_t\} \quad (1)$$

and every B_i is a good block for $1 \le i \le s$. 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.

1) 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_l 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 $B \setminus \{v\}$.

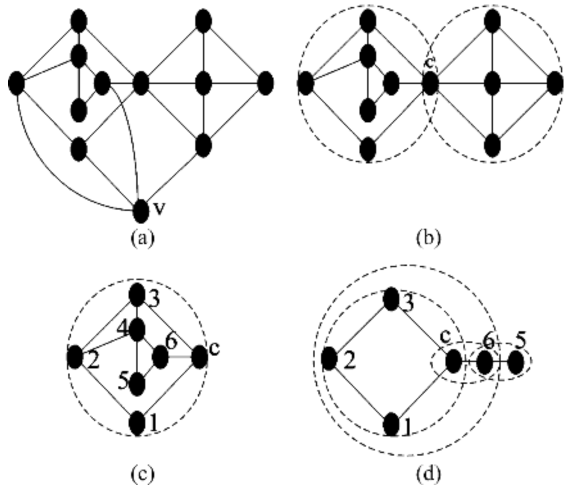


Fig. 3. Multilevel decomposition and merging of blocks. (a) Given 2-connected $C_{2,3}$ of a graph G_3; (b) leaf-block tree of level-0 decomposition with root v; (c) bad block in the level-0 decomposition tree with cut-vertices $\{c\}$, spy-nodes $\{1, 2, 6\}$, internal nodes $\{3, 4, 5\}$, and an internal bad node 4; (d) leaf-block tree of level-1 decomposition with root 4, where two blocks have been merged into a bigger virtual block.

2) 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_l 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).
3) If exactly one of x and y is a cut-vertex, i.e., $x = c_i$ is a cut-vertex and $y \in B_i$ 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'_l in which one bigger block VB (we call it a *virtual block*) is added representing all the merged blocks, and all the cut-vertices c_i that 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 4.2, 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_l (now T_l is a path).

3) One Bad-Point Elimination (OBPE): At this point, we have a leaf-block tree T_l as in (1), which is obtained from $B \setminus v$. Suppose that T_l, $l > 0$ 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 $s \ge 2$ 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 $\{v_1, v_2, \cdots, v_t\} \setminus C$ (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_l that 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 that will make our writing more concise.

Algorithm 4: OBPE (G_3, X, Y, T_l, v)

```
 1: Suppose $P = \{\tilde{B}_0, \tilde{c}_1, \tilde{B}_1, \cdots, \tilde{c}_i, \tilde{B}_i, \cdots, VB\}$ is the
    longest leaf-root path of $T_l$.
 2: if there is only one cut-vertex $c$ in $P$ then
 3:     if $c$ is the only cut-vertex in $T_l$ then
 4:         $T_l$ 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:         $T_l$ has at most five leaf blocks connected to $v$. Also
            each leaf block $B_i$ is connected to the root of $T_l$, $VB$, via
            one cut-vertex $c_i$. Employ at most five $H_3$-path such that
            none of $\{v, c_i\}$ for $1 \le i \le 5$ can form a separator.
 7:     end if
 8: else
 9:     Elect a new root $R$ of $T_l$ (see the main text for details)
        and modify the leaf-root path $P$ to $\{\tilde{B}_0, \tilde{c}_1, \cdots, R\}$.
        Let $P_i = P \setminus \{\tilde{B}_0, \cdots, \tilde{c}_i\}$. Let $LB(\tilde{c}_1)$ be the set of
        leaf-blocks being adjacent with $\tilde{c}_1$ in $T_l$.
10:     if $\tilde{B}_1$ is $R$ then
11:         Find an $H_3$-path $\Gamma_i$ connecting $LB_i \setminus \{\tilde{c}_1\}$ with
            $Y \setminus (LB_i \cup \{v\})$ for each $LB_i \in LB(\tilde{c}_1)$ $(1 \le i \le q \le 4)$,
            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 separators of $Y$, and thus there must exist an
            $H_3$-path $\Gamma$ starting from $\tilde{B}_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ ending at $Y \setminus \tilde{B}_1$
            with endpoint $x$.
15:         if $x$ does not lie in $P_2$ then
16:             Add $H_3$ paths $\Gamma_i$ 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
                $\{\tilde{c}_2, \cdots, \tilde{c}_{p-1}\}$ into a good-point by adding a constant
                number of $H_3$-paths.
20:         end if
21:     end if
22: end if
```

Definition 4.1: Consider a leaf-block tree T_l 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_l to VB, the root of T_l.

Now, we pick the *longest leaf-root path*

$$P = \{\tilde{B}_0, \tilde{c}_1, \tilde{B}_1, \cdots, \tilde{c}_i, \tilde{B}_i, \cdots, \tilde{c}_k, VB\} \qquad (2)$$

