Improving Connectivity of Wireless Ad-Hoc Networks

Ning Li and Jennifer C. Hou
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{*nli, jhou*}*@cs.uiuc.edu*

Abstract

A fully connected topology is critical to many fundamental network operations in wireless ad-hoc networks. In this paper, we consider the problem of deploying additional wireless nodes to improve the connectivity of an existing wireless network. Specifically, given a disconnected wireless network, we investigate how to deploy as few additional nodes as possible so that the augmented network can be connected. The problem is termed as the *Connectivity Improvement* (CI) problem. We first prove that CI is NP-complete, and then present a Delaunay Triangulation-based algorithm, *Connectivity Improvement using Delaunay Triangulation* (CIDT). Depending on the priority based on which the components in a disconnected network should be chosen to connect, we devise several different versions of CIDT. We also present two additional optimization techniques to further improve the performance of CIDT. Finally, we verify the effectiveness of CIDT, and compare the performance of its variations via *J-Sim* simulation.

I. Introduction

Network connectivity has always been a research focus in wireless mobile ad hoc networks (MANETs) and wireless sensor networks (WSNs). It is an indispensable function for network services to be in place. Few network services can function if the network is disconnected.

The problem of connectivity maintenance for wireless networks is extremely difficult, because the status of wireless links (which are formed on the fly as nodes move or adjust their transmission power) depends greatly on that of other links in the vicinity, due to wireless interference in the physical layer and medium contention in the MAC layer. Most research efforts have been concentrated on either analyzing the asymptotic connectivity behavior of large-scale networks [1]–[5] or devising topology control protocols to maintain connectivity in the presence of limited mobility (see [6] for a summary). Little attention has been paid to improving or repairing network connectivity during network operations. Given that the connectivity of a wireless network is susceptible to node mobility (typically in MANETs), node failure (typically because of power depletion in WSNs), and unpredictable environment influences (typically in outdoor networks), it is important to *continuously* maintain the connectivity even under all these unfavorable conditions.

In this paper, we consider the problem of deploying additional wireless nodes to improve connectivity of an existing wireless network. Specifically, given a disconnected wireless network, we investigate how to deploy as few additional nodes as possible, so that the augmented network can be connected. This problem can be cast to several applications in practice. For example, a WSN can be partitioned if some sensor nodes die because of energy depletion or hardware failure. It is important to deploy as few new sensor nodes (or mobile nodes such as robots) as possible to reconnect the network. Another example takes place in future combat systems [7]. In a future combat system, a hierarchical communication structure is laid with three layers: i) ground units, including troops, vehicles and sensors that are geographically distributed over a battlefield and form one or more ground ad-hoc networks; ii) low-altitude unmanned aerial vehicles (UAVs) for surveillance and maintenance of connectivity of the ground ad hoc networks; and iii) satellites that connect UAVs and some of the ground vehicles with the joint force command center. As UAVs are equipped with long-range radios and satellite antennas, they can communicate with both the ground vehicles and satellites, and serve as the "relay" nodes in the case that the ad-hoc networks on the ground are partitioned. Since UAVs are scarce resources, it is essential to dispatch as few of them to desirable locations so as to maintain network connectivity in the battlefield. Moreover, their locations should be continuously adjusted, subject to the connectivity status change of ground entities (due to mobility of these ground entities and signal attenuation and path loss caused by terrains). This implies the algorithm should be computationally inexpensive and be invoked on a regular basis to update the fly path of these UAVs.

We first prove the NP-completeness of the *Connectivity Improvement* problem, and then propose a simple, light-weight algorithm, called *Connectivity Improvement using Delaunay Triangulation* (CIDT). As the name suggests, the algorithm constructs a Delaunay Triangulation in the disconnected network, and selects, based on different criteria, triangles to place new nodes. We study several versions of CIDT (each version for a different selection criterion) and prove their correctness. We also present two additional optimization techniques to further improve the performance. Finally we evaluate CIDT, comparing its performance against a simple, baseline heuristic, *Connectivity Improvement using Minimum Spanning Tree* (CIMST) via simulation. Simulation results show that one of the CIDT variations, CIDT$_S$, render the best performance under various scenarios.

The rest of the paper is organized as follows. After summarizing the related work in Section II, we formulate the connectivity improvement problem and prove its NP-completeness in Section III. We then propose CIDT and optimization techniques in Section IV. Following that, we evaluate the performance of CIDT in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

In the section, we briefly review previous work that pertains to the connectivity issue in wireless networks. Most of the works model the wireless networks as *Unit Disk Graphs* (UDGs) [8].

Gupta and Kumar showed in [1] that the critical common range r_n for connectivity of n independently and uniformly distributed wireless nodes in a disk of unit area satisfies that, if $\pi r_n^2 = \frac{\ln n + c(n)}{n}$, then the resulting network is asymptotically connected with probability 1 if and only if $c(n) \to \infty$. In an independent effort, Penrose showed in [2] that M_n, the length of the longest edge in the minimum spanning tree of n points randomly and uniformly distributed in a unit area square satisfies that $\lim_{n\to\infty} \Pr(n\pi M_n^2 - \ln n \leq \alpha) = e^{-e^{-\alpha}}$. This result is strong than that in [1] in the sense that it gives the exact expression of the probability of the connectivity.

For k-connectivity of a geometric random graph with a transmission range r, Penrose [4] proved that the minimum value of r with which the graph is k-connected is equal to the minimum value of r with which the graph has the minimum degree of k, with probability 1 as n goes to infinity. Li *et al.* [5] extended Penrose's work and gave the lower and upper bounds of the minimum value of r at which the graph is k-connected.

Xue and Kumar [3] studied the relationship between connectivity and node degree from another angle. They assumed the same number of nearest neighbors are maintained for each node, and showed that (i) the network is asymptotically disconnected with probability 1 as n increases, if each node is connected to less than $0.074 \log n$ nearest neighbors; and (ii) the network is asymptotically connected with probability 1 as n increases, if each node is connected to more than $5.1774 \log n$ nearest neighbors. Wan and Yi [9] further studied the critical number of neighbors for k-connectivity and found the upper bound to be $\alpha e \log n$, where $\alpha > 1$ is a real number and $e \simeq 2.718$ is the natural base.

Khuller [10] studied the *Connectivity Augmentation* problem and determined a set of edges of minimum weight to be inserted so that the resulting graph is λ-vertex(edge)-connected. The problem is NP-hard for $\lambda > 1$. Khuller does not, however, consider the possibility of adding new vertices into the graph.

Ausiello *et al.* [11] considered the *Minimum Geometric Disk Cover* (MGDC) problem. Given a set of points P in the Euclidean plane and a rational number $r > 0$, they intend to find the set of centers C

with the minimum cardinality, such that every point in P is covered by a disk of radius r that is centered at one of the points in C. This problem can be considered as a special case of a more general problem, the *Facility Location* problem [12]. It is proved to be NP-complete [13]–[16], and a polynomial-time approximation algorithm is presented in [17].

Yannakakis [18] studied the general node (edge) deletion problem in which the minimum number of nodes (edges) is sought whose deletion results in a subgraph satisfying property π, where π belongs to a broad class of nontrivial properties that are hereditary on induced subgraphs. The problem is shown to be NP-complete. Let π be the property of "disconnectivity", then the problem is related to the problem we study in this paper. As Yannakakis approaches the problem from a very different angle, it is not clear whether the results in [18] can also be applied to UDG.

III. Problem Statement

In this section, we define the problem of *Connectivity Improvement* (CI) in wireless networks and prove its NP-completeness. Let the initial network topology be represented by an undirected simple graph $G = (V, E)$ in the plane, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes (vertices) with the common transmission range r_1, and $E = \{(u, v) : d(u, v) \leq r_1, u, v \in V\}$ is the set of links (edges), where $d(u, v)$ is the Euclidean distance between u and v. We assume there exists a monitoring system that provides the coordinates of all nodes to the algorithm. In the example of FCSs, the UAVs may collect and provide the position information of ground vehicles to the command center (or one of the UAVs that act as the coordinator).

We would like to find a set of nodes $U = \{u_1, u_2, \ldots, u_k\}$ with the minimum cardinality, such that the augmented graph $G^* = (V^*, E^*)$ is connected, where $V^* = V \cup U$ and $E^* = \{(u, v) : d(u, v) \leq \min\{r_1, r_2\}, u, v \in V^*\}$, where r_2 is the transmission range of nodes in U. For clarity of presentation, we assume that $r_1 = r_2 = r$. However, the proposed algorithm CIDT can be easily adapted (with little modification) to the case where $r_1 \neq r_2$. Nodes in V and U are termed as *clients* and *connectors*, respectively.

