Robust Planarization of Unlocalized Wireless Sensor Networks

Fenghui Zhang, Anxiao(Andrew) Jiang, and Jianer Chen

Dept. of Computer Science, Texas A&M Univ. College Station, TX 77843. {fhzhang, ajiang, chen}@cs.tamu.edu.

Abstract—**Wireless sensor networks need very efficient network protocols due to the sensors' limited communication and computation capabilities. Network planarization – finding a planar subgraph of the network that contains all the nodes – has been a very important technique for many network protocols. It first became the foundation of various well known routing protocols, including GPSR, GOAFR and several other protocols. Since then, it has also been used in numerous other applications, including data-centric storage, network localization, topology discovery, etc. However, an important problem remains: network planarization itself is very difficult. So far, efficient planarization algorithms exist only for very restrictive models: the network must be a unit-disk graph, and accurate measurements related to the node locations (e.g., node positions or angles between adjacent links) need to be known. For more practical network models, where the transmission ranges are usually not uniform and sensors cannot obtain their accurate location information via expensive localization devices, no efficient planarization algorithm is available.**

In this paper, we present a novel method that robustly planarizes sensor networks of a realistic model: networks with non-uniform transmission ranges and unlocalized sensors (that is, static sensors whose locations are unknown). Our method starts with a simple shortest path between two nodes, and progressively planarizes the whole network. It achieves both efficiency and a good planarization result. We present two planarization algorithms for different settings. Our results not only solve the planarization problem, but also outperform some known results in the graph drawing research field. We demonstrate the practical performance of our method – as well as its application in topology discovery, – through extensive simulations.

I. INTRODUCTION

Wireless sensor networks usually need very efficient network protocols due to the limited communication and computation capabilities of small sensors. Therefore, it is especially important to exploit the special topological properties of sensornets for the many network functions. A common observation is that the topology of a wireless sensornet usually has a strong correlation with the geometry of the sensor field. That observation has been used in numerous notable applications, including geographic routing [12], etc. In these applications, network planarization has become a very important technique, because a well planarized network not only exhibits the geometric properties of the sensor field, but can also be efficiently utilized in network protocols.

The objective of network planarization is to get a connected planar subgraph of the network that contains all the nodes of the network. To well reflect the geometry of the sensor field, the planar subgraph should contain many links, so that the hop distance (or communication distance) between nodes does not change a lot after planarization. Network planarization first became the foundation of several well known geographic routing protocols, including GPSR [12], Compass Routing [13], GOAFR [14], etc. Such protocols use the faces in the planar subgraph to perform perimeter routing, which guarantees packet delivery. Since then, network planarization has also become a fundamental tool in numerous other applications, including data-centric storage [17], network localization [15] and topology discovery [18] [8] [9]. Here the data-centric storage schemes use the planar graph to help determine on which set of nodes to store each datum, as well as for routing; the network localization schemes can use the properties of planar graphs to facilitate localization; and the topology discovery schemes can use the faces of the planar graph to recognize and locate boundaries and holes in the sensor field. Discovering boundaries and holes in a sensor field is useful for understanding the collected sensor data (because the meaning of sensor data often depends on the type of physical environment where they are collected), for understanding the sensing environment (e.g., building floor plan, transportation network, lakes) and detecting events (e.g., fire in a forest), and for load-balanced routing.

Although network planarization has been proven to be an excellent technique for sensor network protocols, an important problem remains: network planarization itself is difficult. So far, efficient planarization algorithms exist only for network of very restrictive models: unit-disk graphs with accurately known measurements related to the nodes' physical locations. The measurements are the nodes' positions, the angles between all adjacent links, or the lengths of all links. A unit-disk graph (UDG) is a graph where two nodes have a link between them if and only if their physical distance is at most one. So it corresponds to a network where the transmission ranges are the same for all nodes and in all directions. For unit-disk graphs with the accurate location measurements, network planarization can be performed efficiently and distributively by using the techniques based on Gabriel-graph, Relative-Neighborhood graph or Delaunay graph [2]. When the accurate node positions are known and the network is very similar to the unit-disk graph, – specifically, when it is the so called $\sqrt{2}$-quasi unit disk graph, – although no network planarization algorithm is known, the problem can be circumvented to some extent by using "virtual links" [1]. However, the virtual links may need to be realized by long paths in the network, which makes the approach not so useful for many applications. For more general networks, no efficient planarization method is

known. Practical sensor networks often deviate significantly from the unit-disk graph model. The transmission ranges of sensors usually vary substantially in different directions due to reasons including multi-path fading, antenna design, etc., and it is common to observe a variation ratio up to five or more [10]. Also, it is often hard for sensors to obtain accurate location measurements via expensive localization devices (e.g., GPS) or localization algorithms [2]. No efficient network planarization algorithm is currently available for such practical wireless sensor networks.

In this paper, we present a novel method that robustly planarizes sensor networks of realistic models: networks with non-uniform transmission ranges and unlocalized sensors (that is, sensors whose location information is unknown). The method starts with a simple shortest path between two faraway nodes, and progressively planarizes the whole network.

The key for our approach is to solve the so called BIPARTITE PLANARIZATION problem. It has been proved to be NP-hard [7]. We present two planarization algorithms for different settings. We first present a $(2 + 3\epsilon)$-approximation algorithm for this problem. The algorithm is applicable to general networks, and achieves the best known approximation ratio. It outperforms the known results in the graph drawing research field [6].

We then present a fixed parameter tractable (FPT) algorithm that solves the problem exactly (namely, it finds the optimal solution). The algorithm uses the key observation that when a certain parameter is small, the problem can be solved in polynomial time. We show the usefulness of the algorithm to practical networks by simulations.

Since no information on node locations is known, the planar subgraph output by our method is not embedded. It is already sufficient for some applications, such as topology discovery (boundary recognition) [8]. If a plane embedding is needed, it can be obtained by applying existing planar embedding algorithms in graph drawing [4], [16] or in [8] [9] on the planarized network. The embedded graph can then be used for many applications, including geographic routing [11]. We demonstrate the performance of our planarization method and its application to topology discovery by extensive simulations. We show that the planar subgraphs maintain the distance between nodes with small stretches, detect holes and boundaries with a much higher precision than existing methods, and are very robust to the network models.

II. OVERVIEW OF THE PLANARIZATION SCHEME

In this section, we present an overview of the planarization scheme. It consists of five steps, described as follows.

