Wireless Netw (2011) 17:53–67
DOI 10.1007/s11276-010-0264-0

# Separability and topology control of quasi unit disk graphs

**Jianer Chen · Anxiao (Andrew) Jiang · Iyad A. Kanj · Ge Xia · Fenghui Zhang**

Published online: 17 July 2010

**Abstract** A deep understanding of the structural properties of wireless networks is critical for evaluating the performance of network protocols and improving their designs. Many protocols for wireless networks—routing, topology control, information storage/retrieval and numerous other applications—have been based on the idealized unit-disk graph (UDG) network model. The significant deviation of the UDG model from many real wireless networks is substantially limiting the applicability of such protocols. A more general network model, the quasi unit-disk graph (quasi-UDG) model, captures much better the characteristics of wireless networks. However, the understanding of the properties of general quasi-UDGs has been very limited, which is impeding the designs of key network protocols and algorithms. In this paper, we present results on two important properties of quasi-UDGs: separability and the existence of power efficient spanners. Network separability is a fundamental property leading to efficient network algorithms and fast parallel computation. We prove that every quasi-UDG has a corresponding grid graph with small balanced separators that captures its connectivity properties. We also study the problem of constructing an energy-efficient backbone for a quasi-UDG. We present a distributed local algorithm that, given a quasi-UDG, constructs a *nearly planar* backbone with a constant stretch factor and a bounded degree. We demonstrate the excellent performance of these auxiliary graphs through simulations and show their applications in efficient routing.

**Keywords** Quasi unit disk graphs · Separability · Topology control · Spanners

A preliminary version of this paper appeared at INFOCOM 2007.

J. Chen · A. (Andrew) Jiang
Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA
e-mail: chen@cse.tamu.edu

A. (Andrew) Jiang
e-mail: ajiang@cse.tamu.edu

I. A. Kanj
School of CTI, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, USA
e-mail: ikanj@cs.depaul.edu

G. Xia (✉)
Department of Computer Science, Lafayette College, Easton, PA 18042, USA
e-mail: gexia@cs.lafayette.edu

F. Zhang
Google Kirkland, 747 6th Street South, Kirkland, WA 98033, USA
e-mail: fhzhang@gmail.com

## 1 Introduction

The connectivity structures of wireless networks exhibit strong correlations with the physical environment due to the signal transmission model of wireless nodes. A deep understanding of the structural properties of wireless networks is critical for evaluating the performance of network protocols and improving their designs. So far, many protocols have been based on the idealized unit-disk graph (UDG) network model, where two wireless nodes can directly communicate if and only if their physical distance is within a fixed parameter $R$. Examples of these protocols include routing [3, 10], topology control [1], distributed information storage/retrieval [5] and a great variety of other applications. In practice, however, the UDG model significantly deviates from many real wireless networks, due to many reasons including: multi-path fading [7, 16], antenna design issues, inaccurate node position estimation, etc. The significant deviation of the UDG model from the real/practical models is substantially limiting the applicability of protocols which are based on UDGs. To combat this problem, a more general network model, the quasi unit-disk graph (quasi-UDG) model, has been recently proposed to capture the nonuniform characteristics of (most) wireless networks [2, 14]. Formally, this model is defined as follows.

**Definition 1** A *quasi-UDG* with parameters $r$ and $R$ ($r$ and $R$ are positive numbers) over a set of points in the plane is defined as follows. The points in the set are the vertices of the graph. For any two points $u$ and $v$ in the set with Euclidean distance $|uv|$: if $|uv| \leq r$ then $uv$ is an edge in the graph; if $|uv| > R$ then $uv$ is not an edge in the graph; and if $r < |uv| \leq R$ then $uv$ may or may not be an edge in the graph.

In the *power model*, each edge of the quasi-UDG (or UDG) is associated with a power or energy cost (usually this is the cost required to transmit across the corresponding link in the network), which is commonly assumed to be the Euclidian distance between the endpoints of the edge raised to some power constant $\beta \in [2, 5]$. In the *geometric* model, the cost of an edge is simply the Euclidean distance between its endpoints.

In sharp contrast to the UDG model whose properties have been well understood ([1, 10]), the understanding of the properties of quasi-UDGs has been very limited, and has been confined mainly to the case when the parameters of the quasi-EDG satisfy $r/R \geq \sqrt{2}$ [2, 14]. Among the limited results about quasi-UDG, is a notable result about the "link-crossing" property for quasi-UDGs satisfying $r/R \geq \sqrt{2}$ [2]. In general, the serious lack of understanding of the properties of quasi-UDGs is impeding the design of key network protocols and algorithms for this model.

