Fine-Grained Boundary Recognition in Wireless Ad Hoc and Sensor Networks By Topological Methods

Dezun Dong [†‡] Yunhao Liu[‡] Xiangke Liao[†]

[†] School of Computer, National University of Defense Technology, Changsha, Hunan, China

[‡] Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology

ABSTRACT

Location-free boundary recognition is crucial and critical for many fundamental network functionalities in wireless ad hoc and sensor networks. Previous designs, often coarse-grained, fail to accurately locate boundaries, especially when small holes exist. To address this issue, we propose a fine-grained boundary recognition approach using connectivity information only. This algorithm accurately discovers inner and outer boundary cycles without using location information. To the best of our knowledge, this is the first design being able to determinately locate all hole boundaries no matter how small the holes are. Also, this distributed algorithm does not rely on high node density. We formally prove the correctness of our design, and evaluate its effectiveness through extensive simulations.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Network Architecture and Design—*Wireless communication*; C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks — *Network Protocols*; G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph Theory*

General Terms

Algorithms, Theory

Keywords

Fine-Grained Boundary Recognition, FGP Transformation, Wireless Ad Hoc and Sensor Networks

1. INTRODUCTION

In many practical deployment of wireless ad hoc and sensor networks, there usually exist regions where there are no nodes, or node density is much lower than other regions. The regions without enough active nodes form 'holes' of the network functions. In other words, a connected network may have many boundaries, outer and inner. Detecting and locating those boundaries have a great relevance to the design of basic networking services, such as point-to-point routing [5] [19], data gathering mechanisms [21] and sensing coverage verification [11] [1] etc. Certainly, precise location information will greatly help the boundary detection and make it simple. Unfortunately, computing exact coordinates requires a significant subset of nodes being equipped with special hardware like GPS and ranging devices. It is practically difficult for large scale networks. Therefore, location-free boundary detection has received considerable attentions in recent years. A number of approaches are proposed, providing reasonable approximations of spatial boundary information [9, 13, 18, 21].

Existing works, offering a visible way to extract *global* topological characteristics of the network without node coordinates, however, mostly detect boundaries coarsely in terms of their assumptions and quality of found boundaries. For example, designs in [7] [6] assume nodes are uniformly distributed with a very high density. Methods in [8] [9] fail to export continuous meaningful boundary cycles. The approach in [21] is good at detecting global 'big' (network scale size) hole in a large network, while is poor to find small holes especially when there exist a lot of small holes in the network. In general, they are not able to answer the question like how many holes above k-size in the network. Some fine-grained methods, by claiming that they are able to detect small holes [5] [11] [4], either heavily rely on accurate location information [5] or cannot locate holes properly [11] [4].

It worth pointing out that there exist great necessities and requirements on fine-grained boundary recognition. First, a network practically contains many relative small holes besides a few big ones. The holes are likely to be formed due to either the irregularity of random deployment or node failures, such as malfunctioning, battery depletion or external events such as fire or structure collapse. Detecting small holes benefits basic network functions. For example, it helps for routing selection since small holes also affect local strategy of routing. Second, algorithms of finding small holes in networks can also be used to identify coverage holes or regions of interests to users. Consider a scenario of event detection. If event value in small regions exceeds a critical threshold, nodes in the regions may become unavailable or being marked as abnormal by themselves. Those small event regions need to be clearly outlined by the boundaries enclosing them.

Major contributions of this work are as follows. We first present a formal definition of topological boundaries, and then propose a fine-grained boundary recognition method, especially locating small holes in the network. We design a graph tool, called FGP transformation, which reduces a graph while preserving its connectedness. To our best knowledge, this is the first location-free design that focuses on exactly finding boundary cycles surrounding small holes in wireless ad hoc and sensor networks.

The remainder of this paper is organized as follows. We discuss related work in Section 2, and introduce the problem formulation in Section 3. Section 4 presents the details of our boundary discovery algorithms. We prove the correctness of our algorithm and validate its effectiveness in Section 5. Section 6 concludes the work.

2. RELATED WORK

In this section, we discuss the current works of boundary recognition in wireless ad hoc and sensor networks. We classify them into two categories according to their quality on detected boundaries.

2.1 Coarse-Grained Methods

According to their respective main ideas, we further classify the coarse-grained methods into *local neighborhood* and *global topology*. The local neighborhood methods observe specific properties among nodes local neighborhood to differentiate whether a node locates at the interior or boundary of the network. Global topology methods explore geometric or topological impacts induced by boundaries in the entire network.

2.1.1 Local Neighborhood

Some works in this category observe that in a uniformly deployed network nodes in the interior of the network have much higher average degrees than nodes on the boundaries . Fekete et al. [7] [6] propose probabilistic methods to distinguish boundary nodes based on the statistical threshold on node connectivity degrees. They assume that networks are uniformly distributed with UDG (unit disk graph) model and a very high density, for example the average degree above 100. Other works exploit specific combinatorial structures among the local neighborhoods to distinguish boundaries. Based on the assumption that communication networks have a $\sqrt{2}/2$-quasi UDG ($\sqrt{2}/2$-QUDG) model, Kröller et al. [13] propose to estimate an interior node by searching specific patterns, called *flower*, among the connectivity graph. The boundary is further detected by augmenting cycles around interior nodes. The success of Kröller's design greatly depends on the identification of the flower structure, which may not always be the case, especially in a sparse networks [21]. Saukh et al. [18] recently extend the concept of patterns in UDG and $\sqrt{2}/2$-QUDG model to make it simple and tunable in sparse networks.

Local neighborhood methods share a common characteristic that they identify the boundary as a node set, the complement of the interior nodes. Those boundary nodes are assumed to form 'thick' strips of discrete nodes [7] [18]. When applied, however, the application still needs to separate the boundary nodes into boundary strips to explicitly obtain the location of each boundary. Such separations require two preconditions. First, thick boundary strips about different holes are not too adjacent to be combined; second, the size of those holes are sufficiently large. Otherwise, if the thick boundary strips overlap, it is difficult, if not impossible, to find the refined independent and continuous boundary merely using connectivity information. Hence, those methods implicitly assume holes are relatively far away in the network region. Indeed, the authors of [7] assume the network regions with holes have a lower bound on the *fatness*. On the other hand, if the hole is too small, the boundaries might be hidden among the thick boundary strips and cannot be distinguished. Moreover, the patterns in combinatorial structures methods are glued by small chordless cycles. They inherently fail to differentiate a common chordless cycle with a short cycle encircling a small hole. As a result, holes of small size are inevitably ignored.

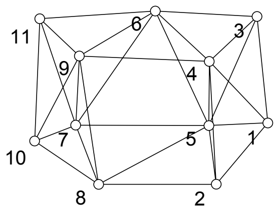
2.1.2 Global Topology

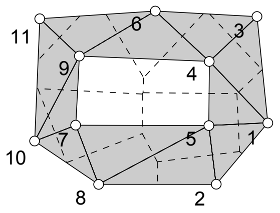
The approaches in this category observe the global geometric or topological impacts of the hole boundary in the continuous domain. They transform the impact to the discrete network, assuming that discrete nodes well represent the underlying geometric environment, and shortest hop distance between nodes provides a reasonable approximation for their geometric distance. For example, Funke [8] [9] locates the hole boundaries by identifying the characteristics of broken isolines. Wang and Gao et al. [21] cut networks politely and combine some extracted shortest paths into cycles which are homotopic to hole boundaries. Funke's approaches require a high average node degree (around 20) to detect the isoline breakage reasonably well. Moreover, the boundaries found are scattered nodes and we do not really get a continuous boundary. Comparatively, Wang and Gao's design [21] requires a much lower node density (above 10) and exports continuous boundary cycles.