Another closely related problem relates to connectivity improvement in the case that the number of connectors is limited. Given the initial network topology $G = (V, E)$ and an integer $m > 0$ indicating the number of available connectors, we intend to find a set of nodes $U = \{u_1, u_2, \ldots, u_m\}$ such that the graph $G^* = (V^*, E^*)$ is connected as much as possible. Here the *connectedness* of a graph G is defined as the percentage of nodes that are in the largest component of G. This problem is termed as *Connectivity Improvement with Limited Connectors (CILC).*

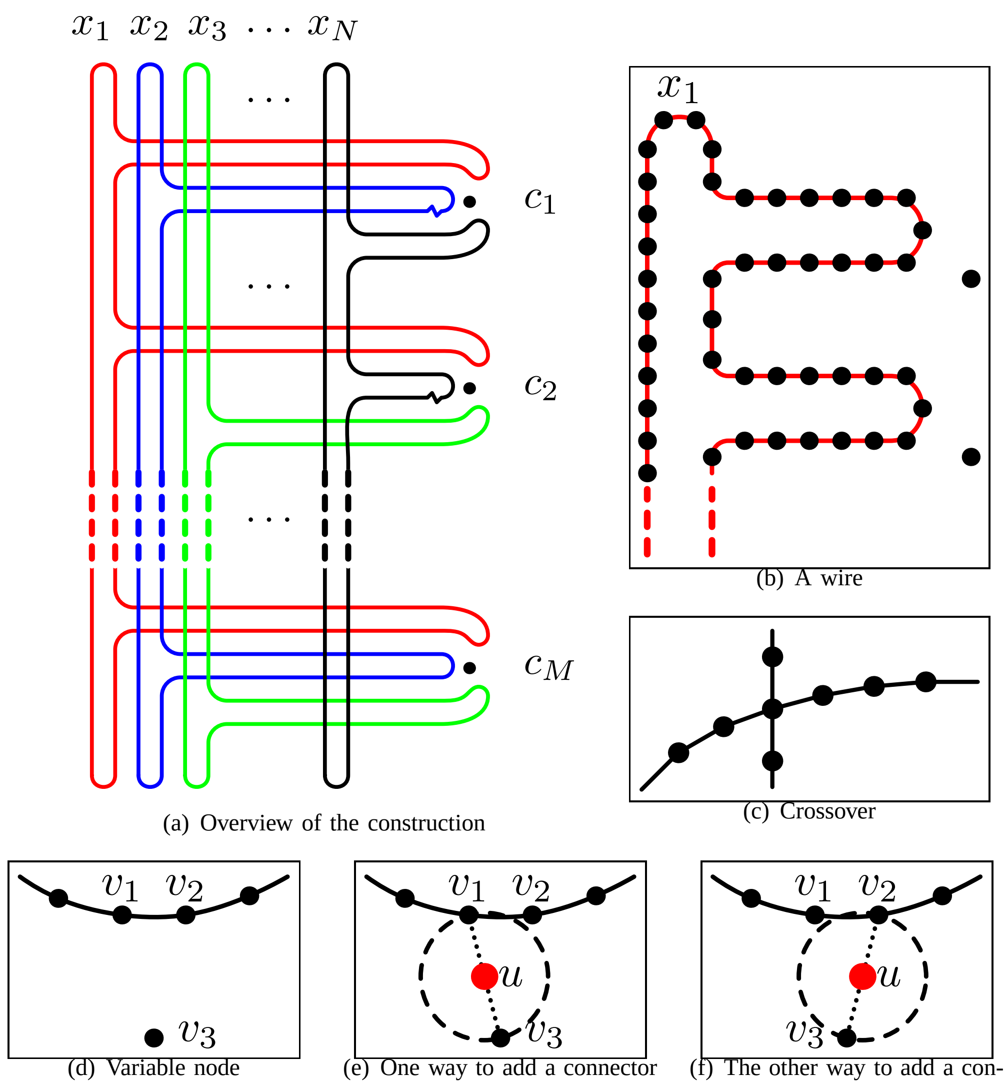
One point is worthy of mentioning. The CI problem is quite different from the MGDC problem. First, if the inserted disks in MGDS are viewed as wireless nodes with certain transmission ranges, then the augmented network in MGDS may not necessarily be connected; while the augmented network in CI is connected. Second, in the MGDS problem, *every* node has to be covered by a disk, while in the CI problem, only a small number of nodes in a component have to be covered by a connector.

Now we prove the NP-completeness of a restricted version of CI where $r_1 = r_2 = 1$. We use strings of constant lengths to encode the coordinates of all client nodes and connector nodes, so that a compact encoding with reasonable precision can be achieved.

Definition 1: **CONNECTIVITY-IMPROVEMENT (CI)**
INSTANCE: A graph $G = (V, E)$ and an integer $m > 0$.
QUESTION: Is there a *connector* set U of size m or less such that the augmented graph $G^* = (V \cup U, E^*)$ is connected?

Theorem 1: The CI problem is NP-complete.

Proof: It is easy to see CI$\in$NP since a nondeterministic algorithm needs only to guess the positions of m connectors and check in polynomial-time whether the augmented graph G^* is connected. To prove that CI is NP-hard, we transform the 3-SAT problem [19] to the CI problem in polynomial-time, using a technique similar to that in [16]. The 3-SAT problem is defined as follows:



(a) Overview of the construction

(b) A wire

(c) Crossover

(d) Variable node

(e) One way to add a connector

(f) The other way to add a connector

Fig. 1. The transformation scheme

Definition 2: **3-SATISFIABILITY (3SAT) [19]**
INSTANCE: A set $C = \{c_1, c_2, \ldots, c_M\}$ of clauses on a finite set X of variables, where $|c_i| = 3$ for $1 \leq i \leq M$.
QUESTION: Is there a truth assignment for X that satisfies all the clauses in C?

Let $C = \{c_1, c_2, \ldots, c_M\}$ (where $|c_i| = 3$ for $1 \leq i \leq M$) be a set of clauses on a finite set $X = \{x_1, x_2, \ldots, x_N\}$ of variables making up an arbitrary instance of 3-SAT. We construct in polynomial time an instance of CI that can be connected by k connectors if and only if C is satisfiable.

The transformation scheme is illustrated in Figure 1. In Figure 1(a), an example is shown for an instance of 3-SAT $C = \{\{x_1, \overline{x_2}, x_N\}, \{x_1, x_3, \overline{x_N}\}, \ldots, \{x_1, x_2, x_3\}\}$, where the wriggles near the clause nodes for c_1 and c_2 represent negated literals. Each of the curved lines in Figure 1(a) is a *wire* (Figure 1(b)) — a set of nodes so arranged that (1) the distance between any two consecutive nodes is always less than 1; and (2) a node can only communicate with its two immediate neighbor nodes. At the crossover, the two wires share one node, as shown in Figure 1(c).

As shown in Figures 1(d), each variable $x_i \in X$ corresponds to a set of *variable nodes*. This set of variable nodes includes a closed loop of wire and an extra node for each clause it participates. These nodes are so arranged that $d(v_1, v_3) = d(v_2, v_3) = 2$ (or extremely close to 2 depending on the encoding scheme we used). To connect v_3 to the wire, at least one connector has to be be added. There are two

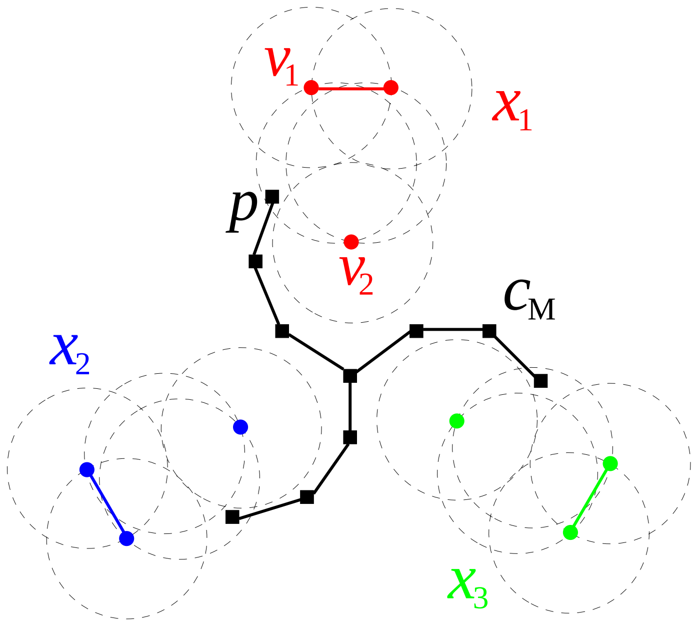


Fig. 2. Clause nodes

possible ways to connect v_3 to the wire with the use of only one connector: Figure 1(e) illustrates the position of the connector u when the literal corresponding to u is true, and Figure 1(f) illustrates the position of u when the corresponding literal is false.