A. Finding A Shortest Path Between Two Faraway Nodes

The first step is to find a shortest path between two faraway nodes. The two faraway nodes can be found with the following common approach: first, randomly choose a node a, use one flooding to build a shortest path tree rooted at a, and find the node b that is the furthest (in hops) from a in the network; then, use a similar method to find the node c that is the furthest

(in hops) from b. b and c are the two faraway nodes we need, and the unique path between b and c in the shortest path tree rooted at b is the shortest path between b and c. The advantage of a shortest path between two faraway nodes is that such a path usually does not twist, regardless of the uniformity of the transmission ranges. One can prove that edges on such path do not cross each other if the sensor nodes in the network have uniform transmission range. In the case when the transmission ranges are very different, it is still very likely that such path does not twist. Thus it has a good planarity property. This observation is validated by extensive simulations. We illustrate this property in Fig. 1. The two networks have drastically different features in the uniformity of transmission ranges, and the path is similar to a straight line in both cases. (When there are holes in the sensor field, the path may not be straight but still spreads out well.)
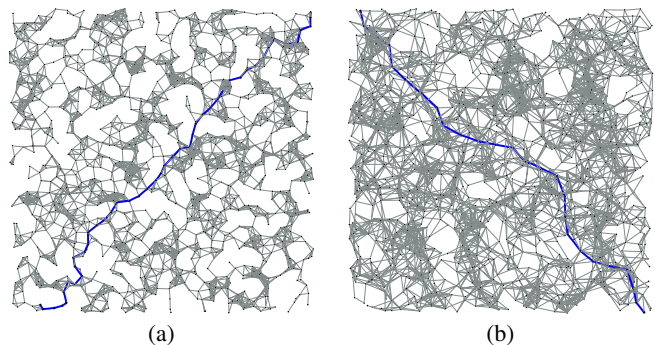

(a)   (b)

Fig. 1. The shortest path between two faraway nodes, in two sensor networks with drastically different transmission ranges. The average degree is 7 in both cases. (a) A sensornet with uniform transmission ranges. (b) A sensornet where the transmission ranges in different directions vary by up to 10 times.

In the following, we will call the shortest path between b and c the *base path*.

B. Building A Shortest Path Tree

The second step is to build a shortest path tree. First, we find a node r_1 that is the furthest away (in hops) from the base path. This can be easily done by viewing the nodes in the base path as a super node, and build a shortest path tree rooted at this super node. Then, we build a (any) shortest path tree rooted at node r_1. See Fig. 2 (a) for an illustration.

C. Planarizing The Network Layer by Layer

In this third step, we planarize part of the network layer by layer. The nodes in the base path are in Layer 1. Recursively, in the shortest path tree rooted at r_1, if a node is the parent of a node in Layer i and is not included in any of the first i layers, then it is in Layer $i+1$. A node that is not the ancestor (in the tree) of any node in the base path is not included in any layer. Let us say that the maximum layer found this way is Layer M. See Fig. 2 (b) for an illustration.

We progressively build a planar graph that includes the nodes in the M layers. First, we process Layer 1 and Layer 2. Let $G = (V_1 \cup V_2, E)$ denote the bipartite graph where the nodes in Layer 1 are in one row and the nodes in Layer 2 are
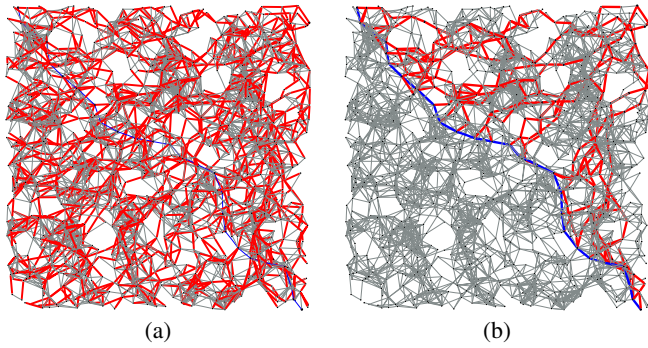
Fig. 2. Red (darker) edges indicate (a) The shortest path tree rooted at node r_1. Here r_1 is near the up right corner of the network. (b) The nodes in the layers, and the edges between layers.
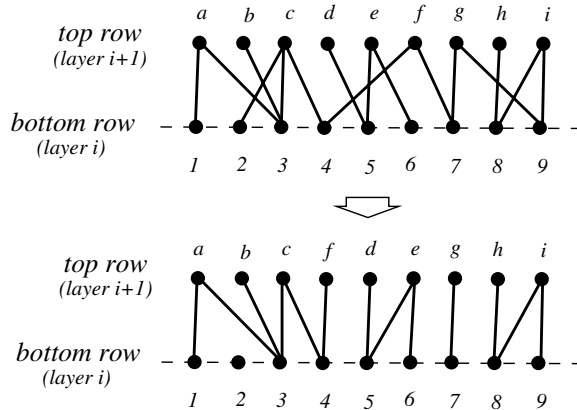

Fig. 3. Planarize two adjacent layers. The upper graph is before planarization, and the lower graph is after planarization.

in the other row. The edges in G are all those network links between Layer 1 and Layer 2. See Fig. 3 for an illustration. In the bipartite graph G, the nodes in the bottom row – which are the nodes in Layer 1 – are placed following their order in the base path. The nodes in the top row – which are the nodes in Layer 2 – are not ordered yet. We then use a planarization algorithm to remove some edges from G, and order the nodes in the top row, so that the remaining edges do not cross each other. (All the edges are straight.) Thus we obtain a planar subgraph between Layer 1 and Layer 2. Then, we process Layer 2 and Layer 3 in the same way, then Layer 3 and Layer 4, $\cdots$, and finally Layer $M-1$ and Layer M. Note that when we are processing Layer i and $i+1$, the nodes in Layer i (the bottom row) have been ordered. So the same algorithm can be used $M-1$ times. All the edges we keep in these $M-1$ steps form a planar graph.

The general idea is that the base path and the shortest path tree rooted at r_1 both act as good references for planarization. By processing the nodes layer by layer, we *comb through* the network and obtain a planar subgraph. The planarization algorithm for planarizing the edges between two adjacent layers is the *key operation* in our method. We present the details of the algorithm in the following sections.