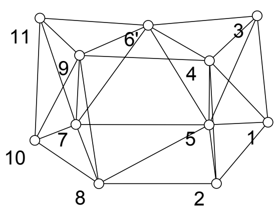
Above mentioned approaches, however, still locate holes in a coarse-grained manner. In [8] [9], the isoline brokenness can be detected only when isolines encounter big holes due to the thickness of isolines. The disturbed small holes are regarded as noises and do not be tracked. The success of [21] depends upon its multiple operation components, such as finding cut pairs, contracting a candidate cycles, and etc. Those components are designed based on the observations in the continuous domain. Given a basepoint in a planar polygonal region with simple polygonal voids inside, the set of points that have at least two geodesic shortest paths to the basepoint forms the cut locus. The cut locus is composed of cut branches. Their algorithm exploits the nice properties of cut locus. The tow different (non-homotopic) shortest paths to a cut branch can be concatenated into a non-contractible cycle. Their algorithm finds boundaries by refining the non-contractible cycles. When the region just contains one hole, the cut locus is very simple and only includes one cut branch. They identify the connected cut pair nodes as a cut branch and recognize the hole well. When there are multiple holes and multiple cuts in the network, their algorithm needs artificially merge the holes by removing nodes on cut branches. This is feasible for well-separated and well-placed big holes. When multiple small holes are close to each other, the cut branches will be combined and connected together. See Section 5.3 for such examples. It is difficult to topologically distinguish the structure of such discrete cut locus, a set of scattered nodes, and split it into discrete cut branches with only connectivity information. Indeed, when there are multiple holes inside a continuous region, the structure of its cut locus generally is very complicated, which can include multiple connected components and each connected component can include many cut branches. Moreover, the process of refining a coarse boundary in their work, by connecting some specific extremal nodes, also inevitably ignores small holes.

2.2 Fine-Grained Methods

Less work can find boundaries in fine-grained manner. As a pioneer work, Fang and Gao et al. [5] propose to accurately locate communication holes using location information through computational geometry techniques. Ghrist et al. [11] [4] further propose to detect holes of insufficient sensing coverage without location information by computational topological methods. They model the sensing range as unit disk and attach Rips complex on the communication graph. They detect coverage holes by using the fact that the first homology group of Rips complex provides sufficient infor-


(a) Embedding ε_1


(b) RDG in ε_1


(c) Embedding ε_2

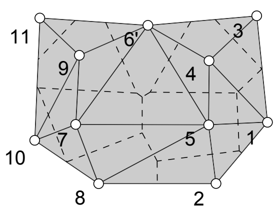

(d) RDG in ε_2

Figure 1: Non-consistency of RDG-based hole definition.

mation about coverage. The advantage of their methods is that it is able to detect the existence of holes of no matter what size without location information. The limitation is that the method cannot accurately localize coverage holes, that is, attach a cycle to each hole which is encircled properly by the cycle. The main reasons are as follows. For more than one hole in the network, the homology group without endowed with geometric information can be generated equivalently with many possible generators combinations so that one hole can be enveloped by more than one generators or one generator surrounds multiple holes. Hence, the approach fails to really locate each hole. Also, the method is centralized, which is often treated impractical for large scale ad hoc and sensor networks. We will focus on locating communication boundaries in a fine-grained manner without using location information.

3. PROBLEM FORMULATION

In this section, we present network assumptions and formally define the network boundaries. We consider a collection of nodes deployed over a plane region. The nodes are only capable of communicating with a finite number of other nodes in its proximity. The coordinates of nodes are assumed to be unavailable, in the sense that nodes can determine neither distance nor orientation. Thus, we detect boundaries in a connectivity graph G, where vertices and edges denote the nodes and communication links, respectively. Without loss of generality, we assume G is connected.

Indeed, providing a formal definition of boundary is far from trivial. We do not see any unified and formal definition about network boundary in existing literatures. The authors in [8] and [21] implicitly describe network boundaries as nodes being close to the boundary of underlying continuous domain where nodes are deployed. Other works [5] [13] [18] explicitly present different definitions about boundary. Those boundary definitions are imperfect, especially when we desire hole boundaries having two properties: *continuity* and *consistency*. Continuity means that all the nodes in a boundary can be connected by themselves, and can form loop-like connection instead of being isolated. The demand on continuity is instinctive for network boundaries, which serves as discrete counterparts of continuous boundaries. Consistency means that a boundary enveloping holes in one embedding still encircles holes in other embeddings. In other words, we desire the defined boundary being independent with the specific embedding.

3.1 Existing Boundary Definitions

Before presenting our definition, let us look at the previous boundary (or hole) definitions [5] [13] [18]. All those definitions can be explained based on the embedding of connectivity graph under UDG or QUDG model. A UDG embedding for a graph is to find a straight-line drawing of the graph in the Euclidean plane such that there is an edge between two vertices if and only if distance between the two vertices is at most 1. For conciseness, the embedding we mentioned in the rest of this paper refers to UDG embedding by default.

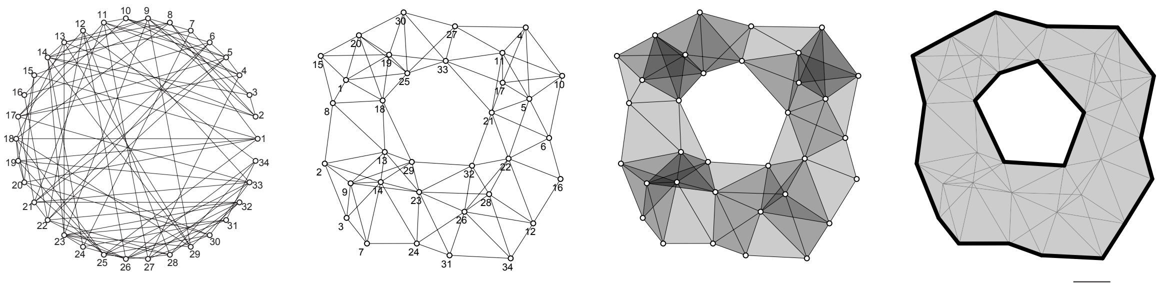
Authors in [5] present a hole definition using computational geometry approaches. They model sensor networks as UDGs and assume location information available, equivalently a valid embedding is given. They define network holes by faces with at least 4 vertices in the *Restricted Delaunay Graph* (RDG) [10] [14], a planar subgraph computed from a UDG graph. Such a definition is of continuity, but not of consistency. Consider the example in Figure 1. An embedded network is shown in Figure 1 (a), and its RDG is in Figure 1 (b) where lines denote edges of RDG and dot lines denote borders of voronoi regions. Clearly, the network has a hole under their definition. We then consider another embedding of the same network in Figure 1 (c) where node 6 in Figure 1 (a) is moved a little to $6'$. There is no hole in the network, as shown by the RDG graph in Figure 1 (d). Hence, such a definition is susceptible to embedding and not of consistency.

An embedding divides the plane into many polygon regions. Authors in [13] and [18] define boundaries based on the polygon regions. Those polygon regions include an infinite face and many finite faces. The basic idea is to view each finite face as a hole and a perimeter of a finite face as the hole boundary, and associate outer boundary with infinite face. Since a vertex on a finite face does not often correspond to a node in the network, authors of [13] define hole boundary as chordless cycles in network graph that surround a finite face. Such boundary is of continuity. The authors in [18] further define the network hole as the set of nodes on the hole perimeter, which make boundary no more continuous. However, neither definition in [13] or [18] is of consistency, because those finite faces change with embedding, leading to the fact that perimeter nodes are susceptible to embedding.