In this paper, we present results on two important properties of quasi-UDGs: *separability* and the existence of efficient power *spanners* of quasi-UDGs.

Network separability is a fundamental property that leads to efficient network algorithms and fast parallel computation [13]. A (vertex) *separator* of a graph $G$ is a set of vertices whose removal splits the graph into two non-adjacent parts of similar sizes. We call a graph $G$ *well separable* if any subgraph of $G$ has relatively small separators. A well separable graph has strong local properties. As a result, the performance of protocols for routing, information retrieval, network monitoring, etc., can be significantly improved for such graphs. We first construct a grid graph, which is an abstraction of the given quasi-UDG $G$, and show that the grid graph is well separable. The separator we obtain for the grid graph is of size $O(\sqrt{N})$, and can split the graph into two parts, each of size roughly $\frac{N}{2}$, where $N$ is the number of nodes of the grid graph. In addition, both the degree of the grid nodes and the number of edges crossing any given edge, are upper bounded by constants. Among many applications of the separators, we present, as an example, a compact routing protocol based on the grid graph construction and the distance labeling technique. We prove that the routing table size of each node in our protocol is bounded by $O(\sqrt{N}\log N)$, which is much better than the tight bound proved for general graphs and close to the lower bound of $\Omega(\sqrt{N})$ for bounded-degree graphs in [8]. The ratio of the routing path length to the shortest path length is upper bounded by $2 + \epsilon$, where $\epsilon$ is a small constant. Extensions of these results are also included.

In the second part of the paper we study the existence and the construction of energy efficient *spanners* for quasi-UDGs. A subgraph $H$ of a quasi-UDG $G$ is said to be a *spanner* of $G$ if there exists a constant $\rho$ such that: for every two nodes $X, Y \in G$, the weight of a shortest path between $X$ and $Y$ in $H$ is at most $\rho$ times the weight of a shortest path between $X$ and $Y$ in $G$. The constant $\rho$ is called the *stretch factor* of $H$ (with respect to $G$). A spanner, in general, is useful for efficient communication (e.g., unicasting): By using only those edges in the spanner for communication, signal interference, routing table size and power usage can be substantially reduced.

In terms of the previous work done on the construction of spanners of quasi-UDGs, Kuhn et al. [14] gave a distributed algorithm that constructs a geometric spanner with stretch factor $O(\lg \frac{1}{d})$, where $d = r/R$. Damian et al. [4] gave a distributed algorithm that constructs a spanner of a quasi-UDG in any $\ell$-dimensional Euclidean space of bounded degree and arbitrarily small stretch factor. The distributed algorithm in [4] runs in poly-logarithmic number of rounds. Moreover, the constant in the upper bound on the spanner weight is unspecified. Very recently, Lillis et al. [12] gave a *local*, i.e., runs in a constant number of communication rounds [17], distributed algorithm that constructs a geometric spanner of a quasi-UDG with constant stretch factor, when the quasi-UDG parameters satisfy $r/R \geq \sqrt{2}/2$.

Whereas local distributed algorithms for the construction of geometric backbones have recently been developed [12], these algorithms only work when the parameters of the quasi-UDG satisfy $r/R \geq \sqrt{2}/2$.

In this paper, we present a local distributed algorithm that constructs a power spanner $B$ for an arbitrary connected quasi-UDG $G$, i.e., with no restrictions on the parameters $R$ and $r$. The node degree of the spanner $B$ is upper bounded by a constant. In addition, even though it is impossible in general to construct planar spanners of quasi-UDGs with constant stretch factors, we show that $B$ is *nearly planar*. More specifically, we show that $B$ has a constant upper bound on the average number of edges crossing any given edge. The latter property is useful for geographic routing algorithms based on cross link detection [11]. In general, plane spanners of quasi-UDGs (and UDGs), when they exist, can then be used to design efficient routing schemes with guaranteed delivery, such as *perimeter routing* (face routing) [3, 10, 15].

We evaluate the performance of the separators, the routing protocol, and the spanner construction, through extensive simulations. Their performance is much better compared to the theoretical analysis of the worst cases. This shows that, although the quasi-UDG model is quite different from the UDG model, efficient algorithms can still be developed by exploiting the locality of the model.