D. Building A Second Shortest Path Tree and Planarizing The Network

The planar graph built so far is a skeleton of the network covering part (often about half) of the sensor field. In this step, we build a second shortest path tree and planarize more of the network. This second tree is rooted at the node r_2 that is the furthest (in hops) from the node r_1. The planarization process is exactly the same as the process in the previous step (namely, the third step), except that here node r_2 replaces node r_1, and we do not include in the layers here those nodes that have been included in Layer 2 through Layer $M-1$ in the previous step. See Fig. 4 (a) for an illustration. The planar graphs built in this step and the previous step together form a large planar subgraph that covers most of the sensor field.
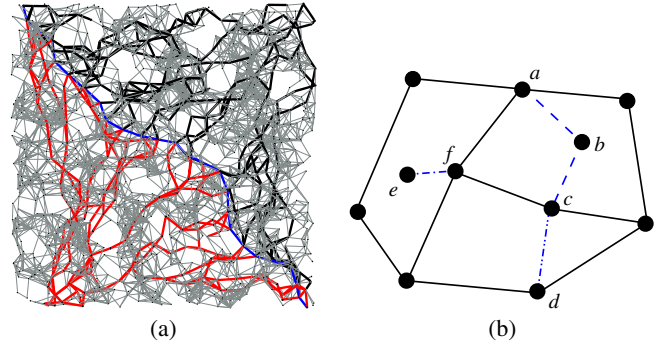

Fig. 4. (a) After finishing the first tree, we build the second and planarize it similarly. Edges of the second tree are red (lighter); (b) The path $a \rightarrow b \rightarrow c$, the edge $c - d$, and the edge $e - f$ can be added into the planar graph.

E. Refining The Planar Graph

The planar graph built so far is a skeleton of the network, which usually covers the whole sensor field. Those nodes outside it are usually within a few hops from it. To include all nodes into the planar graph and to include more edges, three simple steps are performed. First, if a node in the planar graph – which we shall call G_{planar} – finds that it has many 1- or 2-hop neighbors outside G_{planar}, it uses a 4-hop (one can use 3- or 5-hop as well) localized flooding to add one or more paths to G_{planar}, as long as the new path connects nodes in the same face and therefore preserves the planarity of G_{planar}. Note that the previous planarization steps already tell us what the faces in the planar graph are, so this operation is easily doable. Second, if there are still nodes outside G_{planar}, they connect themselves to G_{planar} via small trees, which is a simple operation. The trees preserve the planarity of the graph. At this moment, the planar graph contains all the nodes. Then, to add more edges to G_{planar}, the nodes add their incident edges to it if the new edge connects two nodes in the same face (and therefore preserves the planarity). Now we get the final planar subgraph of the network. See Fig. 4 (b) for an illustration on how the refining is done.

III. AN APPROXIMATION PLANARIZATION ALGORITHM

In the previous section, the planarization scheme is presented. The *key* operation is how to planarize two adjacent

layers. That is also the only part of the scheme whose detail has not been specified. Naturally, our objective is to remove as few edges as possible during planarization, because a dense planar graph is desirable. In this section, we present an approximation algorithm to this NP-hard problem.

We formally define the problem as follows:

> BIPARTITE PLANARIZATION problem (BiPP): In a bipartite graph $G = (V_1 \cup V_2, E)$, the nodes in the bottom row V_1 are already linearly ordered, and the nodes in the top row V_2 are not (See Fig. 3 for an example). How to remove some edges from G, and linearly order the nodes in the top row V_2, so that no two edges cross each other? The objective is to minimize the number of removed edges.

In the graph drawing research field, this problem is also called the ONE SIDED TWO-LAYER PLANARIZATION problem. It is known to be NP hard, even when the nodes in V_2 have degrees at most two and the nodes in V_1 have degrees at most one [7]. The best existing solution that runs in polynomial time is a 3-approximation algorithm [6], i.e., it returns a solution that removes at most three times the number of edges an optimal solution would remove. In this section, we present a new algorithm whose approximation ratio can be arbitrarily close to 2, thus improving the best known result.

In the rest of the paper we assume that each layer (row) is to be placed on a horizontal line. Let us define a few terms. For any two nodes v_1 and v_2 in the bottom row V_1, if they are adjacent (in the sense of the given ordering of the nodes in V_1), then we imagine there is a *virtual edge* between v_1 and v_2. We say that a node $u \in V_2$ *covers* a virtual edge e if u has neighbors in the graph G that are on both the left side and the right side of e in the bottom row. Such a node u is called a *cover node* of e. The number of nodes in V_2 covering e is called the *cover number* of e. All the edges incident to a cover node of e are called the *walls* of e. Note that every cover node of e is incident to at least two walls of e.

Our approximation algorithm is based on the following observations.

Observation 1: In any solution to the BIPARTITE PLANARIZATION problem, the cover number of any virtual edge e is at most 1. Therefore, if the cover number of e is y in the graph G, any solution must remove at least $y - 1$ walls of e.

We illustrate the above definitions and properties in Fig. 3: (1) In the upper graph, e, f are cover nodes of the virtual edge $\{5, 6\}$ and make its cover number be 2. (2) If c is the only cover node for virtual edge $\{3, 4\}$ in an optimal solution, neither of the walls $\{c, 3\}$ and $\{c, 4\}$ is removed by that solution.

Therefore, when the cover number of a virtual edge is large, if we remove two walls around e for each of the cover nodes, we would have removed no more than twice the number of edges removed by any solution plus one. This technique is used in our algorithm.

On the other hand, if the cover numbers of all the virtual edges are relatively small, we can solve the problem efficiently with the divide and conquer technique. The following observation is the basis for the divide and conquer technique.

Observation 2: If the cover number of a virtual edge e is y in the graph G, then in any solution, at most one of e's cover nodes can keep all its corresponding walls around e. For all the other cover nodes of e, each of them must remove all the walls on at least one side of e. Therefore, if we enumerate all the possible ways to solve the conflicts at e in any solution, there are at most $y2^{y-1} + 2^y$ cases to consider. (Specifically, for each cover node, we first consider if its walls should be removed; if yes, we consider which side of the walls to remove. So there are $y2^{y-1} + 2^y$ cases.)

If a virtual edge e has no cover node in a solution, the nodes in the bottom row can be separated into two parts: the left of e and the right of e. Then every node in the top row is adjacent to nodes in only one of the two parts. So in that case, the problem can be solved for the two subgraphs separately. If in an optimal solution, e has one cover node, then the following observation tells us that we can still divide the problem into two parts.

Observation 3: Suppose that in an optimal solution, a virtual edge e has exactly one cover node u. If the wall w_l (respectively, w_r) incident to u is the closest edge to e from the left(respectively, right) side, then w_1, w_2 must have not been removed by that optimal solution (as shown in Fig. 3). In that case (assuming that we have guessed this case to be true), when we search for the optimal solution, we can mark the two walls closest to e from each side (namely, w_1 and w_2) to be *unremovable* (namely, we do not remove them in the algorithm).

The approximation algorithm is shown in Algorithm 1.