3.2 Topologically Defining Boundaries

We now present our mathematical definition of network boundaries (or holes) in a topological fashion. We associate a topological space Δ_G with connectivity graph G. Visually, Δ_G is constructed by filling all the triangles in G with a triangle face. More formally, Δ_G is a 2-simplicial complex whose 0,1,2-simplices one-to-one correspond to vertices, edges and triangles of G, respectively. We then consider the embedding ε of G, which maps the vertices of G to the plane $\mathbb{R}^2$. We obtain the embedding graph G^ε of G. Accordingly, a geometric realization Δ_G^ε of Δ_G is also obtained. See Section 5.1 for basic preliminaries for topology concepts, including *simplicial complex* and *homotopy* and *fundamental group* etc.

Let us look at an example in Figure 2. Connectivity graph G is shown in Figure 2 (a), and its a possible valid embedding G^ε shown in Figure 2 (b). Meanwhile, Figure 2 (c) exhibits the geometric realization Δ_G^ε, G^ε integrated with gray triangles. The region in the plane overlaid by Δ_G^ε forms its *shadow* [2], denoted by $\overline{\Delta_G^\varepsilon}$ with overline standing for shadow mapping, as shown in Figure 2 (d). As a planar geometrical shape, void regions of $\overline{\Delta_G^\varepsilon}$ exactly define its holes. The inner and outer boundaries of $\overline{\Delta_G^\varepsilon}$ can be clearly



(a) Connectivity graph G (b) Embedding ε (c) Complex Δ_G^ε (d) Shadow $\overline{\Delta_G^\varepsilon}$

Figure 2: Topological boundary definition.

formulated as closed polygonal chains, denoted by bold lines in Figure 2 (d). Based on the geometric boundary of $\overline{\Delta_G^\varepsilon}$, we then present the topological boundary of G as following.

DEFINITION 1. *(**Topological Boundaries**) Given a cycle C in connectivity graph G and an embedding ε of G, if $\overline{\Delta_C^\varepsilon}$ can be continuously deformed into (homotopic to) a boundary of $\overline{\Delta_G^\varepsilon}$, then C is a topological boundary of G.*

Note that if $\overline{\Delta_G^\varepsilon}$ does not include any inner hole, its outer boundary indeed trivially be equivalent to a point in it. The Figure 3 illustrates the idea about our definition. In the Section 5.2, we will proof that the topological boundaries do meet the consistency in UDG model, though here Definition 1 is presented in a given embedding.

$$\begin{tikzcd}
G \arrow[r, Leftrightarrow] \arrow[d, "\text{topological boundary}"'] & \Delta_G \arrow[r, "\text{embedding}"] \arrow[d, "\text{subspace}"] & \Delta_G^\varepsilon \arrow[r, "\text{shadow}"] \arrow[d, "\text{subspace}"] & \overline{\Delta_G^\varepsilon} \arrow[r, "\text{geometric}"] \arrow[d, "\text{subregion}"] & \text{Polygonal boundary} \\
C \arrow[r, Leftrightarrow] & \Delta_C \arrow[r, "\text{embedding}"] & \Delta_C^\varepsilon \arrow[r, "\text{shadow}"] & \overline{\Delta_C^\varepsilon} \arrow[ur, "\text{homotopic to}"] &
\end{tikzcd}$$

Figure 3: Idea of boundary definition.

4. BOUNDARY RECOGNITION ALGORITHM

In the previous section, we set up a one-to-one correspondence between connectivity graph G with its topological counterpart Δ_G. In this way, we can attach topological concepts to G, and use the methods of algebraic topology in our boundary recognition algorithm. The topological boundaries of G indeed correspond to homology generators [11] of Δ_G. Our algorithm is to seek specific homology generators in G and refine them to proper boundaries. We design graph tools to develop the algorithm.

This algorithm includes four components: *skeleton extraction*, *primary boundary cycles* and *refined inner boundary cycles* and *refined outer boundary cycle*. For easy to understand, we first present the centralized scheme, and describe its distributed version later. Figure 4 illustrates the procedures of this design. Given a network with multiple holes, such as the one shown in Figure 4 (a), our algorithm aims to recognize all its inner and outer boundaries. In skeleton extraction component shown in Figure 4 (a-f), we reduce the connectivity graph of the original network, and extract its skeleton graph, which features the original network faithfully as shown in Figure 4 (f). The skeleton graph is then separated into primary boundary cycles. See Figure 4 (g). Each primary boundary cycle indicates an inner hole or outer boundary, but it is still too coarse to exactly locate the holes or outer boundary. We further refine the primary boundary cycles into tightest inner boundaries, see Figure 4 (h-k), and proper outer boundary, see Figure 4 (l-o). Finally, the algorithm discovers all the boundaries shown in Figure 4 (p).

4.1 Simple-Connectedness Graph and FGP Transformation

Before presenting the details of algorithm, we first introduce the simple-connectedness graph and FGP (*Fundamental Group Perservering*) transformation, which are the tools we designed for manipulating graphs.

Let H be a simple graph with vertex set $V(H)$ and edge set $E(H)$. A *cycle* C is a subgraph of H if it is connected and each vertex in C has degree two. A cycle C can be identified by its incidence vector $< b_i(C) >_{i\in[1,|E(H)|]}$, with $b_i(C) = 1$ iff $e_i \in E(C)$ and $b_i(C) = 0$ iff $e_i \notin E(C)$. The length of $|C|$ is the number of its edges, $|E(C)|$. The incidence vectors of cycles span a binary vector space $\mathcal{C}(H)$, called the *cycle space* of H. The addition of two cycles C_1 and C_2 is defined as the binary addition of their incidence vectors. It corresponds to the symmetric difference $C_1 \oplus C_2 = (E(C_1) \cup E(C_2)) \setminus E(C_1) \cap E(C_2)$. A *cycle basis* $\mathcal{B}$ of H is a basis of $\mathcal{C}(H)$. The length $\ell(\mathcal{B})$ of $\mathcal{B}$ is the total length of its cycles: $\ell(\mathcal{B}) = \sum_{C\in\mathcal{B}} |C|$. We further define *triangle cycle subspace* $\mathcal{C}_\mathcal{T}(H)$ of H as the set of all 3-length cycles in $\mathcal{C}(H)$, $\mathcal{C}_\mathcal{T}(H) \subseteq \mathcal{C}(H)$.

DEFINITION 2. (***Simple-Connectedness Graph***) *A connected graph H is of* simple connectedness *if its cycle space $\mathcal{C}(H)$ is empty, or for any cycle C in $\mathcal{C}(H)$, there exists a set of 3-length cycles $\mathcal{T}_0 \subseteq \mathcal{C}_\mathcal{T}(H)$ such that $C = \sum_{T\in\mathcal{T}_0} T$.*

From Definition 2, apparently a tree is of simple connectedness. Intuitively, a hole boundary cycle in a connectivity graph cannot be filled by triangles, thus cannot be represented by the linear combination of triangles. Hence, a connectivity graph with holes is not of simple connectedness. Note that in common graph theory terms a simple and connected graph is opposed to a multigraph and disconnected graph. The simple connectedness discussed in this paper needs to be comprehended and rephrased as *simple connectivity* in topological terms.

We then define FGP transformation on graphs. Let X be a vertex (or edge) set in a graph H, we use $H[X]$ to denote the vertex-induced (or edge-induced) subgraph by X. Given vertex set $Y \subseteq V(H)$, we write $H - Y$ for $H[V(H) \setminus Y]$. Given edge set Z with its all endpoints V_Z, we make $H - Z = (V(H) \setminus (V_Z \setminus V(H)), E(H) \setminus Z)$ and $H + Z = (V(H) \cup V_Z, E(H) \cup Z)$. Let x be a singleton, a vertex or edge, $H - \{x\}$ (or $H + \{x\}$) is abbreviated to $H-x$ (or $H+x$). The neighbors of a vertex v in H is denoted by $N_H(v)$. The *neighboring graph* $\Gamma_H(v)$ of vertex v in H is defined as $H[N_H(v)]$. The *neighboring graph* $\Gamma_H(e)$ of an edge $e = (u, v)$ in H is assigned to $H[N_H(u) \cap N_H(v) \cup \{u, v\}] - e$.