The rest of the paper is organized as follows. In Sect. 2 we present the grid graph construction and prove its separability properties. In Sect. 3, we present the backbone construction. In Sect. 4, we present the compact routing protocol based on the grid graph and the distance labeling technique, as well as the simulation results. We conclude the paper in Sect. 5.

## 2 Grid graph of quasi-UDGs

In this section we present a distributed algorithm for constructing a grid graph for any given quasi-UDG. We prove that the grid graph is well separable, and that it has bounded node and edge density. We first start with the following definitions and notations.

A (vertex) *separator* in a graph $G$ is a set of vertices $S$ such that $G \backslash S$ is disconnected. A separator is said to be *balanced* if it separates the graph into two parts, each of which has $O(n)$ vertices, where $n$ is the number of vertices in the graph. We will always use the term separator to denote a balanced separator. We say that a separator is *small*, if the number of vertices in the separator is $o(n)$. A graph $G$ is said to be *well separable* if all subgraphs of $G$ have small separators.

A graph $G$ is said to be *planar* if it can be embedded in the plane without edge crossing. Planar graphs are known to have $O(\sqrt{n})$-size separators [13]. For quasi-UDGs with $r/R \geq \sqrt{2}/2$, the network can be planarized with the help of virtual edges, as shown in [2]. Thus, protocols that rely on the planarity of the underlying structure of the network can still be applied to quasi-UDGs with $r/R \geq \sqrt{2}$.

A quasi-UDG may have highly variable node and edge density, which prevents it from having small separators. The grid graph that we construct will serve as an abstraction of the quasi-UDG that retains some properties of the quasi-UDG, such as connectivity and distance information, and represents well the deployment region of the quasi-UDG. An example of a quasi-UDG and its corresponding grid graph is shown in parts (a) and (b) of Fig. 1, respectively.

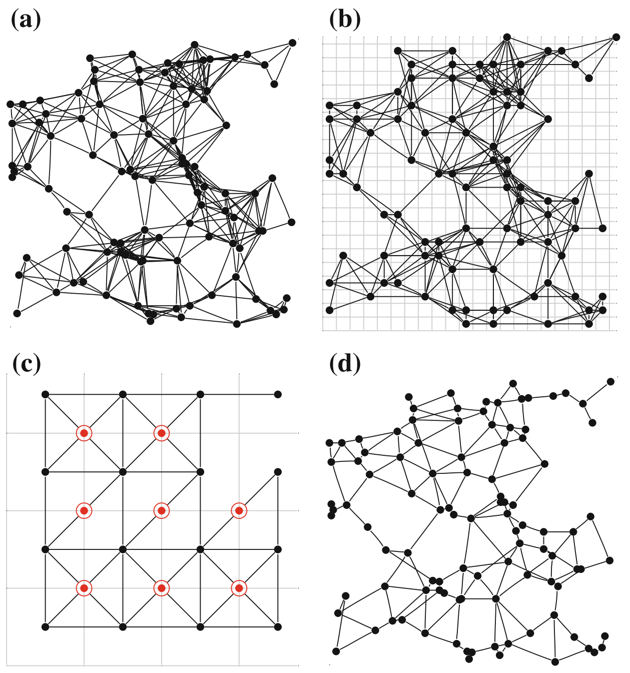


**Fig. 1** **a** A quasi-UDG $G$ with 100 vertices and $r/R = 0.5$; **b** the grid graph corresponding to $G$; **c** the auxiliary graph used to find the top level separator of $G$; **d** The backbone of $G$

### 2.1 Construction of the grid graph

Given a quasi-UDG $G$, we construct a grid graph $H$ for $G$ as follows. First, we impose a grid on the plane. The dimension of the grid cell is chosen so that all the nodes of $G$ residing in the same cell are fully connected. Then, for every non-empty cell of the grid, we create a new vertex $u$ in $H$, and map all nodes of $G$ residing in that grid cell to $u$. We add an edge between two vertices $u$ and $v$ of $H$ if and only if there are two nodes in the corresponding grid cells of $u$ and $v$ that are adjacent in $G$. This completes the construction of the grid graph $H$. The formal algorithm **GridGraph** is given in Fig. 2.

```
Algorithm GridGraph
INPUT:    $G = (V_G, E_G)$: a quasi-UDG with parameters $R$ and $r$
OUTPUT:   $H = (V_H, E_H)$: the grid graph for $G$
  1. Impose a grid of cell dimensions $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ on the plane;
  2. For each cell that has at least one node of $G$, $H$ has a corresponding vertex positioned at the center of the cell;
  3. There is an edge between two vertices of $H$ if and only if there is at least one edge connecting two nodes of $G$ that are, respectively, in the two corresponding cells.
```