Algorithm 1 APX-BiPP

Input: $G = (V_1 \cup V_2, E), \epsilon$
Output: G_{planar}: A solution to the BiPP.
1: **repeat**
2:    $e \leftarrow$ a virtual edge in V_1 with cover number $> 1/\epsilon$
3:    for each cover node of e, remove a wall incident to that node from each side of e
4: **until** no virtual edge in V_1 has cover number larger than $1/\epsilon$
5: call the procedure **Exact-BiPP**(G) and return the result.

Let us first look at the subroutine **Exact-BiPP**, in any case the edge e will be a separator for the problem in the sense that no two nodes on both side of e share a neighbor in V_2 except v, the one we decided to keep "over" e. In that case $\{v, u_1\}$ and $\{v, u_2\}$ are both kept in the final solution and marked unremovable, the two subproblems formed in line 6 of **Exact-BiPP** will not affect each other. In the case e has no cover node in the solution, it is clear that we have split the problem into two independent subproblems. Since we enumerate all possible cases at e in the recursive stage, we will get the optimal solution.

Assuming that the maximum cover number for virtual edges in V_1 is c and $T(n)$ is the running time of the

Algorithm 2 Exact-BiPP

Input: $G = (V_1 \cup V_2, E)$

Output: G_{planar}: A solution to the BiPP; and the number of edges removed

1: Let n be the number of nodes in V_1. If $n < 5$, solve the problem in the brute force way, and return the solution.
2: $e \leftarrow$ the $\lfloor n/2 \rfloor$-th virtual edge in V_1
3: $y \leftarrow$ the cover number of e
4: **for all** $y2^{y-1} + 2^y$ cases at e (see Observation 2) **do**
5:    skip this iteration if in this case, an unremovable edge is to be removed
6:    the problem is now split into two disjoint subproblems of roughly the same size, recursively call **Exact-BiPP** on them.
7:    record the solution of this case if it removes the least number of edges among all cases considered so far,
8: **end for**
9: remove edges according to the best solution got above
10: **return** G and the number of edges removed

---

algorithm where $n = |V_1|$. We have the recurrence relationship $T(n) < 2 \times c2^c T(\frac{n}{2})$. This recursion has a solution $T(n) < (2c2^c)^{\log n} = n^{c+\log c + O(1)}$. Hence we have the following lemma immediately.

Lemma 1: The algorithm **Exact-BiPP** finds the optimal solution for the BIPARTITE PLANARIZATION problem and runs in time $n^{c+\log c + O(1)}$ where c is the maximum cover number for all virtual edges in V_1 and n is the number of nodes in V_1.

In the following theorem, we prove the algorithm **APX-BiPP**'s approximation ratio, $2 + 3\epsilon$, and its polynomial time complexity. The parameter ϵ can be a arbitrarily small positive number, so we assume that $\epsilon \leq 1/3$.

Theorem 1: The planarization algorithm **APX-BiPP** is a $(2 + 3\epsilon)$-approximation for the BIPARTITE PLANARIZATION problem. It runs in time $O(|V_1|^{\frac{1}{\epsilon} + \log \frac{1}{\epsilon} + O(1)} + |E||V_1|)$.

   Proof: First it is simple to see that the algorithm produces a solution for the problem.

Let us look at the approximation ratio. In the recursive stage (subroutine **Exact-BiPP**) of the algorithm, by Lemma 1 we get the exact solution.

While in the preprocessing stage (lines 1 through 4 in **APX-BiPP**), there are $a(e) > 1/\epsilon$ nodes in V_2 covering e, and at most one of them can be the cover node of e in any solution. We removed $2a(e)$ edges over e. That is at most $a(e) + 1$ more than the number of edges removed by an optimal solution S.

Suppose that in the preprocessing stage (lines 1 through 4 in **APX-BiPP**), we have run the loop m times. Let M be the total number of edges removed, we have $M > 2m/\epsilon$. Let the residual graph be B'. Among these M edges at most $M/2 + m$ are unnecessary, in other words, S (the optimal solution) would have to remove at least $M/2 - m$ edges. Therefore the number of edges removed by S, denoted by $R(B)$, is then at least $M/2 - m + R(B'')$ where B'' is the residual graph after we removed these $M/2 - m$ edges according to S. Since in the approximation stage we have removed all walls

at those virtual edges, B' is a subgraph of B''. Hence we have $R(B') \leq R(B'')$ and $R(B) \geq M/2 - m + R(B')$. While in our approximated solution the number of edges removed, denoted by $A(B)$, is exactly $M + R(B')$. Hence we have

$$\frac{A(B)}{R(B)} \leq \frac{M + R(B')}{M/2 - m + R(B')} = 2 + \frac{4m - 2R(B')}{M - 2m + 2R(B')}$$
$$\leq 2 + \frac{4m}{2m/\epsilon - 2m} = 2 + \frac{2\epsilon}{1 - \epsilon} \leq 2 + 3\epsilon.$$

Now we show that the algorithm runs in polynomial time. The preprocessing stage takes time $O(|E||V_1|)$. In the stage when we call **Exact-BiPP**, since the cover numbers of all virtual edges in V_1 are bounded by $\frac{1}{\epsilon}$, by Lemma 1 the running time is upper bounded by $n^{\frac{1}{\epsilon} + \log \frac{1}{\epsilon} + O(1)}$. Therefore the total running time is then $O(|V_1|^{\frac{1}{\epsilon} + \log \frac{1}{\epsilon} + O(1)} + |E||V_1|)$. ∎

IV. FIXED PARAMETER TRACTABLE ALGORITHM FOR OPTIMAL PLANARIZATION

In our extensive simulations, we observed that while planarizing the subgraph induced by two adjacent layers, the number of edges that need to be removed is usually much smaller than the number of nodes in those two layers, for a network of moderate density (average degree roughly between 6 and 12). If the network is very dense, we can use a simple preprocessing operation to reduce the edge density to the moderate level. (Details of the operation will be discussed later.) Therefore a question remains: can we use that observation to practically improve the planarization algorithm?

In recent years, a new approach called parameterized computation has been proposed to solve NP hard problems by exploiting small parameters [5]. Let k be a parameter in a parameterized problem. We say that the problem is *fixed parameter tractable* (FPT) if it can be solved optimally in time $O(f(k)n^{O(1)})$, where n is the input size and $f(k)$ is a function of k. When k is bounded, not only is the time complexity polynomial, but it usually also grows much slower than $O(n^k)$ when n increases. Quite a few NP-hard problems have been proved to be in FPT with effective algorithms. For example, the VERTEX COVER problem with parameter k as the cover size can be solved in time $O(1.286^k + n^3)$ [3]. The difference between the approximation approach and a parameterized approach is that the latter gives optimal solutions.