DEFINITION 3. (***FGP Transformation***) *A FGP transformation is a sequential combination of graph operators, including vertex (or edge) insertion or deletion operator.*

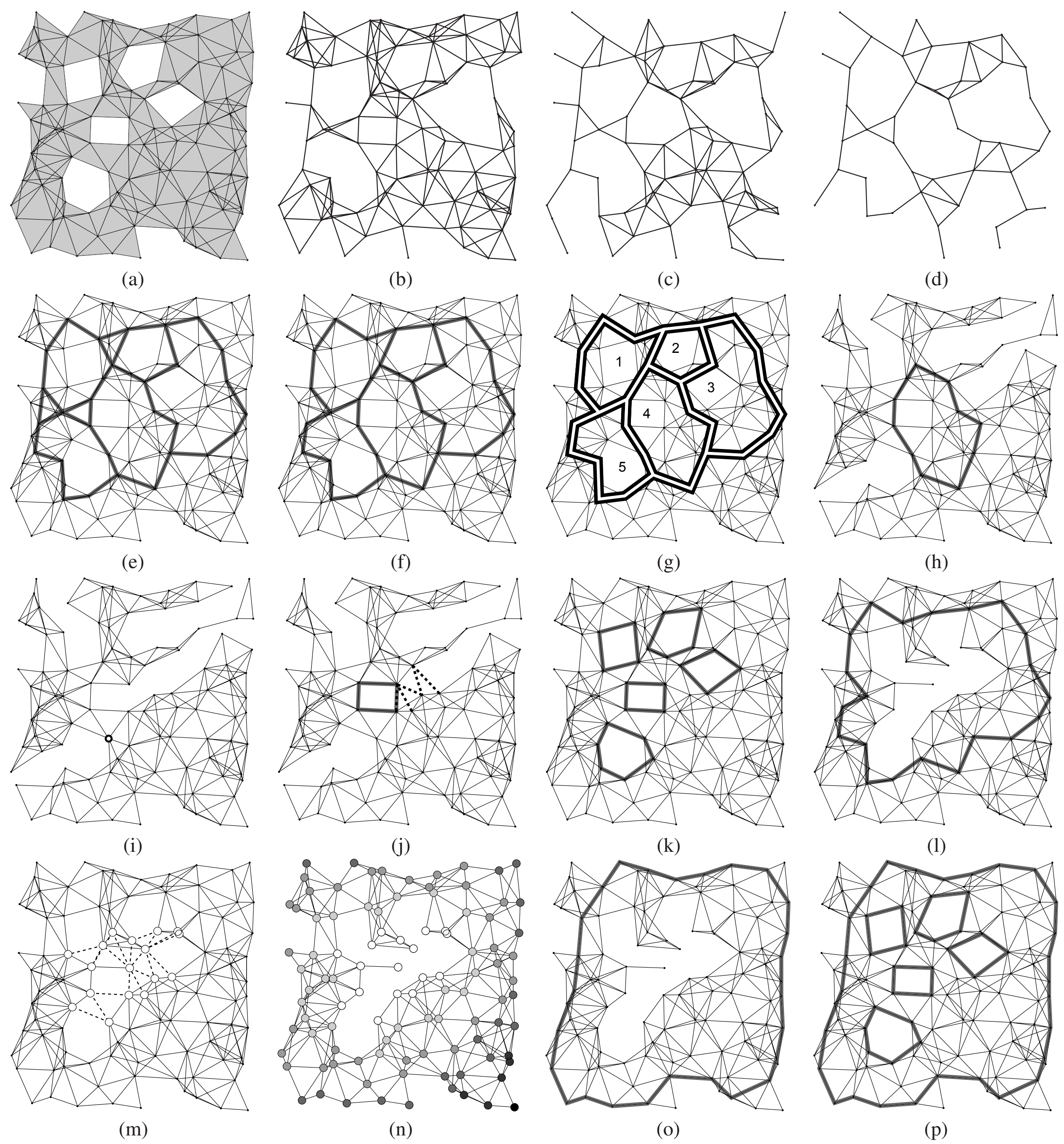


Figure 4: Procedures of our boundary recognition algorithm. (a) Original network G with five holes, 100 nodes and average degree 6.64; (b-d) Snapshot of deleting 15, 35 and 55 vertices from G using FGP transformation; (e) Obtain G_{vd} after maximal vertex deletion from G; (f) Obtain skeleton graph G_S after further maximal edge deletion from G_{vd}; (g) Separate G_S into primary boundary cycles, cycle basis $\mathcal{B}_{G_S}$ and infinite facial cycle C_{inf}; (h) Maximally extend a primary boundary cycle C in $\mathcal{B}_{G_S}$ to obtain graph G_C; (i) Maximally extend graph one vertex v in G_C to obtain G_v; (j) Obtain gap edges and find the tightest cycle from G_C and G_v; (k) Discover all the refined inner boundary cycles; (l) Obtain maximally extended graph G_{inf} from infinite facial cycle C_{inf}; (m) Critical vertices $V_{critical}$ in G_{inf}; (n) Shortest distance of each vertex to critical vertices; (o) Refined outer boundary cycle after prioritized reduction on G_{inf}; (p) Output all the boundary cycles.

• *Insertion operator. A vertex (or edge) x not in H can be inserted to H to construct a new graph H' if (1) neighboring graph $\Gamma_{H'}(x)$ is connected, and (2) existing a simple-connectedness subgraph $H'' \subseteq H$ such that $\Gamma_{H'}(x) \subseteq H''$.*

• *Deletion operator. A vertex (or edge) x of H can be deleted if (1) neighboring graph $\Gamma_H(x)$ is connected, and (2) existing a simple-connectedness subgraph $H' \subseteq H - x$ such that $\Gamma_H(x) \subseteq H'$.*

Figure 5 shows an example for simple-connectedness graph and FGP transformation. The left four graphs are all of simple connectedness while the quadrangle is not. The graphs *a*-*d* can be mutually transformed by executing operators of FGP transforma-

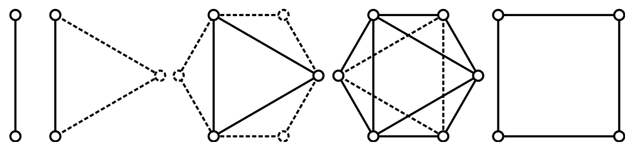


Figure 5: An example for FGP transformation.

tion. For example, graph b can be obtained by adding a vertex to graph a or by deleting three vertices from graph d, and graph c can transformed into graph d by gluing three edges to it. The left four graphs, however, cannot be transformed into the quadrangle through FGP transformation due to their differences in connectedness. Later we will show that a graph remains its simple connectedness under FGP transformation in Theorem 4 in Section 5.

4.2 Skeleton Extraction

This component conducts maximal vertex deletion and edge deletion on original connectivity graph G to extract its skeleton graph G_S.

Step 1: Vertex deletion

In this step, we maximally apply vertex deletion operator of FGP transformation on G to reduce it to graph G_{vd}. Figures 4 (b), (c) and (d) show three snapshots of the process. Due to the correlation dependence between vertices in the local, some vertices must be deleted after other vertices. Most likely, vertices close to the boundary in the network are deleted earlier, and deletion operations gradually spread towards inner vertices. Note that we only require the connectivity among the local neighbors of a vertex v in G to determine whether a vertex v in G can be deleted, which is shown in Section 5.2. This operation iteratively runs in rounds. In each round, a randomly selected vertex in the existing graph is deleted according by FGP transformation. The procedure of vertex deletion terminates until no vertex in G can be deleted, and then outputs graph G_{vd}. The bold lines in Figure 4 (e) shows the generated G_{vd} in the example. We can see that G_{vd} still contain triangles that do not envelop holes, so it needs farther removal.