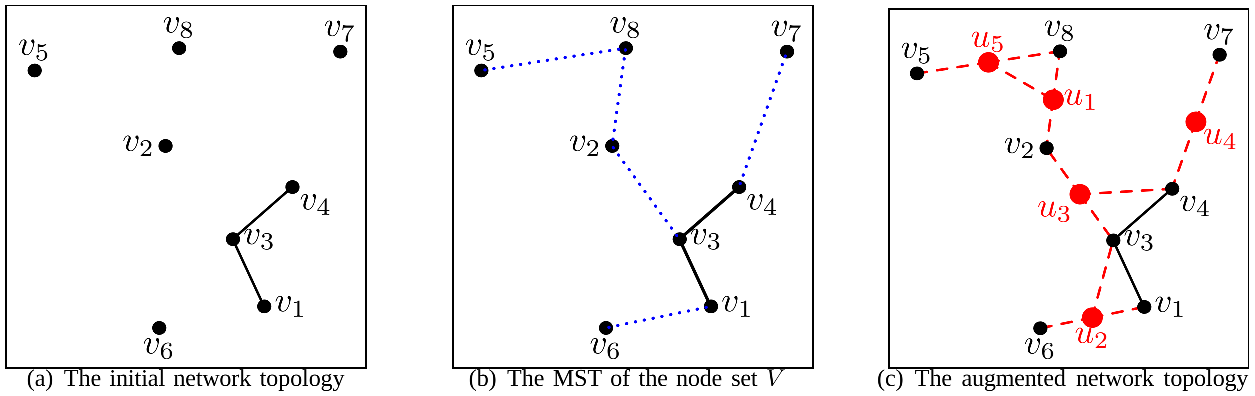
Associated with each of the M clauses in C is a set of 10 clause nodes located in a region in which all 3 variable loops corresponding to the literals in the clause come into close proximity. By properly arranging the variable nodes, it can be ensured that we only need to consider 3 nodes from each variable. An example of such an arrangement for clause c_M (Figure 1(a)) is shown in Figure 2: the round nodes are variable nodes and the square nodes are clause nodes corresponding to the clause c_M. The clause nodes corresponding to the same clause are so arranged that (1) they belong to the same component, i.e., they are connected to each other; (2) none of them can communicate with any variable node.

The variable nodes (round nodes) are arranged so that the clause nodes (square nodes) can be connected to one of the loops for "free" if at least one of the three literals is true. For example in Figure 2, if x_1 is true, a connector should be added in the middle of the line segment $\overline{v_1 v_2}$, which also connects p and v_1. If none of the 3 literals is true, an extra connector is needed to connect the clause nodes to a loop. Therefore, the graph consisting all the variable nodes and clause nodes can be connected by adding $3M$ connectors if and only if C is satisfiable.

To see that this transformation can be performed in polynomial time, it suffices to observing that the number of nodes in the graph is bounded by $O(M \times N)$. Hence the size of the CI instance is bounded by a polynomial function of the size of the 3-SAT instance. In addition, every operations involved is straightforward and can be finished in polynomial time. ∎

IV. Proposed Algorithms

Given that CI is NP-complete, we have to resort to heuristic algorithms to solve the problem unless P=NP. In this section, we first describe a simple, baseline heuristic, called *Connectivity Improvement using Minimum Spanning Tree* (CIMST). Then we elaborate on several versions of the proposed algorithm, CIDT, and prove their correctness. Finally, we present two optimization techniques to further improve the performance of CIDT.



(a) The initial network topology (b) The MST of the node set V (c) The augmented network topology (after adding connectors)

Fig. 3. An example of CIMST

A. Connectivity Improvement using MST

To solve the CI problem, a straightforward solution is to first group the client nodes into connected components, and then add connectors to merge the components. This can be implemented using the Minimum Spanning Tree (MST). We first build the MST, T, of the node set V. Then for each edge $e \in T$ that is not in E (or equivalently, $|e| > r$), we add some connectors along e to connect the two end-nodes. In particular, the number of the connectors added to connect edge e is $\lceil \frac{|e|}{r} \rceil$-1. It is obvious that the resulting augmented network is connected. An example is given in Figure 3, where the solid lines are used to indicate links in E, and dashed lines links that are incident at connectors. The description of the CIMST algorithm is given in Figure 4.

The time complexity of building the MST varies from $O(e \log n)$ (the original Prim's algorithm [20]) to almost linear of e (the optimal algorithm [21]), where n is the number of vertices and e is the number of edges. The time complexity of the rest of the algorithm is $O(e)$. Therefore, the time complexity of CIMST is the same as that of MST.

CIMST will be used as baseline algorithm for comparison. Its shortcoming is, however, that each connector can only be used to connect at most two components. As shown in Figure 5, CIMST would have placed 2 connectors to connect all three components, while in fact one (placed at the position of u_1) is sufficient.

B. Connectivity Improvement using DT

To devise a better algorithm, we re-inspect Figure 5. Let D be the disk bounded by the circumcircle of $\triangle v_2 v_4 v_5$, and u_1 D's center. A disk can *cover* a component if it can cover at least one node in the component. If D is the minimum disk that can cover all three components, then there can be no nodes inside D other than v_2, v_4 and v_5. Therefore, $\triangle v_2 v_4 v_5$ is a triangle of the Delaunay Triangulation (DT) of the node set V. This observation leads to a DT-based connectivity improvement algorithm, CIDT.

The description of CIDT is given in Figure 6. Conceptually, triangles in the Delaunay Triangulation are selected, one by one, with respect to certain criterion, and a connector is inserted into the selected triangle. The process repeats until the augmented network is connected. In what follows, we elaborate on several of the key operations (i.e., which triangle is selected, and where to place the connector inside the triangle) in CIDT.

1) Choosing Candidate Triangles (line 4): Not every triangle in the Delaunay Triangulation is a good candidate for placing a connector. For example, if all three nodes of the triangle are already in the same component, there is no need to place a connector inside the triangle. To select appropriate candidates, we first identify all the connected components in the network by using breadth-first or depth-first search, which can be done in $O(n + e)$ time [22]. Let $Comp(u)$ be the component that node u belongs to, and

```
Procedure: CIMST($G, r$)
Input: $G(V, E)$, a simple graph;
Output: $U$, a set of connector;
begin
 1:    $U := \emptyset$;
 2:    Let $T := (V_T, E_T)$ be the MST of $G$;
 3:    for each edge $(u, v) \in E_T$ do
 4:       if ($d(u, v) > r$) then
 5:          $L := \lceil \frac{d(u,v)}{r} \rceil$;
 6:          for $i = 1$ to $L - 1$
 7:             $p^x := (1 - i/L) \cdot u^x + i/L \cdot v^x$;
 8:             $p^y := (1 - i/L) \cdot u^y + i/L \cdot v^y$;
 9:             $U := U \cup \{p\}$;
10:          end
11:       end
12:    end
end
```

Fig. 4. CIMST algorithm

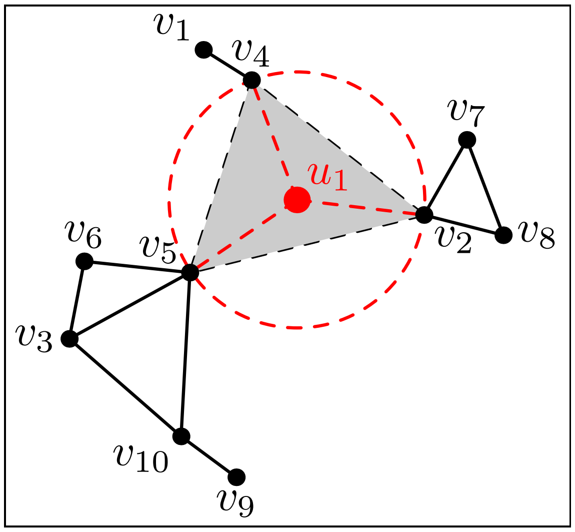


Fig. 5. CIMST is not optimal

$Comp(t)$ be the set of different components that nodes in t belong to, i.e., $Comp(t) = \{Comp(u) : u \in t\}$, where t is a triangle. We select a triangle t as a candidate only if $|Comp(t)| > 1$, i.e., all three nodes of t are not in the same component. An example is given in Figure 7.