**Fig. 2** Constructing the grid graph for a quasi-UDG

The algorithm **GridGraph** can be implemented in a distributed environment as follows. First, each node in $G$ obtains the smallest $x$-coordinate and $y$-coordinate of all the nodes in the network. Initially each node $v$ sets two local variables $x_{min}$, $y_{min}$ to its own coordinates. If a node $u$ is the leftmost (or topmost node) among its neighbors, it sends a message(s) $(XMIN, x_u)$ $(YMIN, y_u)$ to its neighbors notifying them of this fact. Once a node $v$ receives a message of the form $(XMIN, x_u)$ from $w$, it compares $x_u$ with its $x_{min}$. If $x_u$ is smaller, $v$ replaces its $x_{min}$ with $x_u$, and forwards the message to all its neighbors except $w$. If $x_u$ is greater or equal to $x_{min}$, $v$ simply discards the message. A similar thing happens when $v$ receives a message of the form $(YMIN, y_u)$.

In the algorithm **GridGraph**, we place the grid so that its top left corner is at point $(x_{min}, y_{min})$. Each node in $G$ knows the position of the grid, and the value of the small radius $r$ of the quasi-UDG. With this information in hand, each node in $G$ determines the grid cell it belongs to and all its neighbors (in $G$) within this grid cell. Therefore, all nodes in $G$ residing in the same cell can identify themselves as a single vertex of $H$. Identifying the edges of $H$ can now be performed by local operations.

The flooding part in the algorithm can be implemented efficiently. In particular, the same message will not travel through any edge more than once, and the only nodes in $G$ which will initiate the flooding process are those nodes which are local minima among their neighbors. Moreover, very few messages will travel throughout the whole network; those are the messages containing the smallest coordinates in the network.

The following theorem proves constant upper bounds on the node density, the edge density, and the number of edges crossing any given edge in the grid graph $H$.

**Theorem 1** *The grid graph $H$ constructed by the algorithm* **GridGraph** *has the following properties*:

1. *inside any disk of radius $y$, there are $O\left(\frac{y^2}{r^2}\right)$ vertices of $H$*;
2. *the degree of each vertex in $H$ is $O\left(\frac{R^2}{r^2}\right)$*;
3. *the number of edges crossing any given edge in $H$ is $O\left(\frac{R^4}{r^4}\right)$; and*
4. *$H$ retains the connectivity and the distance information of $G$.*

*Proof* To prove part 1, note that the Euclidean distance between any two vertices of $H$ is at least $\frac{r}{\sqrt{2}}$. Hence, if we place an open disk of radius $\frac{r}{2\sqrt{2}}$ centered at every vertex in $H$, then no two disks will intersect. Therefore, given any disk of radius $y$, the number of such open disks intersecting it is $O\left(\frac{y^2}{r^2}\right)$, and so is the number of vertices of $H$ inside that disk.

To prove part 2, consider a vertex $u$ of $H$ and denote by $V(u)$ the set of nodes of $G$ inside the cell corresponding to $u$. By part 1 above, the number of vertices of $H$ within distance $R + r$ from $u$ is $O\left(\frac{R^2}{r^2}\right)$. Note that for any other vertex $v$ in $H$, if the distance between $u$ and $v$ is larger than $R + r$, then no node in $V(u)$ can be adjacent to any node in $V(v)$. It follows that the degree of $u$ is $O\left(\frac{R^2}{r^2}\right)$.

To prove part 3, note that for an edge $\{u, v\}$ in $H$, the number of vertices in $H$ within distance $R + r$ from any point on the line segment connecting $u$ and $v$ in $H$ is $O\left(\frac{R^2}{r^2}\right)$. (This statement can be easily verified by the reader.) Therefore, the total number of edges crossing $\{u, v\}$ is $O\left(\frac{R^4}{r^4}\right)$.

Part 4 follows from the fact that any two vertices $u$ and $v$ of $H$ are $d$ hops away if and only if two nodes of $G$ in the two corresponding cells to $u$ and $v$ are at most $2d + 1$ hops away. □

## 2.2 Separability of the grid graph

Network separability is a fundamental property that leads to efficient network algorithms (for example, those algorithms based on the divide and conquer paradigms that will lead to fast parallel implementations). In particular, many applications in wireless ad hoc networks (routing, information retrieval, etc.), have more efficient solutions if the underlying graph is well separable. For example, shortest path routing can be realized with small routing tables when the graph is well separable, as in the case of planar graphs or graphs with bounded tree width [8].