Step 2: Edge deletion

In this step, we delete edges and eliminate the triangles in G_{vd}. In most cases, we can directly implement edge deletion operator of FGP transformation on G_{vd}. In some special cases, however, additional vertex insertion operations are necessary. More explanations are available in Section 4.6. Here we mainly consider the case that edge deletion is directly applicable. To confirm this, a feasibility test is needed on G_{vd} first to verify whether or not there are no two triangles in G_{vd} sharing a common edge. If none, edges can be deleted safely. Similar to vertex deletion, edge deletion operator is conducted iteratively until no more edge can be eliminated. We then obtain the skeleton graph G_S, as illustrated in Figure 4 (f), where we have following observations. Skeleton graph G_S characterizes the backbone of G faithfully, and is a triangle-free planar graph and comprises some cycles. There is a favorable correspondence between cycles in G_S and holes of G. In Section 5.2, we will formally prove that skeleton graph G_S is a planar graph and equivalent to G in the sense of topological connectedness. See Theorem 6.

4.3 Primary Boundary Cycles

In this component, we split the skeleton graph G_S into a set of primary boundary cycles $\mathcal{P}$ such that each cycle of $\mathcal{P}$ either exactly surround one hole of G or corresponds to the outer boundary, as shown Figure 4 (g). It is not trivial to achieve this without location information. Our method comes from the observation that these cycles $\mathcal{P}$ actually forms a cycle basis of G_S. Thus, calculating the cycle basis seems benefit the splitting. Unfortunately, cycle basis of a graph is not unique, and usually has a considerable amount of possible cycle combinations. We have to explore more constraints. One further observation is that one edge in G_S should be contained in at most two cycles of $\mathcal{P}$. This leads us to find the *2-basis* or *planar basis* [15] in the cycle space $\mathcal{C}(G_S)$. G_S do have 2-basis due to its planarity [15]. A 2-basis of a planar graph consists of all facial cycles, except one. The missing facial cycle can be regarded as the one corresponding to the infinite face. Hence, if G_S has unique planar embedding and the infinite face in the embedding is known, we are able to acquire a unique 2-basis of G_S. Consequently, we can split the skeleton graph determinately. We explain the splitting procedure through the example shown in Figure 4. We can see that the found skeleton graph in Figure 4 (f) has unique planar embedding. We need to determine the right infinite face. Intuitively, the infinite facial cycle corresponds to the outer boundary. Nevertheless, it is well known that any face of a planar embedding can be transformed into infinite face without any geometric constraint. Hence, we need to utilize a little heuristic information in G_S to differentiate the infinite face from finite faces. Since the infinite facial cycle corresponds to the outer boundary, the differentiation becomes straightforward if the outer boundary of the network is explicitly known. In this work, instead of assuming the awareness about outer boundary, we explore the observation that the outer boundary cycle contains all the inner hole cycle and is usually longer than inner hole boundary. Hence, the infinite facial cycle is longer than other inner facial cycles in terms of hop number. As a result, we can formalize the problem of extracting primary boundary cycles as finding the minimum 2-basis.

DEFINITION 4. (***Minimum 2-Basis***) *A* minimum 2-basis *of graph H is a cycle basis $\mathcal{B} \subseteq \mathcal{C}(H)$ such that (1) any one edge in H appears in at most two cycles in $\mathcal{B}$, (2) the total length of $\mathcal{B}$, $\ell(\mathcal{B})$, is minimized.*

We now discuss how to calculate the planar embedding of skeleton graph G_S. All possible planar embeddings of G_S can be computed in $O(|V(G_S)|)$ time [3]. The planar embedding of a planar graph is unique if the graph is tri-connected [22]. If many holes exist in a graph, its skeleton graph tends to be of tri-connected. If it is unfortunately not unique, we can also calculate all its the possible embeddings.

After successfully calculating the minimum 2-basis $\mathcal{B}_{G_S}$ of G_S, correspondingly, we also obtain the infinite facial cycle C_{inf}, and the primary boundary cycles $\mathcal{P} = \mathcal{B}_{G_S} \cup \{C_{inf}\}$. Each primary boundary cycle corresponds to a boundary. The infinite facial cycle is with outer boundary, and other cycles surround inner holes. Primary boundary cycles are still loose or rough yet. We desire to refine these cycles to obtain the ultimate inner and outer boundary cycles. Primary boundary cycles form the generators (a basis) of cycle space of G_S. As pointed earlier, skeleton graph G_S is topologically equivalent to original connectivity graph G. Hence, primary boundary cycles indeed correspond to generators of the fundamental group of Δ_G. Each primary boundary cycle actually represents all its cycles in the same (homotopy) equivalence class. In the next steps, our goal is to seek proper refined boundary cycles in those equivalent cycles.

4.4 Refined Inner Boundary Cycles

This component focuses on finding the refined inner boundary cycles, which accurately locate the holes. We utilize minimum 2-basis as the initial inner boundary cycles and desire to find the tightest cycle surrounding each hole. One instinctive way is to contract each cycle locally. Clearly, if two edges of a cycle belong to a triangle, the two edges can be substituted with the third edge and the length of the cycle is reduced by one. One can use this way to collapse the cycle till the cycle cannot be reduced any more. Such method, however, may probe to local minima and stop before arriving at the shortest cycle. We propose a more tricky and general method.

Given a primary boundary cycle C, we extend C and obtain a maximal graph G_C which topologically equivalent to C, as illustrated in Figure 4 (h). We arbitrarily select one vertex v in G_C, and extend v in G_C to obtain a subgraph G_v of G_C, shown in Figure 4 (i). We thus acquire the *gap* edge set $E_{gap} = E(G_C) \setminus E(G_v)$, shown as dot lines in Figure 4 (j). E_{gap} holds an interesting property that any cycle in G_C surrounding the hole inevitably passes through at least one edge in E_{gap}. Then, for each gap edge, we calculate the shortest cycle passing it and obtain a candidate cycle. We traverse all the gap edges and select the shortest candidate cycle as the final tightest inner boundary cycle, denoted by the bold lines in Figure 4 (j). Through the above steps, all the refined inner boundary cycles are shown in Figure 4 (k). We then present the details in each step.

Step 1: Maximally extending one cycle

This step has three phases: initializing, maximal vertex adding and maximal edge adding.

In initializing phase, we construct the initial extended graph G_C from cycle C. We first set G_C be equal to cycle graph C, and try to transform G_C into a vertex induced subgraph of G. Specifically, let $E_0 = E(G[V(G_C)]) \setminus E(G_C)$. If E_0 is not empty, we check whether one edge in E_0 can be added into G_C by performing edge insertion operator of FGP transformation. If so, we glue this edge onto G_C.

The second phase is vertex-oriented graph extending. For a vertex v in $V(G) \setminus V(G_C)$, we say that v is *adjacent* to G_C if the neighbors of v in G_C is not empty, that is, $N_G(v) \cap V(G_C) \neq \emptyset$. We abbreviate $N_G(v) \cap V(G_C)$ to $N_{G_C}(v)$. We randomly select a vertex u adjacent to G_C from $V(G) \setminus V(G_C)$ to test whether u can be inserted into G_C under FGP transformation. We test whether G_C can be extended to be $G' = G_C \cup G[N_{G_C}(u) \cup \{u\}]$ through inserting vertex u and related edges. If so, u is extended and update $G_C = G'$. The above operations are carried out iteratively until no vertex in $V(G) \setminus V(G_C)$ that can be extended.