In [6], the authors showed that the BIPARTITE PLANARIZATION problem (also called the ONE SIDED TWO-LAYER PLANARIZATION problem) is fixed parameter tractable when the parameter k is an upper bound for the number of edges to be removed. They developed an FPT algorithm for the problem running in time $3^k n^{O(1)}$. In this section, we present an improved FPT algorithm running in time $(2 + \gamma)^k n^{O(1)}$, where γ can be an arbitrarily small positive number.

Formally, the parameterized version of the BIPARTITE PLANARIZATION problem can be stated as follows:

> Given an instance of the BIPARTITE PLANARIZATION problem and a parameter k, either find a solution to the instance that removes at most k edges, or report that no such solution exists.

Let us first define a few terms. Let $u \in V_1$ be the leftmost neighbor of a node $z \in V_2$, and $v \in V_1$ be the rightmost. We call the pair of edges $\{u, z\}, \{z, v\}$ an *arc*, and denote it by $z(u, v)$. We say that the arc $z(u, v)$ *conflicts* with an edge $\{w_1, w_2\}$ if it is necessary to remove $\{w_1, w_2\}$ in order to keep both edges $\{z, u\}, \{z, v\}$ in a solution. To help simplify the following discussion, for each pair of arcs $z(u, v), z'(u, v)$ – i.e., they share the same leftmost and rightmost neighbors u, v in V_1 – we also consider $\{z, u\}$ a conflict edge of $z'(u, v)$. Similarly $\{z', u\}$ is also a conflict edge of $z(u, v)$.

The *conflict number* of an arc $z(u, v)$ is the number of edges that conflict with the arc. (See Figure 5 (a) for an illustration of the terms.)

The conflict number of a given arc $z(u, v)$ is simply the summation of the following two numbers: (1) the number of edges incident to nodes (exclusively) between u, v in V_1 but not z; (2) the number of arcs that of the forms $z'(u, v)$ (where $z' \neq z$), i.e., they share the same leftmost (rightmost) node in V_1. The set of edges that are conflicting with a given arc can also be decided easily by definition.
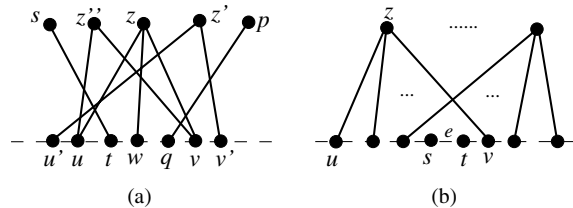

Fig. 5. (a) $z(u, v)$ is an arc; the edges $\{p, q\}, \{s, t\}, \{z'', u\}$ conflict with $z(u, v)$; the conflict number of $z(u, v)$ is then 3; $z'(u', v')$ strictly covers $z(u, v)$ and has a larger conflict number 7. (b) cover number and conflict number at virtual edge $e(s, t)$: there are at least $f - 1$ edges incident to nodes between u and s but not z.

The following lemma shows the relationship between cover numbers and conflict numbers.

Lemma 2: If there is a virtual edge e in V_1 with cover number f, there must exist an arc that has conflict number at least $f - 1$. In other words, if no arc has conflict number larger than $f - 1$, no virtual edge in V_1 will have cover number larger than f.

Proof: Suppose the leftmost walls of e are incident to u in V_1. Let $z(u, v)$ be the arc such that v in V_1 is the furthest away to the right of u. Hence $\{z, u\}$ is a leftmost wall of e and v is the right-most neighbor of the cover nodes adjacent to u. Each of the $f - 1$ cover nodes (other than z) of e either is adjacent to u and has no neighbor exclusively to the right of v, or has its leftmost wall exclusively to the right of u. By definition, each of these $f - 1$ cover nodes (other than z) of e will contribute at least 1 to $z(u, v)$'s conflict number. Either their rightmost walls (if they are adjacent to u but not v) or their leftmost walls will be conflict edges of $z(u, v)$. The conflict number of $z(u, v)$ is, therefore, at least $f - 1$. ∎

We present the FPT algorithm as Algorithm 3. The constant β is any predefined positive number that is no smaller than 1. The algorithm returns a solution if there exists a solution that removes at most k edges, and returns **false** otherwise. (With a little abuse of notations, when the algorithm returns **true**,

it returns the solution as well.) Note that when a solution is found, the conflict number of every arc becomes 0.

Algorithm 3 FPT-BiPP (where $\beta \geq 1$ is any given number)

Input: $G(V_1 \cup V_2, E), k$

Output: G_{planar}: A solution removing no more than k edges.

1: $z(u, v) \leftarrow$ the arc with the maximum conflict number $c(z)$.
2: **if** $c(z) = 0$ **then return** true
3: **if** $k < c(z)$ **then return** false
4: **if** $c(z) \leq \beta$ **then**
5:   planarize G using the procedure **Exact-BiPP**. (That is, run **Exact-BiPP**(G).)
6:   **if Exact-BiPP**(G) removes at most k edges, **return** true; **else return** false
7: **end if**
8: $E(z) \leftarrow$ the set of edges conflicting with $z(u, v)$
9: **if FPT-BiPP**$(G(V_1 \cup V_2, E \setminus E(z)), k - c(z))$ **return** true;
10: **if FPT-BiPP**$(G(V_1 \cup V_2, E \setminus \{\{z, u\}\}), k - 1)$ **return** true;
11: **if FPT-BiPP**$(G(V_1 \cup V_2, E \setminus \{\{z, v\}\}), k - 1)$ **return** true;
12: **return** false

The following theorem proves the correctness and complexity of the algorithm.

Theorem 2: The algorithm **FPT-BiPP** will either find a solution for the BIPARTITE PLANARIZATION problem by removing at most k edges, or correctly report that there is no such solution. The running time of the algorithm is upper bounded by $(2 + \frac{1}{\beta})^k n^{\beta + \log \beta + O(1)}$.

Proof: The correctness of the algorithm is straightforward. Let us look at the running time.

During the calling of the routine **Exact-BiPP** in line 5 (that is, when $c(z) \leq \beta$), since the conflict numbers of the nodes in V_1 are upper bounded by β, by Lemma 2, the cover number of the virtual edges in V_1 will be bounded by $\beta + 1$. Again by Lemma 1, the running time of **Exact-BiPP** in this algorithm will be bounded by $T_1 = n^{\beta + \log \beta + O(1)}$.

As for the recursive calls in lines 9, 10 and 11, we branch into three cases where k decreases by 1, 1 and $c(z)$, respectively. Since we enter this stage only if $c(z) > \beta$, if we use $T(k)$ to denote the running time of the algorithm, the following recurrence relationship holds for $T(k)$: $T(k) \leq 2T(k-1) + T(k-\beta)$.