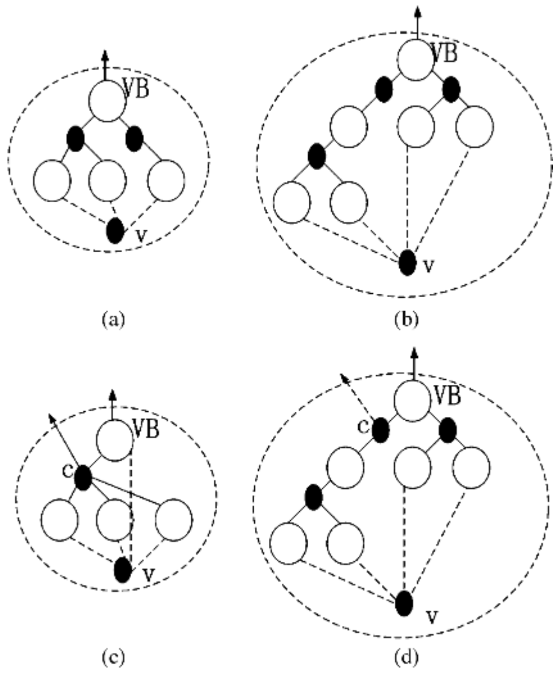


Fig. 4. (a) Case 1, Subcase 1-2: There is more than one cut-vertex in T_l. (b) Case 2 with $R = VB$. (c) Case 1, Subcase 1-1: There is one virtual cut-vertex c. (d) Case 2 with $R = c$.

where $k \ge 1$ is the number of cut-vertices in P. In what follows, we will make either v (if $k = 1$) or one of $\tilde{c}_i$ $(1 \le i \le k-1$, if $k \ge 2)$ 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_l. Then, T_l is a star centered at c, and $Y \setminus \{\tilde{c}, v\}$ 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_l. Then, T_l is a tree such that every leaf-block is connected with VB by a cut-vertex. Let c_i be the cut-vertex for $1 \le i \le q$. Add several H_3-paths to remove the pair of separators $\{v, c_1\}$. Then, check whether v and c_2 are still a separator; if so, add some H_3-paths, $\cdots$, continuing this process until v and c_q are no longer a separator. Since G_3 is a UDG, we have $q \le 5$. By adding at most five H_3-paths, v can be a good-point (see Lemma 4.3).

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 $\tilde{B}_1$ to the root VB. Let R be the first node (a block or a cut-vertex) in P with degree larger than two in T_l. 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 VB as R. Therefore, R can be a block $\tilde{B}_m (1 \le m \le k)$ or a cut-vertex $\tilde{c}_m (2 \le m \le k)$ in P, or can coincide with the root VB or a virtual cut-vertex c lying in VB. After finding the new root R, we modify the leaf-root path P given in (2) into a new leaf-root path with endpoint R, still denoted as P with understood P ends at R. Let $P_i = P \setminus \{\tilde{B}_0, \cdots, \tilde{c}_i\}$.

Let $LB(\tilde{c}_1)$ be the set of leaf-blocks being adjacent with $\tilde{c}_1$ in T_l. Note that the number q of leaf-blocks in $LB(\tilde{c}_1)$ is at most

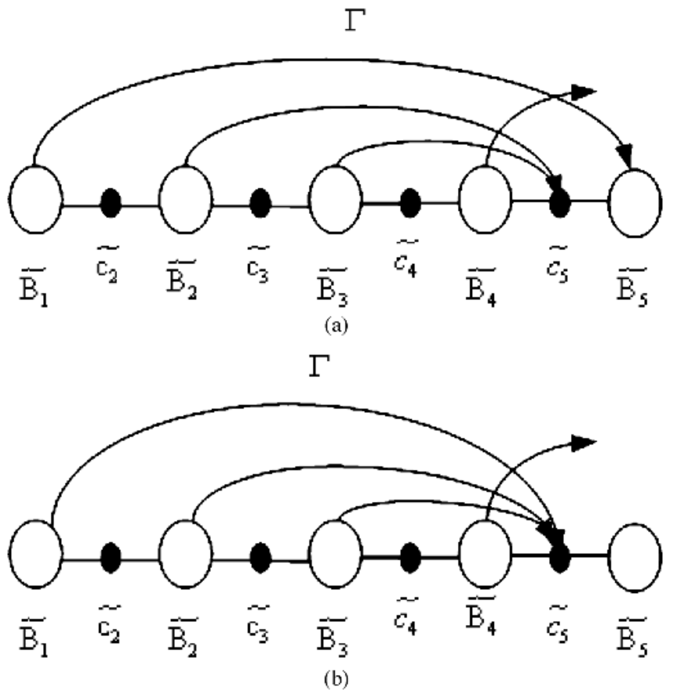


Fig. 5. Procedure-ECP. (a) R ends at a block or the virtual block VB; if $\tilde{c}_2$ and $\tilde{c}_3$ cannot be changed into good, $\tilde{c}_4$ will be eventually changed into good. (b) R ends at a cut-vertex or a virtual cut-vertex c; if $\tilde{c}_2$ and $\tilde{c}_3$ cannot be changed into good, $\tilde{c}_4$ will be eventually changed into good.

four. Since v and $\tilde{c}_1$ constitute a separator of Y, $Y \setminus \{v, \tilde{c}_1\}$ breaks into at most five parts, and there must exist an H_3-path Γ_i connecting $LB_i \setminus \{\tilde{c}_1\}$ with $Y \setminus (LB_i \cup \{v\})$ for each $LB_i \in LB(\tilde{c}_1)(1 \le i \le q)$. Now consider $\tilde{B}_1$. If $R = \tilde{B}_1$ (note R cannot be a virtual block since $k \ge 2$), then adding Γ_i and applying *Procedure-EEN* below makes $\tilde{c}_1$ to be good. Next, suppose $m \ge 2$. Since $\tilde{c}_1$ and $\tilde{c}_2$ constitute a pair of separators of Y, there must exist an H_3-path Γ starting from $\tilde{B}_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ ending at $Y \setminus \tilde{B}_1$ 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 $\Gamma_i(1 \le i \le q)$ and Γ then applying *Procedure-EEN (Eliminating the External Nodes)*; see Lemma 4.6.
- *Subcase 2-2:* If x lies in P_2, say $x = \tilde{c_p}$ or $x \in \tilde{B}_p \setminus \{\tilde{c_p}, \tilde{c}_{p+1}\}$ for some $2 \le p \le k$, 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 4.5.