In this subsection, we study the separability of the grid graph $H$ described in the previous section. We begin with the following definition.

**Definition 2** Given a graph $H$ of $n$ vertices, a $b$-**separator** of $H$ is a set of vertices whose removal splits $H$ into two subgraphs $H_1$ and $H_2$ such that no vertex in $H_1$ is adjacent to a vertex in $H_2$, and such that each of $H_1$ and $H_2$ contains at most $b \cdot n$ vertices. We call a graph $H$ $(f(n'), b)$-**separable**, if every subgraph $H'$ of $G$ on $n'$ vertices has a $b$-separator of $O(f(n'))$ vertices, where $f(n')$ is a function of $n'$.

In order to compute a small separator for the grid graph $H$, we use an auxiliary graph $T$. Similar to the construction of the grid graph, we first impose a grid $G_T$ of larger cell size on the plane. Each non-empty cell (i.e., there is at least one vertex of $H$ in that cell) of the grid is then mapped to a vertex of the auxiliary graph $T$ placed at the center of that cell. Two vertices of $T$ are adjacent if and only if there are two adjacent vertices of $H$ in the two corresponding cells of the larger grid. We choose the cell dimension of $G_T$ so that each vertex $u$ of $T$ at the center of cell $C$ is adjacent only to the vertices at the centers of the neighboring cells of $C$. Therefore each edge of $T$ is either horizontal, vertical, diagonal or anti-diagonal. It is clear that an edge-crossing can only involve a diagonal edge and an anti-diagonal edge. Each of these diagonal edges and anti-diagonal edges can cross at most one other edge. Finally, we planarize $T$ by adding a virtual vertex $\mathcal{V}$ at each edge-crossing (placed at the crossing point) and replace the two crossing edges $e_1$,$e_2$ with four short edges incident to $\mathcal{V}$ and the four endpoints of $e_1$ and $e_2$, thus eliminating all edge-crossings. Note that we consider all the edges to be straight line segments. The detailed construction of the auxiliary graph $T$ is presented in Fig. 3. All the virtual vertices in $T$ are *red (circled) vertices*, and the remaining vertices, which are at the centers of the grid cells in $T$, are *black vertices*. We assign each red vertex weight zero, and we assign each black vertex $u$ a weight equals to the number of vertices of $H$ in the cell corresponding to $u$.

Figure 1(c) shows an example of the auxiliary graph. The longest edge in the auxiliary graph has length $R + \frac{\sqrt{2}r}{2}$, and red vertices are either of degree 2 or 4. Since the cell dimension in the grid we apply is large enough (of side length $R + \frac{r}{\sqrt{2}}$) and all black vertices are placed at the centers of their corresponding cells, any black vertex may only connect to the eight black vertices around it before the red vertices were added. Therefore, around each black vertex, there can be at most four red vertices, and no two red vertices are adjacent to each other. Formally, we have the following lemma:

**Lemma 1** *Let $N_{T,b}$ be the number of black vertices in the auxiliary graph $T$. Then $T$ is a planar graph of at most $2N_{T,b}$ vertices, and no two red vertices in $T$ are adjacent.*

Lipton and Tarjan proved in their celebrated Separation Theorem [13] that for any vertex-weighted planar graph of $n$ vertices, there exists a set of $O(\sqrt{n})$ vertices that separates the graph into two non-adjacent subgraphs, each of which weighs at most $\frac{2}{3}$ of the total weight of the graph. The separator algorithm presented in [13], however, is relatively complex. For the planar auxiliary graph $T$, which has a constrained structure, we present a simpler and more practical algorithm for finding such a small separator. Based on that, the algorithm also finds a small separator for the grid graph $H$.

Note that $T$ is actually a plane graph, i.e., it is an embedded planar graph. To find a separator for $T$, the idea is to "peel off" the outer face of $T$ repeatedly to find a thin cut. To accomplish this, we build a BFS tree rooted at a vertex on the outer face of $T$. When a vertex $u$ is discovered in the process, we mark all undiscovered vertices that share faces with $u$ (denoted by $F(u)$) so that they will be placed into a level that is no later than the next level of the BFS tree. This can be done by adding edges from $u$ to all vertices in $F(u)$. After we have constructed this BFS tree, one of the *fundamental cycles* (a cycle formed by exactly one non-tree edge and some tree edges) will contain the desired separator, i.e., a separator that separates $T$ in a balanced way. The details of the algorithm are presented in Fig. 4.