2) Find the Best Candidate (line 5): Since only one connector is added in each step, it should be placed inside the "best" candidate triangle. There exist many different criteria that can be used to select the best candidate. We present several different criteria (each of which leads to a specific version of CIDT). we will compare the performance of the various versions of CIDT in Section V.

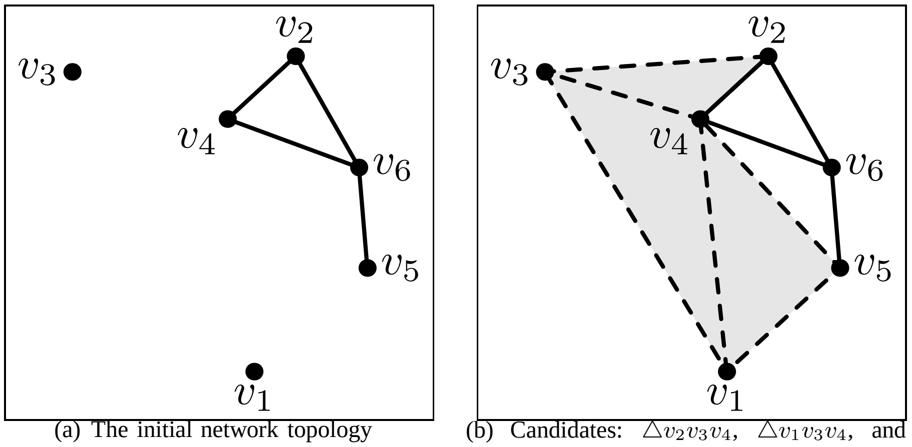
Without loss of generality, given a candidate triangle, we assume that $|e_1| \geq |e_2| \geq |e_3|$, where $e_1 = (v_2, v_3)$, $e_2 = (v_1, v_3)$ and $e_3 = (v_1, v_2)$. Let $r_o(t)$ be defined as the radius of the minimum disk that can *cover* t. Consider the following two cases:

- t is acute: $r_o(t)$ is the radius of the circumcircle of t (Figure 9(a));
- t is not acute: $r_o(t)$ is half of the length of the longest edge in t, i.e., $r_o(t) = |e_1|/2$ (Figure 9(b)).

Let $D(u, r)$ be the disk of radius r centered at node u. A disk $D(u, r)$ is said to *connect* a triangle t

```
Procedure: CIDT($G, r$)
Input: $G(V, E)$, a simple graph;
Output: $U$, a set of connectors;
begin
1:  $U := \emptyset$;
2:  while $G^*$ is not connected do
3:      Let $D_T := (V_D, E_D)$ be the DT of the network
        topology constructed by nodes in $V$ and $U$;
4:      Select the candidate set $CD \subseteq D_T$;
5:      Select the best triangle $t_0 \in CD$;
6:      $p := Placement(t_0)$;
7:      $U := U \cup \{p\}$;
8:  end
end
```

Fig. 6. CIDT algorithm



(a) The initial network topology (b) Candidates: $\triangle v_2v_3v_4$, $\triangle v_1v_3v_4$, and $\triangle v_1v_4v_5$

Fig. 7. Candidates triangles from DT

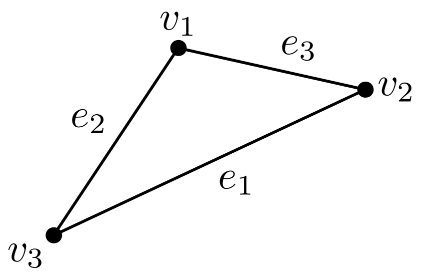
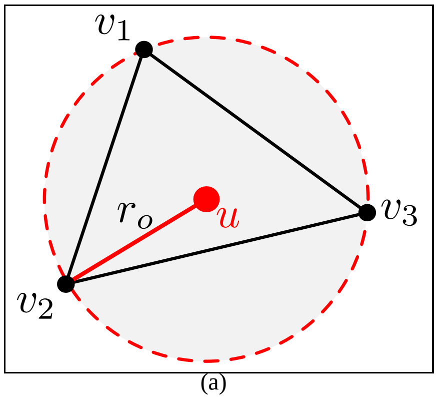


Fig. 8. Illustration of a triangle.

if all three nodes of t are in the same component after adding a connector at u. Let $r_c(t)$ be defined as the radius of the minimum disk that can *connect* t. Since t is a candidate triangle, all three nodes of t cannot be in the same component. Consider the following two cases:

- $|Comp(t)| = 2$: Suppose $Comp(v_1) = Comp(v_2)$ and $Comp(v_1) \neq Comp(v_3)$, then $r_c(t) = \min\{|e_1|, |e_2|\}/2 = |e_2|/2$.
- $|Comp(t)| = 3$: $r_c(t) = r_o(t)$.

Let $S_C(t)$ be the set of components in $Comp(t)$ that can be possibly merged into one by adding one



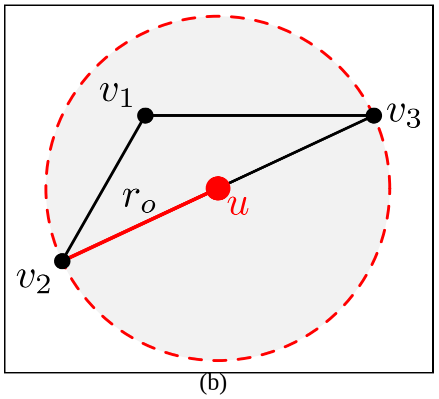


Fig. 9. The definition of $r_o(t)$

connector into t, and $S_S(t)$ the set of nodes in the components that can be possibly merged into one by adding one connector into t. Specifically, $S_C(t) = 0$ and $S_R(t) = 0$ if $|Comp(t)| = 1$. We consider the following criteria for selecting candidate triangles t_0 (summarized in Table I):

- $CIDT_R$: $t_0 = \arg\min_{t \in D_T}\{r_c(t)\}$, i.e., the triangle with the minimum connecting disk is selected.
- $CIDT_S$: $t_0 = \arg\max_{t \in D_T}\{|S_S(t)|\}$, i.e., the triangle is selected such that adding a connector into it can connect the largest number of nodes. If two or more candidates have the same $|S_S(t)|$, the one with the minimum connecting disk is selected.
- $CIDT_{SR}$: $t_0 = \arg\max_{t \in D_T}\{|S_S(t)|/r_c(t)\}$, i.e., the triangle with the largest ratio of the number of connected nodes over the radius of the minimum connecting disk is selected.
- $CIDT_C$: $t_0 = \arg\max_{t \in D_T}\{|S_C(t)|\}$, i.e., the triangle is selected such that adding a connector into it can connect the largest number of components. If two or more candidates have the same $|S_C(t)|$, the one with the minimum connecting disk is selected.
- $CIDT_{CR}$: $t_0 = \arg\max_{t \in D_T}\{|S_R(t)|/r_c(t)\}$, i.e., the triangle with the largest ratio of the number of connected components over the radius of the minimum connecting disk is selected.