Procedure-ECP: Suppose that we are given a leaf-root path P as in (2), and there exists an H_3-path Γ starting from $\tilde{B}_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ 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.

1) If Γ ends at $\tilde{B}_p \setminus \{\tilde{c_p}\}$, 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 separators of Y, there exists an H_3-path Γ' connecting $\tilde{B}_2 \setminus \{\tilde{c}_2, \tilde{c}_3\}$ and $Y \setminus \tilde{B}_2$. If the endpoint y of Γ' does not lie on the path $P_3 \setminus P_p = \{\tilde{B}_3, \cdots, \tilde{c_p}\}$, then $\tilde{c_2}$ can be changed into a good-point by adding Γ and Γ' and then applying Procedure-EEN. Otherwise, we have y lies in $P_3 \setminus P_p$. Now set $\tilde{B}_1 \leftarrow \tilde{B}_2$ and $x \leftarrow y$. Repeat the same process until we have changed one of the cut-vertices into good; see Fig. 5 and Lemma 4.5.
2) The similar arguments can be applied if Γ ends at $\tilde{c}_p$ or the virtual block VB (or c); we omit the details.

Finally, after connecting some good blocks $\tilde{B}_i$ (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 $\tilde{B}_i(1 \le i \le r)$ be good blocks in T_l with cut-vertex $\tilde{c}$ connecting them. Suppose that each of $\tilde{B}_i$ 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 $u \in \cup_{i=1}^{r} E(\tilde{B}_i)$, check whether $\{u, \tilde{c}\}$ is a separator of Y, if so, adding an H_3-path to eliminate this pair of separators, until any external node u cannot constitute a separator of Y with $\tilde{c}$. It can be shown that at most five H_3-paths are needed to finish this step; see Lemma 4.4.

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.

B. Analysis of FT-CDS-CA

In this section, we prove that given any G_3 containing a solution, FT-CDS-CA can correctly generate a $C_{3,3}$. In addition, we show that the approximation ratio of our algorithm is a constant and the running time of our algorithm is polynomial.

Lemma 4.1: Let G_3 be a 3-connected graph and H be a 2-connected subgraph of G_3. Let v be a bad-point of H and T be a leaf-block tree that can be obtained after a decomposition of $H \setminus \{v\}$. Suppose $V(T) = \{B_1, \cdots, B_s\} \cup \{c_1, \cdots, c_t\}$. Let $u \in I(B_i)$ be an internal points of B_i. Then, $\{u, w\}$ is not a separator of graph H for $w \in H \setminus B_i$.

Proof: The intuition behind the lemma is clear. Now, we give a formal proof. The assertion that $\{u, v\}$ is not a separator of H follows easily from the fact that $H \setminus \{v\}$ is 1-connected and B_i is 2-connected. Thus, the deletion of u keeps the connectedness of $H \setminus \{v\}$. Next, let $w \neq v$. We show $\{u, w\}$ is not a separator of H, i.e., $H \setminus \{u, w\}$ is connected. Letting $x, y \in V(H) \setminus \{v, u, w\}$ be two distinct vertices, we shall show x and y can connect to each other in $H \setminus \{u, w\}$. We distinguish two cases.

Case 1: $w \in B_j$ $(j \neq i)$ and $w \notin \{c_1, \cdots, c_t\}$. If x and y are contained in the same block B_k, then x and y can be connected to each other in $H \setminus \{v, u, w\}$. This follows immediately when $k \neq i, j$; while if $k = i$ or $k = j$, since B_k is 2-connected, x and y can still be connected to each other after the deletion of u and w. Next, suppose that x and y are not contained in the same block, and let $(\tilde{B}_1, \tilde{c}_1, \tilde{B}_2, \tilde{c}_2, \cdots, \tilde{B}_q)$ $(q \ge 2)$ be a path in the leaf-block tree T connecting $\tilde{B}_1$ and $\tilde{B}_q$ such that $x \in \tilde{B}_1, y \in \tilde{B}_q$. Clearly, there is a path $P = (xP_1\tilde{c}_1P_2\tilde{c}_2 \cdots \tilde{c}_{q-1}P_qy)$ in $H \setminus \{v, u, w\}$ connecting x and y, where P_1 (resp. P_q) is a path contained in $\tilde{B}_1$ (resp. $\tilde{B}_q$) connecting x (resp. y) and c_1 (resp. c_{q-1}), and P_m $(2 \le m \le q-1)$ is a path contained in $\tilde{B}_i$ connecting $\tilde{c}_{m-1}$ and $\tilde{c}_m$. If u and w is not contained in any of $\tilde{B}_m$, then our assertion clearly holds. Now, suppose that u is contained in, say, $\tilde{B}_{m'}(= B_i)$. Since $\tilde{B}_{m'}$ is 2-connected, there is a path $\tilde{Q}_{m'}$ (which is independent of $\tilde{P}_{m'}$) in $\tilde{B}_{m'}$ connecting $\tilde{c}_{m'-1}$ and $\tilde{c}_{m'}$. If w is also contained in some of $\tilde{B}_{m''}$