We now prove that the algorithm **Separator** constructs small balanced separators for $H$ and $T$. We start with the following lemma:

```
Algorithm AuxiliaryGraph
INPUT:  $H = (V_H, E_H)$: a grid graph with parameters $R$ and $r$
OUTPUT: $T = (V_T, E_T)$: the auxiliary graph for $H$
  1. Impose a grid of cell dimensions $(R + \frac{r}{\sqrt{2}}) \times (R + \frac{r}{\sqrt{2}})$ on the plane;
  2. For each cell that contains at least one vertex of $H$, $T$ has a corresponding black vertex $v$ whose position is set at the center of the cell; we assign to $v$ a weight equals to the number of vertices of $H$ in that cell;
  3. Add an edge between two black vertices $u$ and $v$ of $T$ if and only if there is at least one edge connecting two vertices of $H$ that are, respectively, in the two corresponding cells;
  4. For each pair of crossing edges $\{u, v\}$, $\{w, x\}$, add a red vertex at the intersection of the two edges, and replace those two original edges with four new edges that connect the red vertex, respectively, to the four black vertices $u$, $v$, $w$, and $x$; let the weight of the red vertex be 0;
  5. For each diagonal/anti-diagonal edge between two black vertices, add a red vertex of weight 0 at the middle of the edge, and replace that original diagonal edge with two new edges that connect the red vertex, respectively, to those two black vertices.
```

**Fig. 3** AuxiliaryGraph($H$)

```
Algorithm Separator
INPUT:  $H$: a grid graph with parameters $R$ and $r$
OUTPUT: $S_H$: a separator for $H$,
        $S_T$: a separator for $T$ ($T$ is the auxiliary graph of $H$)
  1. Let $T$ be the auxiliary graph of $H$; Let $T'$ be a copy of $T$;
  2. Build a breadth-first search (BFS) tree for a dynamically changing graph $T'$ ($T'$ changes because new edges are added to it during the BFS procedure) in the following way: (1) pick a vertex $v$ on the outer face of $T'$ to be the root and start BFS; (2) during the BFS process, when a vertex $u$ is discovered (put it into the BFS tree), for every face containing $u$, add edges from $u$ to as many other vertices in the face as possible as long as $T'$ remains a simple planar graph; if after adding those edges there are still faces containing $u$ that are not triangulated, add edges to triangulate them arbitrarily; During BFS, the undiscovered neighbors of a vertex are visited in the clockwise order;
  3. Check every fundamental cycle in the BFS tree; Let $S_T$ be a fundamental cycle that separates $T'$ (and therefore $T$) in the most balanced way, i.e. the difference between the summation of the weights of vertices in the two separated subgraphs $A_1, B_1$ is minimized;
  4. Consider the graph $T$; Let $S'_T$ be a copy of $S_T$; For each red vertex $u$ in $S'_T$ with the set of neighboring vertices $N(u)$, we distinguish two cases: Case (1) All vertices in $N(u)$ belong to $A_1$(respectively, $B_1$) except those in $S'_T$; We move $u$ from $S'_T$ to $A_1$(resp. $B_1$); Case (2) Both $A_1$ and $B_1$ contain vertices of $N(u)$; We put all vertices in $N(u)$ into $S'_T$ and move $u$ from $S'_T$ to $A_1$;
  5. Let $S_H$ be the set of vertices of $H$ in those cells corresponding to the black vertices of $T$ in $S'_T$; Let $A_2$, $B_2$ be the two sets of vertices of $H$ in those cells corresponding to the black vertices of $T$ in $A_1$ and $B_1$. Clearly, $S_H$ separates $H$ into $A_2$ and $B_2$.
```

**Fig. 4** The Algorithm **Separator**

**Lemma 2** *Let $\hat{T}$ be any subgraph of the auxiliary graph T. If its outer face has k vertices, then the number of inner vertices (the vertices not on the outer face) is at most $\left\lfloor \frac{k^2}{2\pi} \right\rfloor$.*

*Proof* The outer face of the plane graph $T'$ is a closed curve (or closed curves if $\hat{T}$ is disconnected) on the plane. Let $x = R + r/\sqrt{2}$ be the side-length of the cells in the construction of the auxiliary graph $T$. For each inner vertex of $\hat{T}$, we place a $\frac{\sqrt{2}x}{2} \times \frac{\sqrt{2}x}{2}$ square centered at it, then rotate the square by 45 degrees. It is easy to see that now these (diamond shaped) squares centered at the inner vertices do not overlap each other. The area of each square is $\frac{x^2}{2}$.