TABLE I

SUMMARY OF CRITERIA FOR SELECTING CANDIDATE TRIANGLES

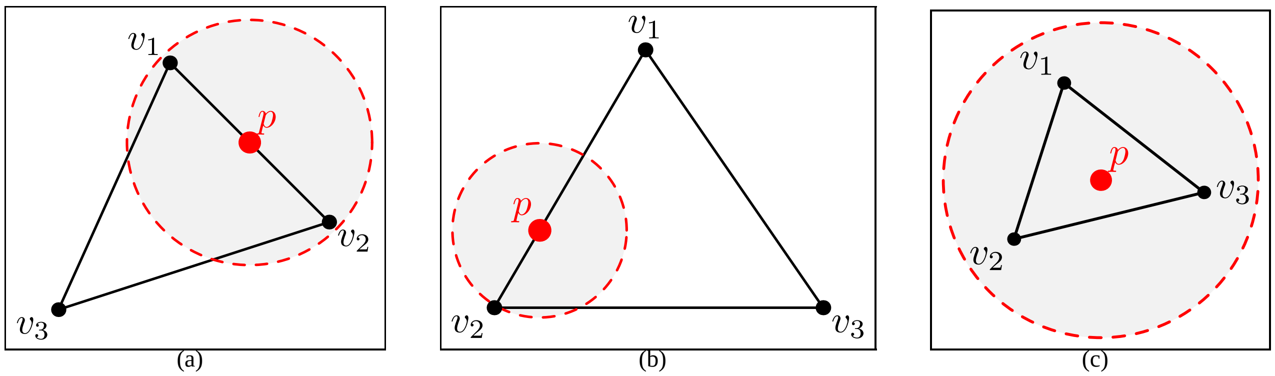
Algorithm	Criterion	Order
$CIDT_R$	$r_c(t)$	min
$CIDT_S$	$\|S_S(t)\|$, then -$r_c(t)$	max
$CIDT_{SR}$	$\|S_S(t)\|/r_c(t)$	max
$CIDT_C$	$\|S_C(t)\|$, then -$r_c(t)$	max
$CIDT_{CR}$	$\|S_C(t)\|/r_c(t)$	max

3) Connector Placement (line 6): After the best candidate t_0 is selected, a connector p will be placed inside the triangle. The objective is to have p cover as many nodes in t_0 as possible. Let $E_d(t)$ be the set of edges in t whose two end-nodes are in different components, i.e., $E_d(t) = \{(u,v) : Comp(u) \neq Comp(v), u, v \in t\}$. Given that t_0 is a candidate, we have $2 \leq |E_d(t_0)| \leq 3$. We consider the following two cases (a complete description of procedure $Placement(t)$ can be found in Figure 18 in Appendix I):

(a) $|E_d(t_0)| = 2$: Find the longest edge $e \in E_d(t_0)$ such that $r > |e|/2$. Put p in the middle of e (as shown in Figure 10(a)). If there is no such edge, put p on the shortest edge $e_{min} = (u,v)$ in $E_d(t)$ such that either $dist(p,u) = r$ or $dist(p,v) = r$ (as shown in Figure 10(b)).

(b) $|E_d(t_0)| = 3$: If a disk of radius r can cover t_0, put p at the center of t_0's circumcircle (as shown in Figure 10(c)); otherwise, attempt to locate p to cover some edge as in (a).

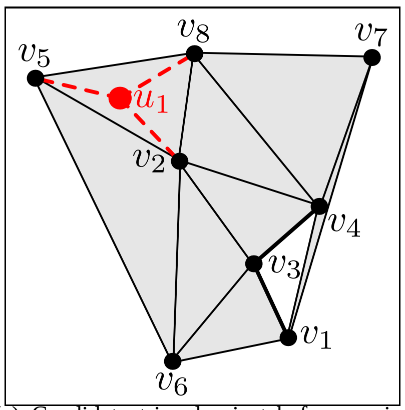
4) Correctness Proof and Complexity Analysis:



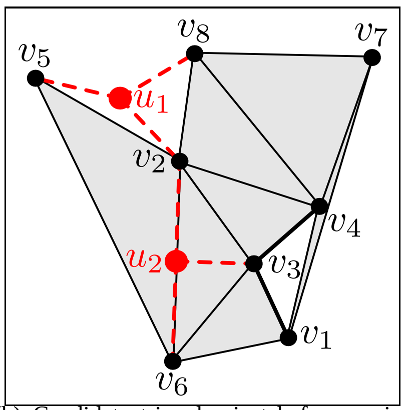
(a) (b) (c)

Fig. 10. Placement of connectors inside a candidate triangle

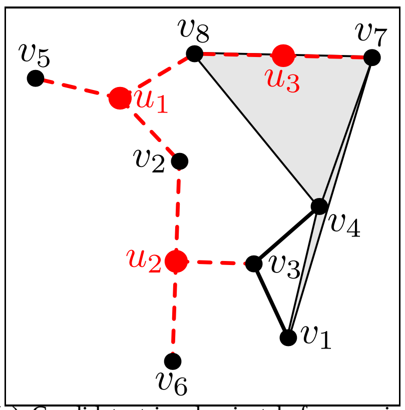
Correctness proof:: To prove the correctness of CIDT, we only need to show that it terminates in finite steps. Indeed in each step, the two steps (a) and (b) in Section IV-B.3 (or equivalently procedure $Placement(t)$) either reduce the number of components in G^* by at least one by connecting an edge or a triangle, or reduce the size of t by a constant portion. The latter is important since the condition that the length of every edge in the Delaunay Triangulation is less than or equal to r is a sufficient condition for connectivity of the network. By reducing the size of candidate triangles step by step, the two steps (a) and (b) in Section IV-B.3 ($Placement(t)$) can make the network connected and terminate the CIDT algorithm in finite time.



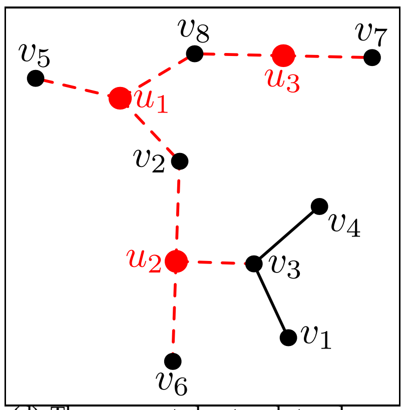
(a) Candidate triangles just before u_1 is added



(b) Candidate triangles just before u_2 is added



(c) Candidate triangles just before u_3 is added



(d) The augmented network topology

Fig. 11. An example of CIDT. The initial topology is the same as that in Figure 3(a).

Example:: An example of CIDT is given in Figure 11, where the initial network topology is the same as that in Figure 3(a).

Time complexity:: Now we analyze the time complexity of the CIDT algorithm. It takes $O(n+e)$ time to identify all connected components in the initial network topology, and $O(n)$ in each step thereafter. The Delaunay Triangulation can be calculated in $O(n \log n)$ time, by using a randomized incremental algorithm [23]. After the initial construction, it takes $O(\log n)$ to insert each additional node. The time complexity of line 4 in the algorithm is $O(n)$, since the number of triangles in the Delaunay Triangulation is $O(n)$. Line 5 takes $O(n)$ time and line 6 takes $O(1)$ time in each step. In line 7, the topology update takes $O(n)$ time in each step. With all the above considered, we conclude that the time complexity of CIDT is $O(n \log n + mn + e)$, where $n = |V|$ and $m = |U|$.

Recall that the time complexity of CIMST is the same as that of MST. The MST can be constructed in $O(n \log n)$ time by first building the Delaunay Triangulation. Therefore, the time complexity of CIMST and CIDT is approximately the same, if $m = o(\log n)$ and $e = o(n \log n)$.

C. Optimizations

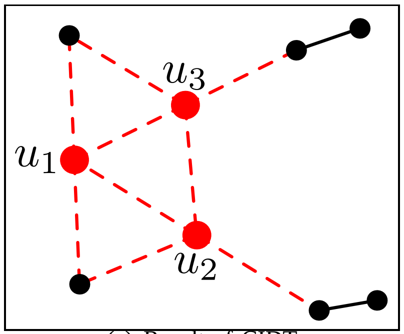
In this section, we introduce two optimization techniques, *Shrink* and *Merge*, to further improve the performance.

1) Shrink: Shrink aims to remove unnecessary connectors in U. The description of the algorithm is given in Figure 12, and an example is shown in Figures 13(a) and 13(b). Line 5 of the algorithm takes $O(n+m+e)$ time, and hence the worst case time complexity of *Shrink* is $O((n+m)^2(n+m+e))$. As we will see in Section V, however, the number of connectors that can be removed is quite small under most cases. Therefore, the time complexity could be much lower.