($m' \neq m''$), then there exists a path a path $\tilde{Q}_{m''}$ (which is independent of $\tilde{P}_{m''}$) in $\tilde{B}_{m''}$ connecting $\tilde{c}_{m''-1}$ and $\tilde{c}_{m''}$. Replace $\tilde{Q}_{m'}$ (resp. $\tilde{Q}_{m''}$) with $\tilde{P}_{m'}$ (resp. $\tilde{Q}_{m''}$) in the original path P. Then, x and y are still connected to each other in $H \setminus \{v\}$ after the deletion of u and w. In the case that one of u and w is contained in some of $\tilde{B}_m$, it is not difficult to see that the same result still holds.

Case 2: $w \in B_j$ $(j \neq i)$, $w \notin B_i$ and $w \in \{c_1, \cdots, c_t\}$. Note H is 2-connected. It follows that there exists another path $P' = (xP'_1c'_1P'_2c'_2 \cdots, c'_pP'_py)$ connecting x and y, which is independent of the previous path P and passes through vertex v. If u is not contained in the path P', clearly x and y are still connected through P' after the deletion of u in $H \setminus \{w\}$. Otherwise, suppose u is contained in P'. Similar to Case 1, we can replace one P'_m with another independent path Q'_m in B_i such that after the deletion of u, the vertices x and y still connect each other in $H \setminus \{w\}$.

Thus, x and y are always connected with each other in $H \setminus \{u, w\}$ for $x, y \in V(H) \setminus \{v, u, w\}$. Therefore, the lemma holds. ■

Lemma 4.2: After the merging process, there is exactly one virtual block left in T'_l 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 (1).

Construct a subtree $\tilde{T}_l$ as follows. Let $B'_1, B'_2, \cdots, B'_m$ 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 connecting to nodes outside B directly, then VB is just block B'_1. Next, suppose $m \geq 2$. Then, $V(\tilde{T}_l)$ consists of those blocks and cut-vertices lying on the path between any two B'_i and B'_j for $i \neq j$.

Examine the leaf-blocks of $\tilde{T}_l$. 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 $\tilde{T}_l$, the remaining blocks can be merged into a unique block VB. ■

Lemma 4.3: After Line 2 of Algorithm 4 is executed, then v becomes a good-point by adding at most five H_3-paths.

Proof: Suppose after level-l decomposition we obtain a leaf-block tree T_l from $B \setminus \{v\}$ with vertex set given as in (1) whose blocks B_i are all good. P is the longest leaf-root path in T_l with a root that may end at the virtual cut-vertex c or the virtual block VB. 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 VB) with all blocks being leaf-blocks except for the center VB.

In the former case, since $\{v, c\}$ is a pair of separators 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 latter case, let $c_i (1 \leq i \leq q)$ be the cut-vertices in T_l that 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 UDG, every nodes can 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 4.1 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 4.4: 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 $\tilde{B}_i$ for $1 \leq i \leq r$.

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

Now we show at most five H_3-paths are needed in this step. Let $\hat{T}_i$ be the leaf-block tree of $B_i \setminus \tilde{c}$, then $\tilde{c}$ can constitute separators only with those cut-vertices in $\hat{T}_i$. However, we note that every cut-vertex $\tilde{c}$ in $\hat{T}_i$, 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 that are adjacent to nodes in $Y \setminus B_i$. Thus, the number of nodes in $E(B_i)$ that make a pair of separator with $\tilde{c}$ is no more than the number of leaf-blocks in $\hat{B}_i$.

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

Lemma 4.5: After Procedure-ECP is executed, at least one cut-vertex $\tilde{c}_i$ 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 Γ 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 $\tilde{B}_1 \setminus \{\tilde{c}_1, \tilde{c}_2\}$ and $\tilde{B}_p$ are connected by an H_3-path Γ, then $\tilde{c}_2$ can constitute a separator of Y with only $\tilde{c}_3, \cdots, \tilde{c}_p$, and some of those external nodes of $\tilde{B}_1$ and $\tilde{B}_2$ (which we have ignored in the multidecomposition process). Now, since $\tilde{c}_2$ and $\tilde{c}_3$ constitute a pair of separators of Y, there exists an H_3-path Γ' connecting $\tilde{B}_2 \setminus \{\tilde{c}_2, \tilde{c}_3\}$ and $Y \setminus \tilde{B}_2$. If the endpoint y of Γ' does not lie on the path $P_3 \setminus P_p = \{\tilde{B}_3, \cdots, \tilde{c}_p\}$, then $\tilde{c}_2$ cannot constitute a separator of Y with any of $\tilde{c}_3, \cdots, \tilde{c}_p$. Then, after applying Procedure-EEN, $\tilde{c}_2$ cannot constitute a separator of Y with those external nodes of $\tilde{B}_1$ and $\tilde{B}_2$. Therefore, $\tilde{c}_2$ can be changed into a good-point by adding two H_3-paths Γ and Γ' 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 $\tilde{c}_3, \cdots, \tilde{c}_{p-1}$ can be changed into a good-point.

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

Lemma 4.6: After Line 13 of Algorithm 4 is executed, at least one $\tilde{c}_i$ becomes a good-point by adding at most 10 H_3-paths.