First consider the case when the outer face is connected, i.e. $\hat{T}$ is connected. The outer face of $\hat{T}$ consists of several (at least one) simple cycles. Suppose there are $i$ such simple cycles of lengths $k_1$, $k_2$, ..., $k_i$ in the outer face. Note that the value $\sum_{j=1}^{i} k_j$ can be greater than $k$—the number of vertices in the outer face—because a vertex may have been counted more than once. The simple cycles form the outer face of a planar graph, so the number of times vertices are over-counted is exactly $i - 1$. Thus $\sum_{j=1}^{i} k_j = k + i - 1$.

We have:

$$\begin{aligned} k^2 &= \left[\left(\sum_{j=1}^{i} k_j\right) - i + 1\right]^2 \\ &= \sum_{j=1}^{i} k_j^2 + \sum_{j=1}^{i}\left(k_j \sum_{l \neq j}^{i} k_l\right) - \sum_{j=1}^{i} 2(i-1)k_j + (i-1)^2 \\ &\geq \sum_{j=1}^{i} k_j^2 + \sum_{j=1}^{i}\left[k_j\left(\sum_{l \neq j}^{i} k_l - 2(i-1)\right)\right] \\ &\geq \sum_{j=1}^{i} k_j^2. \end{aligned}$$

The last inequality holds because $k_j \geq 2$ and $\sum_{l \neq j} k_l$ contains exactly $i - 1$ terms. The equality holds when $i = 1$.

Each simple cycle of $k_j$ vertices has $k_j$ edges; thus the perimeter of the cycle is at most $k_j x$. Therefore the area of the region inside the cycle of length $k_j$ is at most $\left\lfloor \frac{k_j^2 x^2}{4\pi} \right\rfloor$, and the total area of the regions inside the outer face is bounded by $\sum_{j=1}^{i} \left\lfloor \frac{k_j^2 x^2}{4\pi} \right\rfloor \leq \left\lfloor \frac{k^2 x^2}{4\pi} \right\rfloor$.

Now if there are several disconnected cycles in the outer face, each connected part, say of $k'$ vertices, surrounds a region of area no more than $\left\lfloor \frac{k'^2 x^2}{4\pi} \right\rfloor$. Since $\sum k'^2 \leq \left(\sum k'\right)^2 = k^2$, the total area of the regions surrounded by the outer face is also bounded by $\left\lfloor \frac{k^2 x^2}{4\pi} \right\rfloor$. Thus, in all cases, the total number of inner vertices is bounded by $\left\lfloor \frac{k^2 x^2}{4\pi} \right\rfloor / \left(\frac{x^2}{2}\right) = \left\lfloor \frac{k^2}{2\pi} \right\rfloor$. □

Define the *depth* of a tree to be the maximum number of edges on any root-leaf path in the tree. We have the following lemma:

**Lemma 3** *Let $N_T$ be the number of vertices in the auxiliary graph T. The BFS tree constructed in Step 2 of the algorithm* **Separator** *is of depth at most $\sqrt{N_T}$.*

*Proof* Let $d$ be the depth of the BFS tree. Because of the triangulation operation enforced on the graph $T'$ during the BFS process, for $i = 1, 2, ..., d - 1$, level $i$ (if $i = 1$, include the root as well) of the BFS tree contains all the vertices on the outer face of the subgraph induced by the vertices at levels $i$, $i + 1$, ..., $d$. So it suffices to show that if we "peel off" one outer face from $T'$ at each step, $T'$ becomes an empty graph after $t \leq \sqrt{N_T}$ steps.