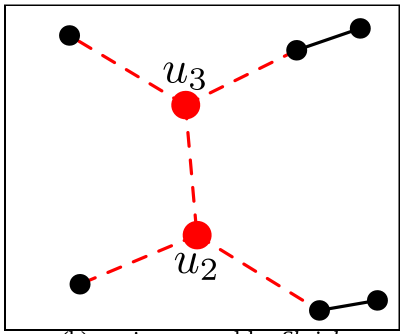
```
Procedure: Shrink($U$)
Input: $U$, a set of connectors;
Output: $s$, the number of connectors removed;
begin
 1:  finish:=false; $s := 0$;
 2:  while (!finish) do
 3:     finish:=true;
 4:     for each node $u \in U$ do
 5:        if ($G^* - \{u\}$ is connected) then
 6:           $U := U - \{u\}$;$G^* := G^* - \{u\}$;
 7:           $s := s + 1$;
 8:           finish:=false;
 9:           break;
10:        end
11:     end
12:  end
end
```

Fig. 12. Shrink algorithm

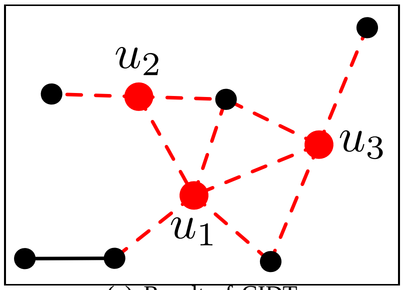
2) Merge: Merge aims to remove unnecessary connectors by replacing two or more connectors with one. The optimization procedure essentially executes CIDT on $G_M = (U, E_M)$, the network of connectors, where $E_M = \{(u, v) : d(u, v) \leq r, u, v \in U\}$. As exemplified in Figure 14, two or more connectors $\{u_i\}$ can be replaced by a new connector u only if the disk $D(u, r)$ can cover every disk $D(u_i, r_{u_i})$, where r_{u_i} is the minimum radius for u_i to connect an edge or a triangle in the *Connector Placement* phase of CIDT (Section IV-B.3). As discussed in Section IV-B, the time complexity of *Merge* is $O(m \log m + ml + e_M)$, where l is the number of merging actions and $e_M = |E_M|$.



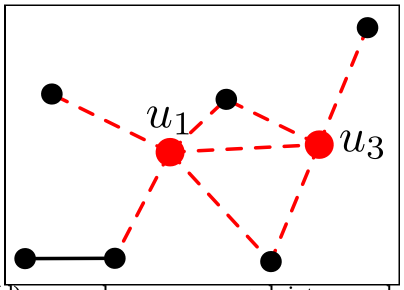
(a) Result of CIDT



(b) u_1 is removed by *Shrink*



(c) Result of CIDT



(d) u_1 and u_2 are merged into one by *Merge*

Fig. 13. Optimizations: *Shrink* and *Merge*

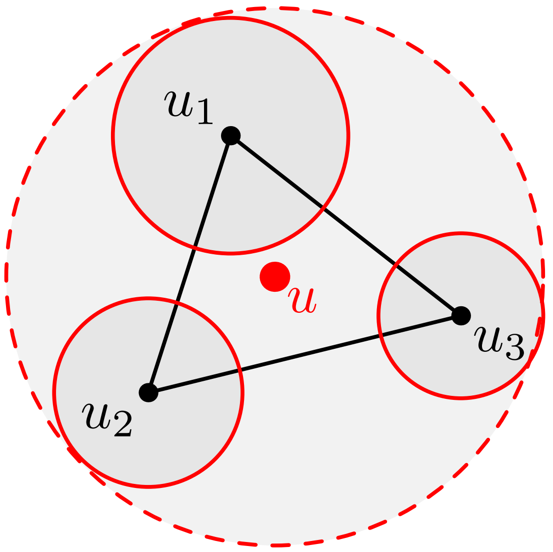


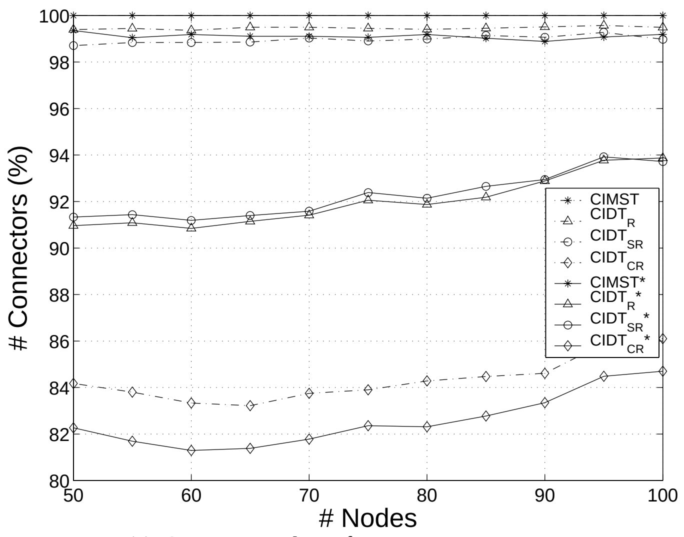
Fig. 14. u_1, u_2 and u_3 can be replaced with u only if $D(u, r)$ can cover $D(u_1, r_{u_1})$, $D(u_2, r_{u_2})$, and $D(u_3, r_{u_3})$

The correctness of both *Shrink* and *Merge* are self-evident, since a connector can be removed or replaced only if it does not affect the connectivity of the network.

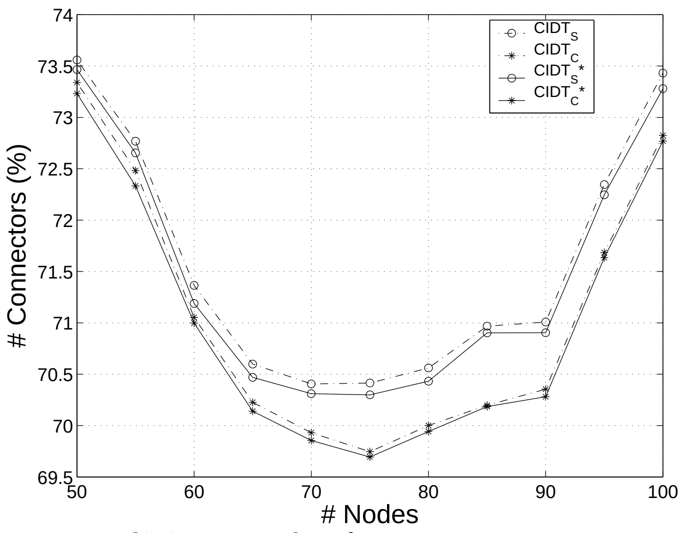
V. Performance Evaluation

In this section, we evaluate the performance of the various versions of CIDT using the *J-Sim* simulator [24]. In the simulation study, all client nodes are uniformly distributed in a square region, and the connector nodes are added as required by various algorithms. Each data point is the average of 1000 simulation runs.

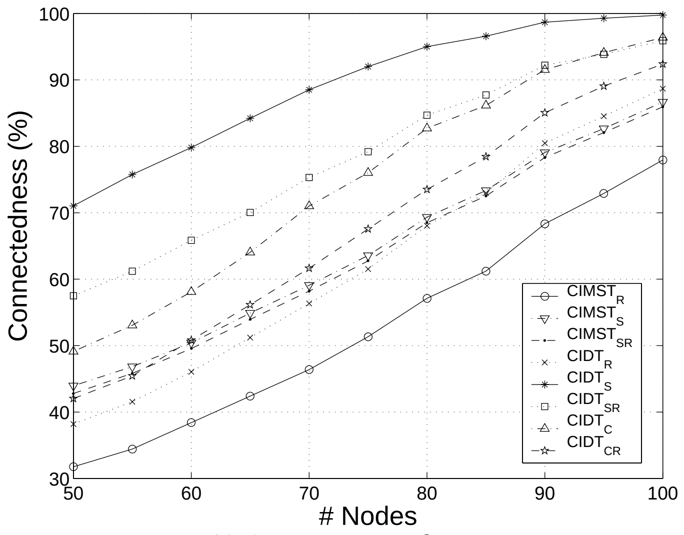
In the first set of simulations, we consider a WSN scenario. The transmission range of both client sensors and connector sensors is $25m$. The size of the region is fixed at $200m \times 200m$. We vary the