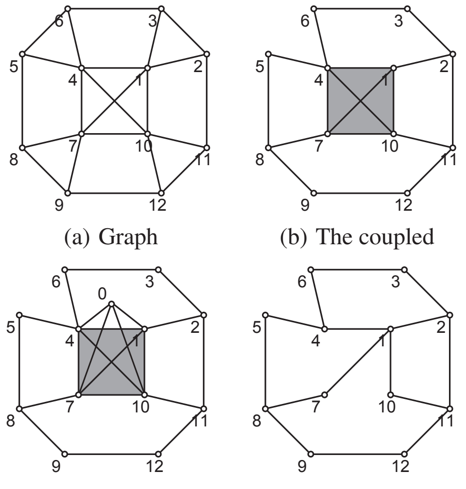
Third phase is edge-oriented graph extending. We consider the subgraph $G_l = G[E_l]$ induced by the left edge set $E_l = E(G) \setminus E(G_C)$, and continue to add the edges and vertices in G_l into G_C. For each edge $e = (v_{e,1}, v_{e,2})$ in G_l, if $v_{e,1}$ and $v_{e,2}$ both are in G_C, we test whether e can be added into G_C under FGP transformation, if yes, set $G_C = G_C + e$ and $G_l = G_l - e$. If only one endpoint of e, say $v_{e,1}$, is in G_C, we make $G_C = G[E(G_C) \cup \{e\}]$, and $G_l = G_l - e$. The above operations are conducted iteratively until there are no more edges in G_l can be extended, and export graph G_C. Figure 4 (h) shows the extended graph from the 4th primary boundary cycle.

Step 2: Maximally extending one vertex

We randomly select a vertex v in G_C, and maximally extend v to obtain a subgraph G_v of G_C, as shown in Figure 4 (i). The extending procedures is similar to step 1 for one cycle, including maximally adding vertices and edges, so we skip the details.

Step 3: Seeking the tightest cycle

Comparing G_v with G_C, we obtain the gap edge set $E_{gap} = E(G_C) \setminus E(G_v)$, denoted by the dot lines in Figure 4 (j). We can prove that any cycle in G_C surrounding the same hole with c must contain at least one edge in E_{gap}. For each edge $e = (v_{e,1}, v_{e,2})$ in E_{gap}, we calculate the shortest path P_e between $v_{e,1}$ and $v_{e,2}$ in G_v. By connecting P_e with edge e, we obtain one cycle, which is the shortest cycle in G_C passing e and surrounding the hole. Through comparing all the shortest cycles passing gap edges, the final tightest inner boundary cycle is obtained. See the bold-line cycle in Figure 4 (j).



(a) Graph (b) The coupled
(c) Substitution (d) Restoration

Figure 6: Handling coupled triangles.

4.5 Refined Outer Boundary Cycle

In this component, we refine the infinite facial cycle C_{inf} among primary boundary cycles to acquire the outer boundary.

We first maximally extend C_{inf} under FGP transformation as foregoing manner of refined inner boundary cycles, and obtain the extended graph, denoted as G_{inf}, as shown in Figure 4 (l). We further obtain the patching edge set $E_{patch} = E(G) \setminus E(G_{inf})$, as the dot lines in Figure 4 (m). Let $G_{patch} = G[E_{patch}]$. We get a vertex set $V_{critical} = V(G_{inf}) \cap V(G_{patch})$, and name $V_{critical}$ *critical vertices* between G_{inf} and G_{patch}. In a given embedding ε of G, all the vertices and edges of G^{ε} locating outside the C^{ε}_{inf} tend to be extended. The edges of E_{patch}, which cannot be extended, mainly emerge in the inside of C^{ε}_{inf}. Hence, the critical vertices also tend to be inside of C^{ε}_{inf}. We utilize them as a hint to achieve refined the outer boundary. We calculate the shortest hop distance that a vertex in G_{inf} to those critical vertices, as shown in Figure 4 (n) where the darker dots denote those vertices with the larger shortest hop distance to critical vertices. We further reduce the G_{inf} similar with the above skeleton exaction component. The main difference with skeleton extraction lies in that the vertices and edges are deleted with considering priority. A vertex or edge having smaller distance to critical vertices is deleted much earlier. Finally, we obtain the extracted cycle from G_{inf} and use it as the outer boundary, shown in Figure 4 (o). Combined with the inner boundary found previously, we obtain all the boundaries of G, as shown in Figure 4 (p).

4.6 Handling Special Cases

In the above, we describe the main process of boundary recognition algorithm. In this subsection, we discuss and deal with one special case: coupled triangles. In this case, edges cannot be deleted directly in skeleton extraction.

We consider to extract the skeleton graph shown in Figure 6 (a). Clearly, there are no vertices in the graph in Figure 6 (a) can be

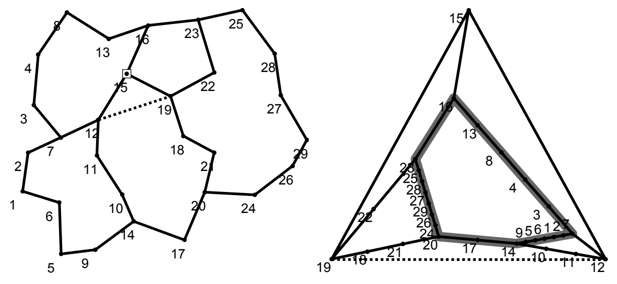

(a) Extended skeleton graph (b) Planar embedding

Figure 7: Distributed Planar Embedding.

deleted. Therefore, we consider to delete edges in it and check the feasibility test to confirm whether or not direct edge deletion is applicable. Unfortunately, the test cannot be passed since existing triangles in the graph that sharing edges, see the clique of vertex 1,4,7,10. Those edges that do not appear in the coupled triangles still can be deleted directly, as shown Figure 6 (b), whereas we require additional mechanisms for coupled triangles to guarantee the planarity of skeleton graph. One simple method tries to find multiple possible deletion and choose one output while retaining the planarity.

We next present a more flexible and reliable method. We first assemble all coupled triangles into components. We replace a simple-connectedness component with an artificial vertex and some edges. In this example, the coupled triangles form a component, denoted by grey square in Figure 6 (b). We replace it with the vertex 0 and four edges, which connect vertex 0 with all the vertices in the component, as shown in Figure 6 (c). Now all the edges in clique of 1,4,7,10 can be deleted and a planar skeleton graph is obtained. Further, cycle basis of (0,1,2,3,6,4), (0,4,5,8,7), (0,7,8,9,12,11,10), (0,10,11,2,1) can be extracted. After extracting the cycle basis, the path segments involved with the added vertex and edges can be replaced by vertices and edges in original component again. Such as for cycle (0,1,2,3,6,4), the added edges (4,0),(0,1) can be replaced with edge (4,1) and remodified cycle (1,2,3,6,4) is obtained which does not contain the artificial vertex 0 any more. Figure 6 (d) shows the recovered graph. Note that vertex and edge insertion operator involved in the substitution also follows the FGP transition. Hence, if the coupled triangles are not of simple connectedness, it needs first split into simple-connectedness components politely. After that, similar operations can be conducted for each split component.

4.7 Distributed Implementation

We show how we implement this design in a distributed manner. Due to the space limitation, the discussion is necessarily short. We will focus on addressing (or sketching) the principles of distributed conducting each component in our algorithm.

First, we describe the distributed scheme of skeleton extraction. The key issue is to distributedly execute vertex and edge deletion operators of FGP transformation. Deleting one vertex (or edge) from a graph only depends the connectivity among its k-hop neighbors (k is a small constant, See Theorem 9). Hence, it is easy to run distributed vertex deletion on G in rounds. In each round, each vertex v collects its k-hop neighboring graph $\Gamma_G^k(v)$, and estimates whether itself can be virtually removed according to FGP transformation. All the vertices can be deleted form a candidate vertex set V_{del}. Further, a k-hop maximal independent set ($kMIS$) V_{kMIS} can be selected from V_{del}. Hop distance between tow vertices in V_{kMIS} is at least k. Vertices in V_{kMIS} are deleted simultaneously and safely without causing concurrency conflict. Accordingly, G is updated to $G - V_{kMIS}$ and the next round starts up until V_{del} is empty. The distributed edge deletion can also be achieved similarly by inserting additional feasibility testing.