Proof: Suppose that we are confronted 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 Γ_i, $\tilde{c}_1$ and v cannot constitute a separator of Y. Adding Γ 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.

Moreover, applying Procedure-EEN removes all separators $\{\tilde{c}_1, u\}$ for every external node 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 are confronted with Case 2, Subcase 2-2, Procedure-ECP makes one of $\tilde{c}_2, \cdots, \tilde{c}_{p-1}$ to be good by adding at most $2 + 5 = 7$ H_3-paths. ■

Lemma 4.7: Each time Line 12 of Algorithm 1 is executed, at least one bad-point in X will become a good-point by adding at most 20 nodes, where X is the number of bad-points in $Y := C_{2,3}$.

Proof: By Lemmas 5.3–5.6, at each iteration, at least one bad-point becomes good by adding at most 10 H_3-paths, the length of which is at most three (which includes two extra nodes). Therefore, at most $2 \times 10 = 20$ nodes are needed to change one bad-point into a good-point. ■

Theorem 4.8 ([17]): There exists a polynomia-time approximation algorithm A that can generate an approximated solution for $(2, m)$-CDS with performance ratios $(5 + 25/m)$ for $2 \leq m \leq 5$ and 11 for $m > 5$.

Lemma 4.9: Given a G_3, let H be a connected subgraph containing a $C_{2,3}$ of the G_3. Then, adding an H_3-path P to H does not introduce a new bad-point to 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 constitutes 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. Hence, $(H \cup P) \setminus \{u, w\}$ 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 4.10: 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 eliminate 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 $\{B_1, B_2, \cdots, B_s\}$ and cut vertices $\{c_1, c_2, \cdots, c_s\}$. This can be achieved by multilevel 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 the shortest path between any pair of nodes.

Once B is found, we have to change either one of the bad-points in $\{c_1, c_2, \cdots, c_s\}$ or v into a good-point by adding several H_3-paths. This step takes time $O(n^3)$ since finding an H_3-path between two connected components of a graph takes time $O(n^2)$, and we have to try at most $O(s) = O(n)$ 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 4.11: Algorithm 1 is a 280-approximation for the 3-Connected 3-Dominating Set problem.

Proof: From Theorem 4.8, we have an r-approximation algorithm for computing a $C_{2,3}$, where $r = 40/3$. Then, we can have a $C_{2,3}$ such that $|C_{2,3}| \leq r|C_{2,3}^{\text{opt}}|$. 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 $|Y| = |C_{2,3}| + 20|X| \leq r|C_{2,3}^{\text{opt}}| + 20|C_{2,3}| \leq r|C_{2,3}^{\text{opt}}| + 20r|C_{2,3}^{\text{opt}}| \leq 21r|C_{2,3}^{\text{opt}}| \leq 21r|C_{3,3}^{\text{opt}}|$. ■

C. Generalization for Any $m \neq 3$

When $m > 3$, we first compute a $C_{2,m}$ using the existing algorithm in [17] and [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 4.12: The approximation ratio of this strategy is $21r$ for $m > 3$, where $r = (5 + 25/m)$ for $3 \leq m \leq 5$ and 11 for $m > 5$

Proof: For $m > 3$, it is easy to show that $|Y| \leq 21r|C_{3,m}^{\text{opt}}|$ using the argument in the proof of Theorem 4.11. ■

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] and [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 4.13: 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 $C_{3,1}^{\text{opt}}$. In fact, by the algorithms in [10] and [11], we have $C_{1,3} = I_1 \cup I_2 \cup I_3 \cup C$, where I_i is the maximal independent set obtained sequentially and C is some additional nodes added to make I_1 to be connected. Since $|I_i| \leq 5|C_{1,1}^{\text{opt}}|$ $(i = 1, 2, 3)$ and $|C| \leq 2|I_1|$, we have $|C_{1,3}| \leq 17|C_{1,1}^{\text{opt}}|$. Moreover, by the proofs in [10] and [11], we have $|C_{2,3}| \leq 2|C_{1,3}|$. By Algorithm 1, $|C_{3,3}| \leq 21|C_{2,3}| \leq 2 \cdot 21|C_{1,3}| \leq 17 \cdot 2 \cdot 21|C_{1,1}^{\text{opt}}|$. Note that $|C_{1,1}^{\text{opt}}| \leq |C_{3,1}^{\text{opt}}|$. It follows that $|C_{3,3}| \leq 17 \cdot 2 \cdot 21|C_{3,1}^{\text{opt}}|$.

For $m = 2$, the same algorithm as above can be applied. The approximation ratio can be obtained similarly by noting that $|C_{1,1}^{\text{opt}}| \leq |C_{3,2}^{\text{opt}}|$. ■

By combining Theorems 4.11–4.13, we have the following conclusion.

Theorem 4.14: There exists an $O(1)$-approximation algorithm for computing $(3, m)$-CDS in UDGs for any m.