(a) Average number of connectors: group 1

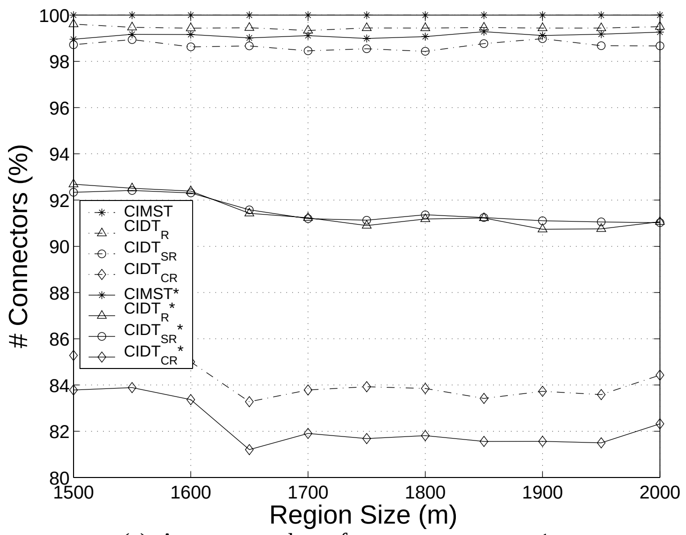


(b) Average number of connectors: group 2

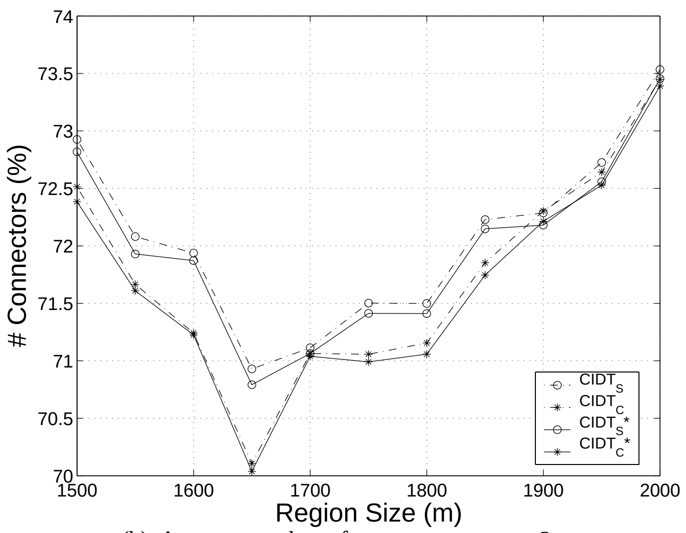


(c) Average connectedness

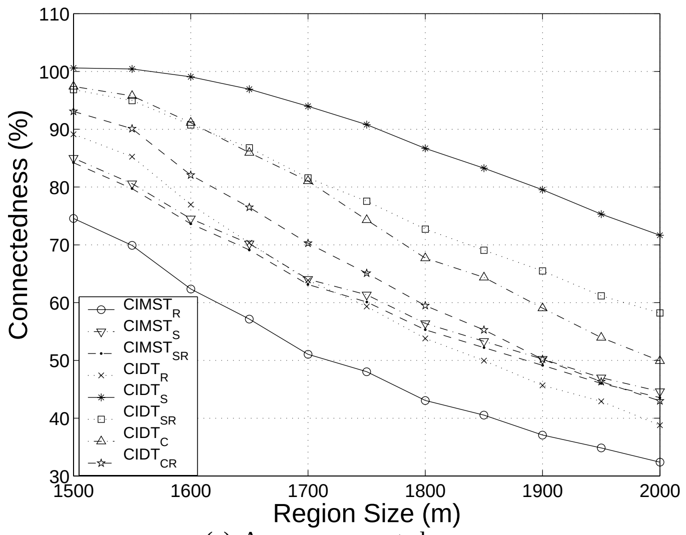
Fig. 15. Simulation results for a region of fixed size



(a) Average number of connectors: group 1



(b) Average number of connectors: group 2



(c) Average connectedness

Fig. 16. Simulation results for a fixed number of nodes

number of clients in the region from 50 to 100. Figures 15(a) and 15(b) show the average connectors needed by the various algorithms to solve the CI problem. The algorithm with a star (*) is the corresponding optimized version. For example, $CIMST$ is the version without use of the two optimization techniques and $CIMST^*$ is the version with both (*Shrink* and *Merge*) applied.

For the sake of comparison, results are normalized with respect to those of the baseline heuristic $CIMST$. We divide the algorithms into two groups. The results of the algorithms in the first group are shown in Figure 15(a). $CIMST$ performs the worst in terms of connectors added, and $CIDT_R$ and $CIDT_{SR}$ are approximately the same. The performance of $CIDT_{CR}$ is notably better compared to others in the same group. The optimization techniques do reduce the number of connectors needed, in particular, by 1% for $CIMST$, by $5\% - 8\%$ for $CIDT_R$ and $CIDT_{SR}$, and by 2% for $CIDT_{CR}$. The results of the algorithms in the second group are shown in Figure 15(b). $CIDT_S$ and $CIDT_C$ performs much better than those in the first group. $CIDT_C$ performs better since it is less “greedy” than $CIDT_S$. The optimization techniques help to some extent, but not significantly.

We also apply aforementioned algorithms to solve the *Connectivity Improvement with Limited Connectors* (CILC) problem. Suppose there are only 3 connectors. We compare the connectedness of the resulting network in Figure 15(c). $CIDT_S$ is clearly the best, and $CIMST$ the worst. Since we fixed the number of connectors, no optimization technique is applied.

In the second set of simulations, we consider a FCS scenario. For ease of exhibition, the transmission range of both ground vehicles (clients) and UAVs (connectors) is $250m$. The number of vehicles is fixed at 50. We vary the size of the square from $1500m \times 1500m$ to $2000m \times 2000m$. The average number of UAVs (normalized) and the average connectedness are shown in Figure 16. Conclusions similar to those in the first set of simulation results can be drawn from this set of results. We have also included the $CIDT_S$ algorithm to solve the CILC problem in simulating a FCS in *J-Sim*. The simulation is conducted based on the real-life traces of ground vehicles provided by SAIC, Inc and with all the path loss and signal attenuation effects caused by the terrain considered. The number of UAVs is limited to 3 in the scenario. Figure 17 gives a snapshot of the FCS simulation.

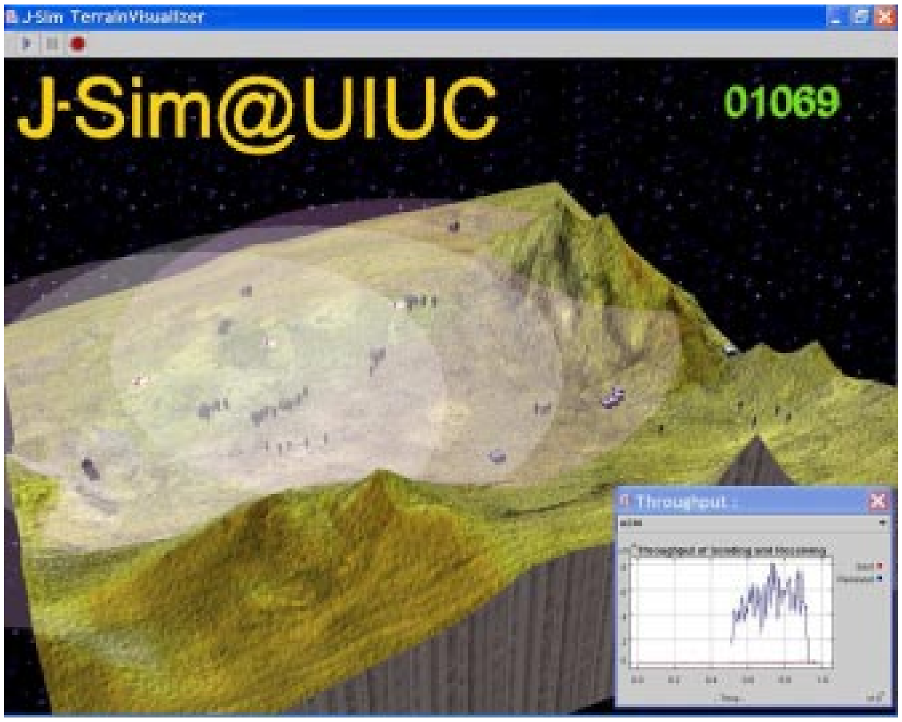


Fig. 17. A snapshot of the FCS simulation in J-Sim. The $CIDT_S$ algorithm is included to determine the fly path of the three UAVs in the (imaginary) battlefield in San Diego, CA. The simulation is conducted based on the real-life traces of ground vehicles provided by SAIC and with the path loss effect due to the real terrain considered. Each circle denotes the coverage area of an UAV.