Second, we need to compute the minimum 2-basis of the skeleton graph to find primary boundary cycles. The key problem is find a planar embedding of skeleton graph in a distributed manner. Firstly, we identify *branch vertices* in skeleton graph. The branch vertices have degree greater than or equal to three. For example, the vertices 7,12,14,15,16,19,20,23 are branch vertices in Figure 7 (a). Then, we randomly select one branch vertex and its two neighbors in skeleton graph to generate a triangle. In the example of Figure 7 (a), vertex 15 is selected and the triangle (12,15,19) is created by adding a virtual edge to connect two neighbors, 12 and 19, of vertex 15. We call it as *landmark triangle*. We initially assign to landmark triangle vertices coordinates of an equilateral triangle, and other vertices in skeleton graph the center of the equilateral triangle. Afterward, we fix the landmark triangle vertices and iteratively place every vertex into the center of gravity of its neighbors in skeleton graph. Such a force-directed layout algorithm can be performed distributedly as done in [17]. When the process comes to an equilibrium state, as shown in Figure 7 (b), we can obtain a planar straight-line drawings of the skeleton graph with theoretical guarantee (Tutte embedding [20]). The planar embedding assigns each vertex in the skeleton graph a virtual coordinate. Further, vertices in skeleton graph can use virtual coordinates to split distributedly skeleton graph into facial cycles. Accordingly, the longest cycle, shown as the bold lines in Figure 7 (a), can be obtained.

Third, we deal with the distributed inner boundary refining. The first two steps, extending a primary boundary and a vertex, mainly implement the operations of vertex and edge insertion operator, which can be run distributively by proper adaptation on previous distributed vertex deletion. We mainly handle distributed implementation of seeking the tightest cycles. Specifically, we need to traverse the gap edges E_{gap} and find one shortest candidate cycle in vertex-extended graph G_v for each gap edge. The distributed scheme can also be run iteratively. The initial tightest cycle C_{min} is set to be the length of the primary boundary cycle. In each round, an edge $e = (s, t)$ is randomly selected from existing gap edges and s floods messages in G_v, to construct a shortest path tree T_s with root at s in G_v. Mostly, this can be finished as what is done in Section 2.1 of [21], while the only difference is that we restrict the flooding with at most $\ell(C_{min}) - 1$ hops. If there exists a path P_e in T_s connecting s and t such that $\ell(P_e) + 1 < \ell(C_{min})$, a shorter cycle $C_{s,t} = P_e + e$ is obtained. The C_{min} is updated.

In addition, computing refined outer boundary cycles needs to extend one cycle, calculate the shortest path between, and obtain the skeleton. These steps can be achieved similarly as what we have discussed previously, so we leave out the details.

5. CORRECTNESS AND EFFECTIVENESS

This section proves the correctness of our boundary recognition algorithm and evaluate its effectiveness by simulations.

5.1 Preliminaries

We first give a brief overview on the concepts and theories involved in this work. Not all definitions are necessarily standard. For detailed explanations, refer to the books by Munkres [16], Hatcher [12].

In algebraic topology, simplicial complexes are well-defined blocks for building topological spaces and often useful for concrete calculations. A *k-simplex* σ is a set of $k+1$ size. A *simplicial complex* $\mathcal{K}$ is a collection of simplices that satisfies the following conditions:

(1) Any face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$; (2) The intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2. The dimension of a simplicial complex is equal to the largest of the dimensions of its simplices. Given two topological spaces X and Y, two continuous maps $f, g : X \to Y$ are said to be *homotopic* if there exists a continuous map $F : X \times I \to Y$ such that $F(x, 0) = f(x)$ and $F(x, 1) = g(x)$ for all $x \in X$, $I = [0, 1]$. Any such mapping is called a *homotopy* connecting f and g. A continuous map of X onto a subspace A is called a *deformation retraction* if it is homotopic to the identity map id_X, then A is called a *deformation retract* of X. A continuous map $f : X \to Y$ is called a *homotopy equivalence* if existing $g : Y \to X$ such that the composition $g \circ f$ and $f \circ g$ is homotopic to id_X and id_Y, respectively. If there exists a homotopy equivalence $X \to Y$, X is said to be *homotopy equivalent* to Y. Homotopy equivalence as a equivalence relation divides topological spaces into *homotopy classes*. A continuous mapping of the interval I into X is called a *path* in X. Closed paths are also called *loops*. A loop is *contractible* if it is homotopic to the constant loop. The set of homotopy equivalence classes of loops based at x_0 in X forms a group under concatenation, called the *fundamental group* and denoted $\pi_1(X; x_0)$. A space is called *simply connected* if it is path-connected and has trivial fundamental group. If X is path-connected, then $\pi_1(X, x_1) \simeq \pi_1(X, x_2)$ for any $x_1, x_2 \in X$; Thus, the notation $\pi_1(X, x_0)$ is often abbreviated to $\pi_1(X)$.

5.2 Proof of Correctness

In this section, we prove that topological boundaries hold the property of consistency in UDG model, and our recognition algorithm can correctly find boundaries with theoretical guarantee. We show that our definition on topological boundaries meets the requirement of consistency. We prove the extracted skeleton graph is a planar graph and features the original network faithfully. Each found primary and refined boundary cycle either exactly encircles one inner hole or corresponds to the outer boundary. Due to space limitation, only major theorems are presented and most proofs are omitted here.

As mentioned in Section 3, an embedding ε of G defines uniquely a shadow $\overline{\Delta_G^\varepsilon}$ of Δ_G. Let $\overline{\Delta_G}$ denote the set of all possible shadows of Δ_G. Following recent result of Chambers et al. (Theorem 3.1 of [2]), we have Lemma 1.

LEMMA 1. *Given UDG G, the fundamental groups of Δ_G and $\overline{\Delta_G}$ are isomorphic, $\pi_1(\Delta_G) \simeq \pi_1(\overline{\Delta_G})$.*

THEOREM 2. *The topological boundaries defined in Definition 1 keep the consistency in UDG models.*

PROOF. Given an embedding ε_1 of G, let cycle C in G be a topological boundary, then $\overline{\Delta_C^{\varepsilon_1}}$ is homotopic to a geometric boundary of $\overline{\Delta_G^{\varepsilon_1}}$ and is non-contractible in $\overline{\Delta_G^{\varepsilon_1}}$. Hence, C must be non-contractible in Δ_G from Definition 1 and Lemma 1. For another embedding ε_2 of G, $\overline{\Delta_C^{\varepsilon_2}}$ will still be non-contractible and envelop at least a hole in $\overline{\Delta_G^{\varepsilon_2}}$. Otherwise, C will be contractible in Δ_G, inducing contradiction. $\square$

In the proof, we attach topological concepts to graph G by virtue of its topological counterpart Δ_G. For instance, we say that the graphs G and H are homotopic if Δ_G and Δ_H are homotopic, and that the fundamental group $\pi_1(G)$ of G is $\pi_1(\Delta_G)$.

THEOREM 3. *FGP transformation does not change the fundamental group of a graph.*

THEOREM 4. *A graph remains its simple connectedness after exercising FGP transformation.*

From Theorem 3, we can see that our found skeleton graph is topologically equivalent to original graph in terms of fundamental group. Note that Theorem 3 is true regardless of the model of a graph, which can be of independent interest for more applications. We present more results in UDG models. G and ε in the rest theorems are of a UDG and its any one invalid embedding, respectively.