We use induction to show that $T_1(2 + \frac{1}{\beta})^k$ is an upper bound for $T(k)$. In the case when $k < c(z)$, we return false right away. If $c(z) \leq k \leq \beta$, we already have $T(k) \leq T_1$. Hence when $k \leq \beta$, we always have $T(k) \leq T_1(2 + \frac{1}{\beta})^k$.

Suppose for $\beta \leq k < t$ we have $T(k) \leq T_1(2 + \frac{1}{\beta})^k$. Using basic calculus, it is easy to verify that when $\beta \geq 1$, $2(2 + \frac{1}{\beta})^{\beta - 1} + 1 \leq (2 + \frac{1}{\beta})^\beta$. Thus we have $T(t) \leq 2T(t - 1) + T(t - \beta) \leq 2T_1(2 + \frac{1}{\beta})^{t-1} + T_1(2 + \frac{1}{\beta})^{t-\beta} \leq T_1(2 + \frac{1}{\beta})^{t-\beta} \left[2(2 + \frac{1}{\beta})^{\beta - 1} + 1 \right] \leq T_1(2 + \frac{1}{\beta})^t$ Therefore we have proved that the running time of the algorithm is bounded by $(2 + \frac{1}{\beta})^k n^{\beta + \log \beta + O(1)}$. ∎

The exponential part of the running time of the algorithm **FPT-BiPP** can be arbitrarily close to 2^k by choosing a large

enough value for β.

Although the problem is NP-hard, the FPT algorithm solves the problem in polynomial time when the parameter is small, i.e., the exponential part of the running time is independent of the number of the nodes in the input graph. As we will show later in simulations, the number of edges to be removed is actually very small (less than 15 in most cases for networks of up to 2500 nodes) for networks with practical models.

V. Implementation and Simulations

We conducted extensive simulations to test the performance of the planarization method. The performance has been very stable for different network models, sensor deployment methods, network sizes and sensor densities. In the following, we present some typical simulation results. A planarized network has numerous applications, including topology discovery, localization, geographic routing, etc. (For geographic routing, embedding is needed and such embedding algorithms are available [4], [8], [9], [16].) We illustrate the performance of our result by showing its application to topology discovery. We show that our planarized network can locate holes and outer boundaries of the sensor fields very accurately, significantly improving the known results on topology discovery.

Let us explain the distributed implementation of our planarization method. The method utilizes a few shortest path trees, and a practically limited amount of localized flooding to refine the final result. (The total cost of the localized flooding is about the same as flooding the network once or twice.) Both building shortest path trees and localized flooding are very mature techniques in networking. Both algorithms presented in the paper planarize the network layer by layer, which naturally corresponds to a distributed implementation. While planarizing two adjacent layers, we take the simple approach of letting one node in the two layers act as a proxy and run the algorithm in a centralized way. (The nodes in the two layers can easily send their information to the proxy node via the corresponding shortest path tree.) We comment that our algorithms take the divide and conquer approach, so it is simple to decentralize its implementation.

The faces in the planarized network are always very clear throughout the planarization process. That is because nodes are planarized layer by layer, and the planarization algorithm sorts the edges incident to each node by ordering the nodes in the two adjacent layers. That makes the refinement step and the topology discovery application of using faces to recognize holes/boundaries very simple.

Our method works well for networks of moderate or sparse edge densities. For dense networks (average degree 15 or more), the following simple preprocessing approach can effectively reduce the edge density and edge crossing: for every maximal clique in the network, we remove some edges from it so that the remaining edges form a star. Note that the average degree of a planar graph is always less than 6. So such an edge-reduction preprocessing step goes along well with planarization.

We have presented two planarization algorithms: an approximation algorithm and an FPT algorithm. They have very similar performance in practice. Due to the space limitation, we present the simulation results for the FPT algorithm only. (The results for the approximation algorithm are very close.) In most simulations, the optimal planarization solution removes less than 15 edges for all the layers. So by setting $k = 15$ or a little above, the FPT algorithm finds the optimal solution and maintain a low time complexity at the same time.

A. Network Planarization

We randomly deploy N sensors uniformly randomly in a 15000×15000 square area. The network follows the quasi unit disk graph (quasi-UDG) model: two nodes do not have a link if their distance is greater than R, have a link if their distance is less than r, and have a link with probability $1/3$ if their distance is between r and R. (Here $r \leq R$.) Let $\alpha = R/r$. When N and α are given, we adjust r and R to obtain the desired average node degree. To make the sensor field non-trivial, we also randomly place holes in the network. Our reported results are for two holes of radius about $2R$. For each configuration, 1000 networks are generated and measured.

A typical planarization result is shown in Fig. 6. To quantitatively analyze the performance of planarization, we measure the stretch ($str.$) of hop distance and its standard deviation σ. Let u, v be two nodes, whose hop distance is $h(u, v)$ before planarization and is $h'(u, v)$ after planarization. The *multiplicative stretch* is defined as $h'(u, v)/h(u, v)$, and the *additive stretch* is $h'(u, v) - h(u, v)$. To better characterize the stretch of different node pairs, we measure the multiplicative stretch for nodes whose hop distance is greater than 1/4 of the network diameter, and measure the additive stretch for the others. The results are shown in Table I where d is the average degree of the input networks, d_p is the average degree of the planar spanning subgraph found and D is the average diameter of the input networks.

TABLE I
PLANARIZATION RESULTS FOR $N = 1500$

	d=7			d=9			d=11		
	Additive stretch for nodes within $Diameter/4$ hops								
α	D	$str.$	σ	D	$str.$	σ	D	$str.$	σ
1	48.3	4.03	9.80	40.1	3.56	8.04	36.1	3.36	7.27
2	36.2	3.45	8.11	30.6	3.28	7.19	27.4	3.24	6.66
5	28.9	3.23	6.76	24.0	3.05	6.25	21.5	2.98	6.00
10	27.0	3.11	6.82	23.7	3.01	6.13	21.4	2.96	5.83
	Multiplicative stretch for nodes more than $Diameter/4$ hops apart								
α	d_p	$str.$	σ	d_p	$str.$	σ	d_p	$str.$	σ
1	3.5	1.27	0.57	3.8	1.28	0.56	3.8	1.30	0.57
2	3.7	1.31	0.54	3.8	1.35	0.56	3.9	1.38	0.58
5	3.7	1.36	0.55	3.8	1.40	0.55	3.8	1.43	0.56
10	3.7	1.36	0.54	3.8	1.40	0.55	3.8	1.43	0.57