In summary, all the simulation results suggest that $CIDT_S$ is a better solution algorithm to solving both CI and CILC problems.

VI. Conclusions

In this paper, we have considered the problem of improving connectivity in wireless networks, i.e., how to deploy a set of additional wireless nodes to improve the connectivity of an existing wireless network. The simulation results show that the proposed Delaunay Triangulation based algorithm, $CIDT_S$ renders the best performance in solving the problem.

We are currently investigating several related problems. First, we are studying whether or not there exists any polynomial-time algorithm that can approximate the CI/CILC problems within a constant factor? Second, in wireless mobile ad-hoc networks, the positions of wireless nodes are changing continuously. It is important that the connectors can adjust their positions accordingly, so that the augmented network is kept connected continuously. This problem is not trivial, because (i) the optimal number of connectors may vary with time; and (ii) the speed of the connectors may be too small to keep up with the position trajectory (i.e., the curve that connects all the positions rendered by consecutive invocations of the algorithm). Third, we are investigating the issue of adding connectors so that the network can be k-connected.

References

[1] Piyush Gupta and P. R. Kumar, “Critical power for asymptotic connectivity in wireless networks,” in *Stochastic analysis, control, optimization and applications: a volume in honor of W. H. Fleming*, pp. 547–566. Birkhauser, Boston, MA, 1998.

[2] Mathew D. Penrose, “The longest edge of random minimal spanning tree,” *Annals of Applied Probability*, , no. 7, pp. 340–361, 1997.

[3] Feng Xue and P. R. Kumar, “The number of neighbors needed for connectivity of wireless networks,” *Wireless Networks*, vol. 10, no. 2, pp. 169–181, Mar. 2004.

[4] Mathew D. Penrose, “On k-connecitivity for a geometric random graph,” *Random Structures and Algorithms*, vol. 15, no. 2, pp. 145–164, 1999.

[5] Xiang-Yang Li, Peng-Jun Wan, Yu Wang, and Chih-Wei Yi, “Fault tolerant deployment and topology control in wireless networks,” in *Proc. ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, Annapolis, MD, USA, June 2003, pp. 117–128.

[6] Ning Li and Jennifer C. Hou, “FLSS: a fault-tolerant topology control algorithm for wireless networks,” in *Proc. ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Philadelphia, PA, USA, Sept. 2004.

[7] P. Sass and J. Freebersyser, “FCS communications technology for the objective force,” MITRE Technical Papers, Mar. 2002, http://www.mitre.org/work/tech_papers.

[8] Brent N. Clark, charles J. Colbourn, and David S. Johnson, “Unit disk graphs,” *Discrete Mathematics*, vol. 86, pp. 165–177, 1990.

[9] Peng-Jun Wan and Chih-Wei Yi, “Asymptotic critical transmission radius and critical neighbor number for k-connectivity in wireless ad hoc networks,” in *Proc. ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, Roppongi, Japan, may 2004.

[10] Samir Khuller, “Approximation algorithms for finding highly connected subgraphs,” in *Approximation algorithms for NP-hard problems*, D. S. Hochbaum, Ed. PWS Publishing Co., Boston, MA, 1996.

[11] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: combinatorial optimization problems and their approximability properties*, Springer-Verlag, Berlin, Germany, 1999.

[12] Jean-Marc Robert and Godfried T. Toussaint, “Computational geometry and facility location,” in *Proc. International Conference on Operations Research and Management Science*, Manila, The Philippines, Dec. 1990, pp. B1–B19.

[13] David S. Jonhson, “The NP-completeness column: an ongoing guide,” *Journal of Algorithms*, vol. 3, no. 2, pp. 182–195, 1982.

[14] Kenneth J. Supowit, *Topics in computational geometry*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1981.

[15] Shigeru Masuyama, Toshihide Ibaraki, and Toshiharu Hasegawa, “The computational complexity of the m-center problems on the plane,” *Transactions of the IECE of Japan*, vol. 64E, no. 2, pp. 57–64, Feb. 1981.

[16] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto, “Optimal packing and covering in the plane are NP-complete,” *Information Processing Letters*, vol. 12, no. 3, pp. 133–137, June 1981.

[17] Dorit S. Hochbaum and Wolfgang Maass, “Approximation schemes for covering and packing problems in image processing and VLSI,” *Journal of ACM*, vol. 32, no. 1, pp. 130–136, 1985.

[18] Mihalis Yannakakis, “Node-and edge-deletion NP-complete problems,” in *Proc. Tenth Annual ACM Symposium on Theory of Computing*, San Diego, CA, USA, 1978, pp. 253–264.

[19] Michael R. Garey and David S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman and Co., New York, NY, 1979.

[20] R. Prim, “Shortest connection networks and some generalizations,” *The Bell System Technical Journal*, vol. 36, pp. 1389–1957, 1957.

[21] Seth Pettie and Vijaya Ramachandran, “An optimal minimum spanning tree algorithm,” *Journal of the ACM*, vol. 49, no. 1, pp. 16–34, Jan. 2002.

[22] Steven S. Skiena, *The algorithm design manual*, Springer-Verlag, New York, NY, 1997.

[23] Mark de Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf, *Computational geometry: algorithms and applications*, Springer-Verlag, New York, NY, second edition, 2000.

[24] Hung-Ying Tyan, *Design, realization and evaluation of a component-based software architecture for network simulation,* Ph.D. thesis, The Ohio State University, Columbus, OH, USA, 2001, `http://www.j-sim.org`.

Appendix I
Connector Placement

```
**Procedure**: Insert($u, v, rt$)
**Input:** $u$ and $v$, two vertices of a triangle;
       $rt$, a real number in $[0, 1]$.
**Output:** $p = (p^x, p^y)$;
**begin**
    $p^x := rt \times u^x + (1 - rt)v^x$; $p^y := rt \times u^y + (1 - rt)v^y$;
**end**

**Procedure**: $Placement(t)$
**Input:** $t = (v_1, v_2, v_3)$, a candidate triangle;
**Output:** $p = (p^x, p^y)$;
**begin**
 1:   **if** $Comp(v_2) = Comp(v_3)$
 2:     **if** ($r \geq |e_2|/2$) **then** $p$ :=Insert($v_1, v_3, 0.5$);
 3:     **elseif** ($r \geq |e_3|/2$) **then** $p$ :=Insert($v_1, v_2, 0.5$);
 4:     **else** $p$ :=Insert($v_2, v_1, r/|e_3|$);
 5:   **elseif** $Comp(v_1) = Comp(v_3)$
 6:     **if** ($r \geq |e_1|/2$) **then** $p$ :=Insert($v_2, v_3, 0.5$);
 7:     **elseif** ($r \geq |e_3|/2$) **then** $p$ :=Insert($v_1, v_2, 0.5$);
 8:     **else** $p$ :=Insert($v_1, v_2, r/|e_3|$);
 9:   **elseif** $Comp(v_1) = Comp(v_2)$
10:     **if** ($r \geq |e_1|/2$) **then** $p$ :=Insert($v_2, v_3, 0.5$);
11:     **elseif** ($r \geq |e_2|/2$) $p$ :=Insert($v_1, v_3, 0.5$);
12:     **else** $p$ :=Insert($v_1, v_3, r/|e_2|$);
13:   **else** /* $|Comp(t)| = 3$ */
14:     **if** ($r \geq r_o(t)$) **then**
15:       **if** ($t$ is obtuse) **then** $p$ :=Insert($v_2, v_3, 0.5$)
16:       **else** $p$ :=the center of $t$'s circumcircle;
17:     **else**
18:       **if** ($r \geq |e_1|/2$) **then** $p$ :=Insert($v_2, v_3, 0.5$);
19:       **elseif** ($r \geq |e_2|/2$) **then** $p$ :=Insert($v_1, v_3, 0.5$);
20:       **elseif** ($r \geq |e_3|/2$) **then** $p$ :=Insert($v_1, v_2, 0.5$);
21:       **else** $p$ :=Insert($v_1, v_2, r/|e_3|$);
22:     **end**
23:   **end**
**end**
```

Fig. 18. Connector placement algorithm