THEOREM 5. *Given G, ε and G' obtained by deleting some vertices or edges from G under FGP transformation, $\overline{\Delta_{G'}^\varepsilon}$ is a deformation retract of $\overline{\Delta_G^\varepsilon}$.*

THEOREM 6. *The skeleton graph G_S generated by our algorithm is a triangle-free planar graph, and the fundamental group of G_S is isomorphic to that of G.*

THEOREM 7. *Given G, ε and skeleton graph G_S of G, after successfully separate G_S into primary boundary cycles, each primary boundary cycle $\overline{\Delta_C^\varepsilon}$ either exactly encircles one inner hole $\overline{\Delta_G^\varepsilon}$ or is homotopic to the outer boundary of $\overline{\Delta_G^\varepsilon}$.*

THEOREM 8. *Each refined boundary cycle either exactly surrounds one inner hole or corresponds to the outer boundary.*

THEOREM 9. *Given G, it only requires local connectivity to delete (or insert) a vertex or edge on G under FGP transformation.*

5.3 Simulations

We conduct qualitative simulations to evaluate the effectiveness of this approach. We compare our approach based on topological transformations on graphs (denoted as TTG) with the state of the art approach proposed by Wang, Gao and Mitchell [21] (denoted as WGM), which is widely accepted as one of the best distributed methods using solely node connectivity to detect boundaries.

In this set of simulations, nodes are deployed using two distribution models: random placement and perturbed grid. These models have also been adopted in most existing boundary recognition algorithms [21] [18]. The bold lines in Figure 8 (a) show the skeleton graph generated by TTG in the random networks, which is in UDG model, and of 400 nodes with average node degree 8.16. Figure 8 (b) further shows the found inner and outer boundary cycles in the same network as Figure 8 (a). We can visually examine the result to confirm that TTG successfully finds all the boundaries of different sizes. Similar effects are also achieved in more examples with perturbed networks and different network scales. TTG can find both small and big holes in a random network.

We further evaluate the capability of WGM on small holes. We run WGM in the same networks as shown in Figure 4 (a) and Figure 8 (b), respectively. The first key step of WGM is to find cuts, where two threshold parameters δ_1 and δ_2 are needed to specify the minimum size of the holes to be detected. Figure 9 shows the scenarios of finding cuts in the networks with threshold for minimum 4-hop holes in (a) and 8-hop holes in (b). Bold lines show the shortest path tree with the square root while circles denote the cut nodes. The cut branches (connected cut components) found by WGM are denoted by dot lines. We can see that some cut branches are merged and some cut branches actually do not correspond to any holes, such as the bottom left one in Figure 9 (a), due to the discreteness of the shortest path tree. Through more simulations on WGM, we have the following observations. When there are a lot of small holes in a network, it usually tends to be the case that there

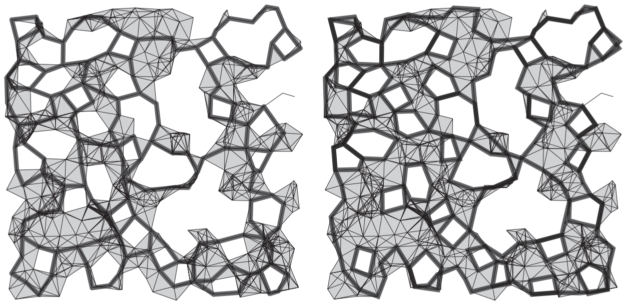

(a) Skeleton, random (b) Boundary cycles

Figure 8: Qualitative evaluation on TTG.

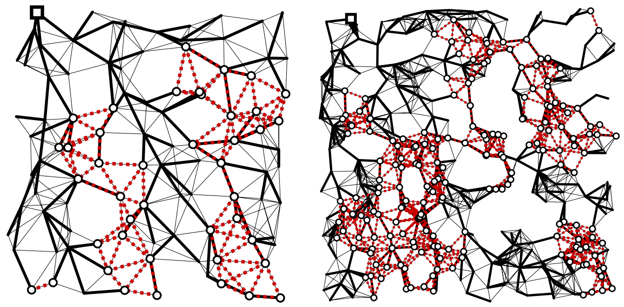

(a) Cut branches (b) Cut branches

Figure 9: Qualitative evaluation on WGM.

always exist combined cut branches no matter where to select the root node. Hence, it is difficult for WGM to acquire cut branches correctly, find candidate boundary cycles and recognize the small holes.

6. CONCLUSIONS

As a crucial issue in wireless ad hoc and sensor networks, location-free boundary recognition is previously addressed in a coarse-grained fashion. We present a formal boundary definition and the first distributed and fine-grained boundary recognition algorithm, which can locate all the network boundaries of no matter how small inner holes are, without using location information. We formally prove the correctness of this design, and evaluate its effectiveness by comparing with the state-of-the-art approach through extensive simulations.

7. ACKNOWLEDGMENTS

The authors are grateful for a variety of valuable comments from the anonymous reviewers. This work is supported in part by the Hong Kong RGC grant HKUST6169/07E, the NSFC/RGC Joint Research Scheme N_HKUST 602/08, the National Basic Research Program of China (973 Program) under grant No. 2006CB303000, the National High Technology Research and Development Program of China (863 Program) under grants No. 2002AA1Z2101, No. 2007AA01Z177 and No. 2007AA01Z180, NSFC under grants No. 60621003 and No. 90718040.

8. REFERENCES

[1] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T. Lai. Deploying wireless sensors to achieve both coverage and connectivity. In *Proc. of ACM MobiHoc*, 2006.

[2] E. W. Chambers, V. de Silva, J. Erickson, and R. Ghrist. Rips complexes of planar point sets. *Preprint, ArXiv:0712.0395*, 2007.

[3] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[4] V. de Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *International Journal of Robotics Research*, 25(12):1205–1222, 2006.

[5] Q. Fang, J. Gao, and L. J. Guibas. Locating and bypassing holes in sensor networks. In *Proc. of IEEE INFOCOM*, 2004.

[6] S. P. Fekete, M. Kaufmann, A. Kröller, and N. Lehmann. A new approach for boundary recognition in geometric sensor networks. In *Proc. 17th Canadian Conference on Computational Geometry*, 2005.

[7] S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. In *Proc. of the 1st Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.

[8] S. Funke. Topological hole detection in wireless sensor networks and its applications. In *Proc. of ACM DIALM-POMC*, 2005.

[9] S. Funke. Hole detection or:"how much geometry hides in connectivity?". In *Proc. of ACM SoCG*, 2006.

[10] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proc. of ACM MobiHoc*, 2001.

[11] R. Ghrist and A. Muhammad. Coverage and hole-detection in sensor networks via homology. In *Proc. ACM/IEEE IPSN*, 2005.

[12] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[13] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proc. of ACM SODA*, 2006.

[14] X. Li, G. Calinescu, P. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(10):1035–1047, 2003.

[15] C. Liebchen and R. Rizzi. Classes of cycle bases. *Discrete Applied Mathematics*, 155:337–355, 2007.

[16] J. R. Munkres. *Topology, Second Edition*. Prentice Hall, 2000.

[17] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proc. of ACM MobiCom*, 2003.

[18] O. Saukh, R. Sauter, M. Gauger, P. J. Marrón, and K. Rothermel. On boundary recognition without location information in wireless sensor networks. In *Proc. of ACM/IEEE IPSN*, 2008.

[19] I. Stojmenovic and X. Lin. Power-aware localized routing in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1122–1133, 2001.

[20] W. Tutte. How to draw a graph. *Proc. London Math. Soc*, 13(3):743–768, 1963.

[21] Y. Wang, J. Gao, and J. S. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proc. of ACM MobiCom*, 2006.

[22] H. Whitney. Congruent graphs and the connectivity of graphs. *Amer. J. Math*, 54(1):150–168, 1932.