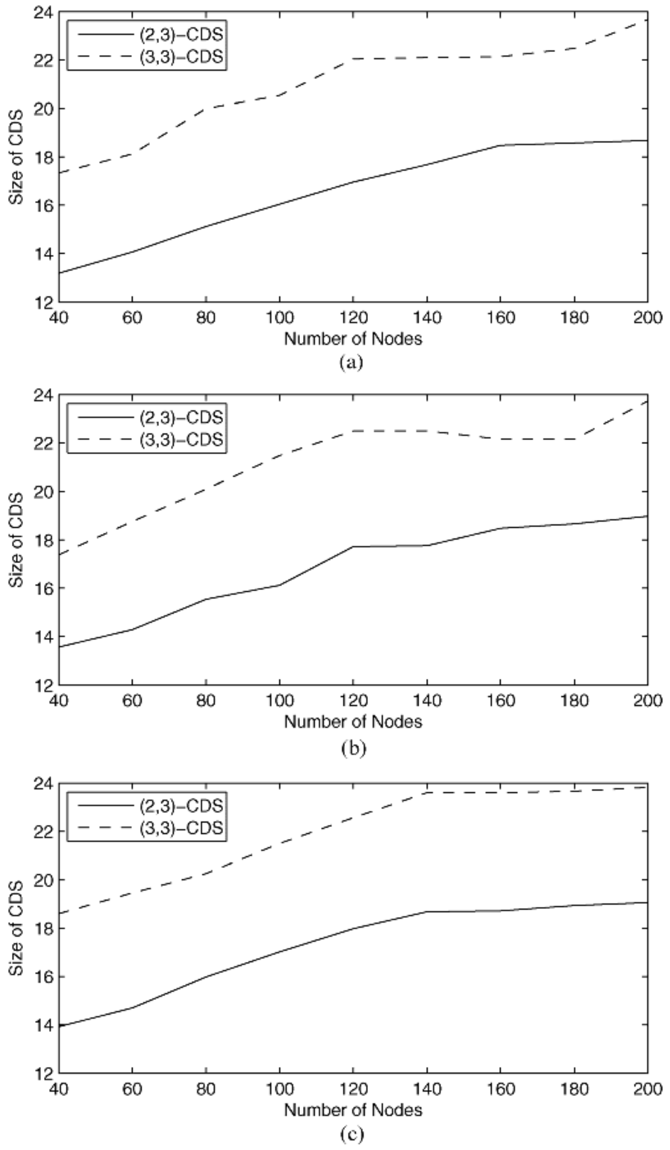


Fig. 6. Comparison of the size of CDS by the (2, 3)-CDS computation algorithm by Shang *et al.* [17] and our (3, 3)-CDS computation algorithm, FT-CDS-CA. (a) 25 × 25 virtual space. (b) 50 × 50 virtual space. (c) 75 × 75 virtual space.

V. 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 25×25, 50×50, and 75×75. 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.

Fig. 6(a)–(c) is the result of our simulation performed over a 25×25, 50×50, and 75×75 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.

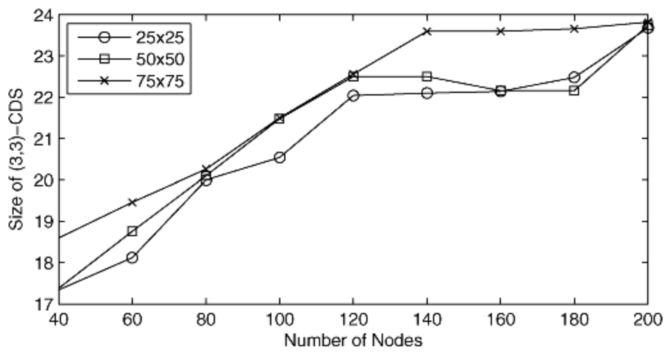


Fig. 7. Size of (3, 3)-CDS generated by FT-CDS-CA under various virtual space size.

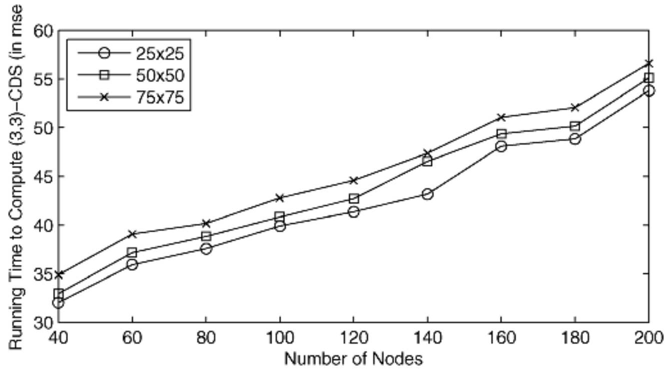


Fig. 8. Running time of FT-CDS-CA to compute (3, 3)-CDS under various virtual space size.

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 under various virtual space size and number of nodes. The figure is based on the average of 100 trials per each parameter setting. For this simulation, we ran the simulation on a personal computer with AMD Phenom IIX4 840 Processor 3.20 GHz and 8 GB memory. The simulation code is written Java (ver. 1.7.0_03) and executed Windows 7 64 bits Operating System with Java SE Runtime Environment Build: 1.7.0_03-b05 and Java HotSpot(TM) 64-Bit Server VM Build:

22.1-b02, mixed mode. In general, as the number of nodes increases, 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 10×10 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 to the (2, 3)-CDS by Shang *et al.*'s algorithm and the (3, 3)-CDS by FT-CDS-CA, respectively. After 30 repetitions, we compute the average. 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.* 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.

VI. Conclusion

This paper investigated 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 algorithms for this problem is flawed for $k \geq 3$. Therefore, we proposed 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 a disk graph. In addition, we plan to continue studying the problem of computing quality fault-tolerant virtual backbone in more realistic network abstractions as our future work.