We see that for networks of very different connectivity models and densities, the stretch is constantly small. This shows that the planarized networks well preserved the connectivity of the original networks.
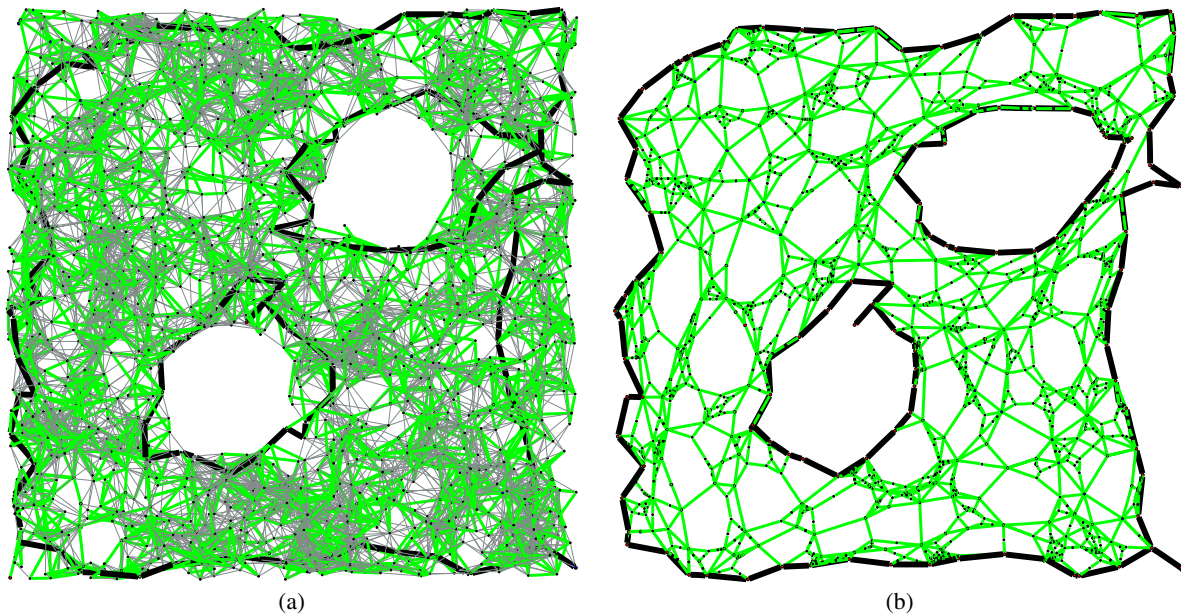
(a)	(b)

Fig. 6. Wireless sensor network and its planarization. (a) The dark (green and black) edges are the planarized network. The light (grey) edges are the remaining edges in the original network. The original network is a quasi-UDG with $N = 2000$ nodes and average degree 12, where transmission ranges vary by as much as $R/r = 5$. (b) A plane embedding of the planarized network.

B. Topology Discovery

One important application of network planarization is topology discovery [18] [8] [9]. In topology discovery, we use the large faces in a planarized network to find holes and the outer boundary of the sensor field. The importance of such prominent features in sensor fields has been presented earlier in the Introduction. Mainly, they help us learn about the sensor field, detect obstacles and abnormal events, interpret the sensed data, and balance routing load.

In our simulations, we place two circular holes of radius $2R$ in the sensor field, where the sensors cannot be deployed. We comment that the actual holes are very irregular and look far from circles, because of the random deployment of nodes and the sparsity of edges. By checking large faces (whose boundaries containing 20 edges or more), we are able to identify the faces close to he boundaries of holes and the outer boundary in more than 95% cases for all configurations.

Four typical results are shown in Fig. 7. To quantitatively analyze the performance of topology discovery, we compare the Euclidean length of the actual hole or outer boundary, l, to the Euclidean length of the face, l', surrounding it. Both l and l' are computed offline using real coordinates of the nodes. (Note that the actual boundary consists of a set of fractional segments of edges since edges can cross.) The results are shown in Table II where σ is the standard deviation of l'/l and m is the average of the total number of edges removed to planarize both shortest path trees for each network.

We see that the length ratio is usually close to 1. This shows the excellent performance of the results. Our results compare favorably with the known results on topology discovery. In [8], [9], [18], the authors presented very nice algorithms for detecting holes and outer boundaries. Their algorithms work for large holes (radius are 5 times the communication range

of the sensors or more) or very dense networks (e.g., average degree larger than 15). In comparison, our method detects holes whose radius are twice the communication range or more, and works for both low and high densities. So the improvement in performance is actually remarkable.

VI. CONCLUSION

In this paper, we present a new network planarization approach. It stably achieves very nice performance for different network models, and has a very nice application to topology control. In addition, the main planarization algorithms outperform existing theoretical results in the graph drawing area. It remains as our future research to study planarization for sensornets with mobile nodes and dynamic links.

REFERENCES

[1] L. BARRIERE, P. FRAIGNIAUD AND L. NARAYANAN, Robust position-based routing in wireless ad hoc networks with unstable transmission ranges, in *Proc. DialM*, pp. 19-27, 2001.
[2] J. BRUCK, J. GAO AND A. JIANG, Localization and routing in sensor networks by local angle information, in *Proc. MobiHoc*, pp. 181-192, 2005.
[3] CHEN, IYAD A. KANJ, WEIJIA JIA, Vertex Cover: Further Observations and Further Improvements, *J. Algorithms 41(2)*, pp. 280-301, 2001.
[4] R. DAVIDSON AND D. HAREL, Drawing graphs nicely using simulated annealing, in *ACM Transactions on Graphics 15*, pp. 301-331, 1996.
[5] R. DOWNEY AND M. FELLOWS, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