Acknowledgment

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

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[2] C. R. Dow, P. J. Lin, S. C. Chen, J. H. Lin, and S. F. Hwang, "A study of recent research trends and experimental guidelines in mobile ad hoc networks," in *Proc. 19th AINA*, Taiwan, Mar. 2005, pp. 72–77.

[3] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proc. 5th Annu. ACM/IEEE MobiCom*, Washington, DC, Aug. 1999, pp. 152–162.

[4] A. Ephremides, J. Wieselthier, and D. Baker, "A design concept for reliable mobile radio networks with frequency hopping signaling," *Proc. IEEE*, vol. 75, no. 1, pp. 56–73, Jan. 1987.

[5] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing ad hoc routing with dynamic virtual infrastructures," in *Proc. 20th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2001, vol. 3, pp. 1763–1772.

[6] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20, pp. 374–387, Apr. 1998.

[7] M. R. Garey and D. S. Johnson*, Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1978.

[8] J. Wu and H. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," in *Proc. 3rd DIAL-M*, 1999, pp. 7–14.

[9] P.-J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *Proc. 21st Annu. IEEE INFOCOM*, New York, NY, Jun. 2002, vol. 3, pp. 1597–1604.

[10] X. Cheng, X. Huang, D. Li, W. Wu, and D.-Z. Du, "A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks," *Networks*, vol. 42, no. 4, pp. 202–208, 2003.

[11] Y. Li, S. Zhu, M. T. Thai, and D.-Z. Du, "Localized construction of connected dominating set in wireless networks," in *Proc. NSF TAWN*, Chicago, IL, Jun. 2004, pp. 1–9.

[12] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, and K.-I. Ko, "A greedy approximation for minimum connected dominating sets," *Theoret. Comput. Sci.*, vol. 329, no. 1–3, pp. 325–330, 2004.

[13] P.-J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," *J. Mobile Netw. Appl.*, vol. 9, no. 2, pp. 141–149, 2004.

[14] F. Dai and J. Wu, "On constructing k-connected k-dominating set in wireless network," in *Proc. 19th IEEE IPDPS*, 2005, p. 81a.

[15] W. Wu, H. Du, X. Jia, Y. Li, and S. C.-H. Huang, "Minimum connected dominating sets and maximal independent sets in unit disk graphs," *Theoret. Comput. Sci.*, vol. 352, pp. 1–7, 2006.

[16] S. Funke, A. Kesselman, U. Meyer, and M. Segal, "A simple improved distributed algorithm for minimum CDS in unit disk graphs," *Trans. Sensor Netw.*, vol. 2, no. 3, pp. 444–453, 2006.

[17] W. Shang, F. Yao, P. Wan, and X. Hu, "On minimum m-connected k-dominating set problem in unit disc graphs," *J. Combin. Optim.*, vol. 16, no. 2, pp. 99–106, Dec. 2007.

[18] M. T. Thai, N. Zhang, R. Tiwari, and X. Xu, "On approximation algorithms of k-connected m-dominating sets in disk graphs," *Theoret. Comput. Sci.*, vol. 358, pp. 49–59, 2007.

[19] Y. Wu, F. Wang, M. T. Thai, and Y. Li, "Constructing k-connected m-dominating sets in wireless sensor networks," in *Proc. IEEE MILCOM*, Orlando, FL, Oct. 29–31, 2007, pp. 1–7.

[20] M. T. Thai, F. Wang, D. Liu, S. Zhu, and D.-Z. Du, "Connected dominating sets in wireless networks with different transmission ranges," *IEEE Trans. Mobile Comput.*, vol. 6, no. 7, pp. 721–730, Jul. 2007.

[21] Y. Li, D. Kim, F. Zou, and D.-Z. Du, "Constructing connected dominating sets with bounded diameters in wireless networks," in *Proc. 2nd WASA*, Chicago, IL, Aug. 1–3, 2007, pp. 89–94.

[22] X. Li, X. Gao, and W. Wu, "A better theoretical bound to approximate connected dominating set in unit disk graph," in *Proc. WASA*, 2008, pp. 162–175.

[23] D. Li, H. Du, P.-J. Wan, X. Gao, Z. Zhang, and W. Wu, "Minimum power strongly connected dominating sets in wireless networks," in *Proc. ICWN*, 2008, pp. 447–451.

[24] D. Kim, X. Li, F. Zou, Z. Zhang, and W. Wu, "Recyclable connected dominating set for large scale dynamic wireless networks," in *Proc. 3rd WASA*, Dallas, TX, Oct. 26–28, 2008.

[25] F. Zou, X. Li, D. Kim, and W. Wu, "Construction of minimum connected dominating set in 3-dimensional wireless network," in *Proc. 3rd WASA*, Dallas, TX, Oct. 26–28, 2008, pp. 134–140.

[26] N. Zhang, I. Shin, F. Zou, W. Wu, and M. T. Thai, "Trade-off scheme for fault tolerant connected dominating sets on size and diameter," in *Proc. ACM FOWANC*, 2008, pp. 1–8.

[27] Y. Wu and Y. Li, "Construction algorithms for k-connected m-dominating sets in wireless sensor networks," in *Proc. 9th ACM MobiHoc*, Hong Kong, May 26–30, 2008, pp. 83–90.

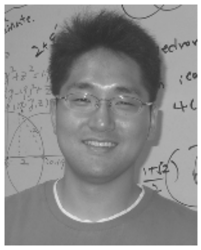
[28] F. Wang, M. T. Thai, and D.-Z. Du, "On the construction of 2-connected virtual backbone in wireless network," *IEEE Trans. Wireless Commun.*, vol. 8, no. 3, pp. 1230–1237, Mar. 2009.

[29] M. Li, P.-J. Wan, and F. Yao, "Tighter approximation bounds for minimum CDS in wireless ad hoc networks," in *Proc. 20th ISAAC*, Dec. 16–18, 2009, pp. 699–709.

[30] D. Kim, Y. Wu, Y. Li, F. Zou, and D.-Z. Du, "Constructing minimum connected dominating sets with bounded diameters in wireless networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 147–157, Feb. 2009.
[31] D. Kim, Z. Zhang, X. Li, W. Wang, W. Wu, and D.-Z. Du, "A better approximation algorithm for computing connected dominating sets in unit ball graphs," *IEEE Trans. Mobile Comput.*, vol. 9, no. 8, pp. 1108–1118, Aug. 2010.
[32] H. Du, W. Wu, S. Shan, D. Kim, and W. Lee, "Constructing weakly connected dominating set for secure clustering in distributed sensor network," *J. Combin. Optim.*, vol. 23, no. 2, pp. 301–307, Feb. 2012.
[33] Y. Li, Y. Wu, C. Ai, and R. Beyah, "On the construction of k-connected m-dominating sets in wireless networks," *J. Combin. Optim.*, vol. 23, no. 1, pp. 118–139, 2012.
[34] D. Kim, X. Gao, F. Zou, and D.-Z. Du, "Construction of fault-tolerant virtual backbones in wireless networks," in *Handbook on Security and Networks*, Y. Xiao, F. H. Li, and H. Chen, Eds. Singapore: World Scientific, Apr. 2011, pp. 488–509.
[35] H. Du, L. Ding, W. Wu, D. Kim, P. M. Pardalos, and J. Willson, , P. M. Pardalos, D.-Z. Du, and R. Graham, Eds., "Connected Dominating Set in Wireless Networks," in *Handbook of Combinatorial Optimization*. New York: Springer, Jul. 2013.
[36] M. L. Huson and A. Sen, "Broadcast Scheduling Algorithms for Radio Networks," in *Proc. IEEE MILCOM*, 1995, vol. 2, pp. 647–651.
[37] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Math.*, vol. 86, pp. 165–177, Dec. 1990.
[38] R. Diestel, "Graph theory," in *Graduate Texts in Mathematics*, 3rd ed. Heidelberg, Germany: Springer-Verlag, 2005, vol. 173.
[39] D. S. Johnson, "Approximation algorithms for combinatorial problems," *J. Comput. Syst. Sci.*, vol. 9, pp. 256–278, 1974.
[40] K. Menger, "Zur allgemeinen Kurventheorie," *Fundam. Math.*, vol. 10, pp. 96–115, 1927.
[41] B. Korte and J. Vygen, *Combinatorial Optimization, Theory and Applications*. New York: Springer, 2000.

Wei Wang received the B.S. degree from ZheJiang University, Hangzhou, China, in 1991, and the M.S. and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China, in 1994 and 2006, respectively, all in mathematics.

He is currently an Assistant Professor with the Department of Mathematics, Xi'an Jiaotong University. His research interests include algebraic graph theory and approximation algorithm design and analysis.

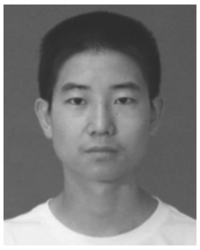


Donghyun Kim (M'10) received the B.S. degree in electronic and computer engineering and M.S. degree in computer science and engineering from Hanyang University, Ansan, Korea, in 2003 and 2005, respectively, and the Ph.D. degree in computer science from the University of Texas at Dallas, Richardson, in 2010.

He is currently an Assistant Professor with the Department of Mathematics and Computer Science, North Carolina Central University, Durham. His research interests include wireless networks, mobile computing, and algorithm design and analysis.

Min Kyung An (S'11) received the M.S. degree in computer science from the University of Texas at Arlington in 2007, and is currently pursuing the Ph.D. degree in computer science at the University of Texas at Dallas, Richardson.

Her major areas of interest are wireless ad hoc and sensor networks, design and analysis of approximation algorithm, and graph theory.



Wei Gao is currently an undergraduate student with the Department of Information and Computational Science, Xi'an Jiaotong University, Xi'an, China.

His research interests include algorithms for routing in wireless ad hoc networks.

Xianyue Li received the B.S. and Ph.D. degrees in mathematics from Lanzhou University, Lanzhou, China, in 2003 and 2009, respectively.

He is currently an Assistant Professor with the School of Mathematics and Statistics, Lanzhou University. His research interests include wireless networks, graph theory, and approximation algorithm design and analysis.

Zhao Zhang received the Ph.D. degree in mathematics from Xinjiang University, Urumqi, China, in 2003.

She is a Full Professor with the College of Mathematics and System Sciences, Xinjiang University. Her current research interests are in combinatorial optimization and graph theory.

Weili Wu (M'09) received the Ph.D. degree in computer science from the University of Minnesota, Twin Cities, in 2002.

She is currently an Associate Professor with the Department of Computer Science, University of Texas at Dallas, Richardson. 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. Her main research interest is in database systems.