TABLE II
HOLE DETECTION RESULTS FOR $N = 1500$

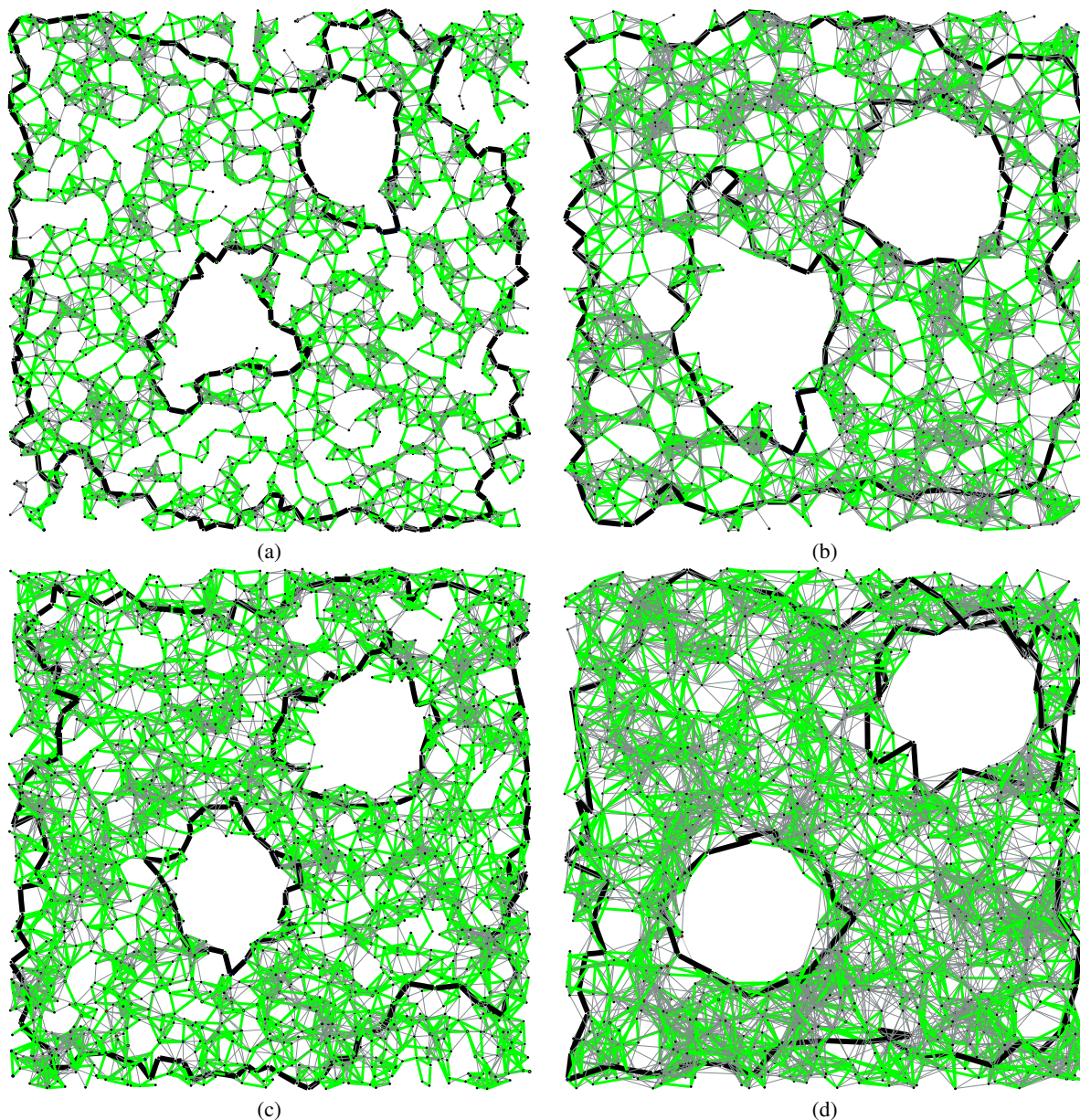|	d=7			d=9			d=11		
α	l'/l	σ	m	l'/l	σ	m	$l'/l.$	σ	m
1	1.02	0.06	4.8	1.15	0.31	9.5	1.20	0.05	14.6
2	1.13	0.05	6.4	1.22	0.04	10.1	1.29	0.19	13.2
5	1.37	0.26	8.9	1.43	0.46	11.2	1.50	0.16	11.2
10	1.41	0.82	8.5	1.48	0.49	9.9	1.42	0.46	12.2

Fig. 7. Typical hole detection results: green edges are those in the planarized network; black cycles are detected boundaries. (a) UDG network with $N = 2000$ nodes and average degree 8.14. (b) UDG network with $N = 1500$ nodes and average degree 12.72. (c) quasi-UDG network with $N = 2500$ nodes and average degree 8.05, where the transmission ranges vary by as much as $R/r = 10$ times. (d) quasi-UDG network with $N = 1500$ nodes and average degree 12.05, where the transmission ranges vary by as much as $R/r = 5$ times.

[6] V. DUJMOVIC, M. FELLOWS, M. HALLETT, M. KITCHING, G. LI-OTTA, C. MCCARTIN, N. NISHIMURA, P. RAGDE, F. ROSAMOND, AND M. SUDERMAN, A fixed-parameter approach to two-layer planarization, *Proc. 9th International Symposium on Graph Drawing*, pp. 1-15, 2001.

[7] P. EADES AND S. WHITESIDES, Drawing graphs in two layers, *Theoretical Computer Science 131*, pp. 361-374, 1994.

[8] S. FUNKE AND C. KLEIN, Hole detection or: "How much geometry hides in connectivity?", in *Proc. SCG*, pp. 377-385, 2006.

[9] S. FUNKE AND N. MILOSAVLJEVIC, Network sketching or: "How much geometry hides in connectivity?-Part II," in *Proc. SODA*, 2007.

[10] D. GANESAN, B. KRISHNAMACHARI, A. WOO, D. CULLER, D. ES-TRIN, AND S. WICKER, Complex behavior at scale: An experimental study of low-power wireless sensor networks, Technical Report UCLA/CSD-TR 02-0013, UCLA, 2002.

[11] B. KARP AND H. KUNG, GPSR: Greedy perimeter stateless routing for wireless networks, in *Proc. MobiCom*, pp. 243-254, 2000.

[12] Y.-J. KIM, R. GOVINDAN, B. KARP AND S. SHENKER, Geographic routing made practical, in *Proc. NSDI*, 2005.

[13] E. KRANAKIS, H. SINGH, AND J. URRUTIA, Compass Routing on Geometric Networks , in *Proc. 11 th Canadian Conference on Computational Geometry*, 1999.

[14] F. KUHN, R. WATTENHOFER, AND A. ZOLLINGER, Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing, in *Proc. MobiHoc*, pp. 267-278, 2003

[15] D. MOORE, J. LEONARD, D. RUS, S. TELLER, Robust Distributed Network Localization with Noisy Range Measurements, *Proc. 2nd ACM SenSys*, pp. 50-61, 2004.

[16] S. NAKANO, Planar drawings of plane graphs, in *IEICE Trans. Information and Systems*, vol. E00, no. 3, 2000.

[17] S. RATNASAMY, L. YIN, F. YU, D. ESTRIN, R. GOVINDAN, B. KARP, S. SHENKER, GHT: A geographic hash table for data-centric storage, *Proc. WSNA*, pp. 78-87, 2002.

[18] Y. WANG, J. GAO, AND J. S.B. MITCHELL, Boundary recognition in sensor networks by topological methods, *Proc. MobiCom*, pp. 122-133, 2006.