Let $n_x$ be the number of vertices remaining in the graph $T'$ after $x$ steps, and define $n_0 = N_T$. By Lemma 2, we know that in the $x$th step we have "peeled off" at least $\lceil \sqrt{2\pi N_x} \rceil$ vertices. Hence, $n_{t-1} \geq 1$, and $n_i \geq n_{i+1} + \lceil \sqrt{2\pi n_{i+1}} \rceil$ for $i = t - 2, t - 3, ..., 0$. Now let us prove that $n_{t-j} \geq j^2$ by induction. When $j = 1$, we have $n_{t-1} \geq 1$, and when $j = 2$, we have $n_{t-2} \geq 4$. Suppose our claim is true for $2 \leq j \leq i$, and consider the case $j = i + 1$. Then

$$\begin{aligned} n_{t-(i+1)} &\geq n_{t-i} + \left\lceil \sqrt{2\pi n_{t-i}} \right\rceil \geq i^2 + \left\lceil \sqrt{2\pi} \right\rceil i \geq i^2 + 2i + 1 \\ &= (i+1)^2. \end{aligned}$$

We have $N_T = n_0 = n_{t-t} \geq t^2$. So $t \leq \sqrt{N_T}$. □

By Lemma 2 in [13], if a vertex-weighted planar graph has a spanning tree of depth $h$, then there exists a fundamental cycle of size at most $2h + 1$ that separates the graph into two non-adjacent subgraphs, each of which weighs no more than 2/3 of the total weight of the graph. As the BFS tree obtained in Step 2 of Algorithm **Separator** is of depth at most $\sqrt{N_T}$, we have the following theorem:

**Theorem 2** *Let $N_T$ be the number of vertices in the auxiliary graph T, and let $N_H$ be the number of vertices in H. Then the total weight of the vertices of T is $N_H$. Moreover, the set $S_T$ obtained in Algorithm* **Separator** *contains at most $2\sqrt{N_T} + 1$ vertices and separates T into two non-adjacent subgraphs, each of which weighs no more than $\frac{2N_H}{3}$.*

We now prove that the algorithm **Separator** also finds a small balanced separator for the grid graph $H$.

**Theorem 3** *Let $N_H$ be the number of vertices in the grid graph H. Then the algorithm* **Separator** *constructs a separator $S_H$ of size $O\left(\sqrt{N_H}\right)$ that separates H into two non-adjacent subgraphs each of which has no more than $\frac{2N_H}{3}$ vertices. Moreover, the grid graph H is $\left(\sqrt{n'}, \frac{2}{3}\right)$-separable when the weights of all the vertices of H are set to 1.*

*Proof* Let $N'$ be the number of black vertices in $T$. Clearly $N' \leq N_H$; and it is straightforward that each cell corresponding to a black vertex of $T$ contains at most

$\lceil\frac{2(R+\sqrt{2}r/2)^2}{r^2}\rceil$ vertices of $H$. Hence we have $N' = \Theta(N_H)$. From Lemma 1 we know that the number of red vertices is no more than $N'$, and the total weight of vertices in $T$ is $N_H$. Hence, the separator $S_T$ of $T$ contains no more than $2\sqrt{2N'}+1$ vertices, whose weights sum up to $O(\sqrt{N_H})$, that separates $T$ into two parts each of which weighs no more than $\frac{2N_H}{3}$.

Now we show that after Step 4 of Algorithm **Separator**, $S'_T$ is still a separator for $T$ of size $O(\sqrt{N'})$, and $A_1$ and $B_1$ are still of weights no more than $\frac{2N_H}{3}$. Consider any red vertex $u \in S'_T$ in Step 4. In the case when all of $u$'s neighbors are either in $S_T$ or in $A_1$ (resp. $B_1$), $S'_T \setminus \{u\}$ separates $T$ into $A_1 \cup \{u\}$ and $B_1$ (resp. $A_1$ and $B_1 \cup \{u\}$). Note that $u$ has weight 0, so moving $u$ from $S'_T$ to $A_1$ (or $B_1$) does not change their weights. In the other case, the algorithm moves all of $u$'s neighbors into $S'_T$, and moves $u$ into $A_1$; clearly $S'_T$ still separates $A_1$ and $B_1$, and by doing that, we decrease the weights of both $A_1$ and $B_1$. The size of $S'_T$ increases by at most 3 for each red vertex.

Hence, after Step 4, we have replaced all red vertices in $S'_T$ by black ones, increasing the size of $S'_T$ by at most three times, and not increasing the weights of $A_1$ and $B_1$. Most importantly, $S'_T$ still separates $A_1$ and $B_1$. Therefore, $S'_T$ is still of size $O(\sqrt{N'}) = O(\sqrt{N_H})$, and the weights of $A_1$ and $B_1$ are no more than $\frac{2N_H}{3}$. Each cell corresponding to a black vertex of $T$ contains a bounded number of vertices of $H$, so $S_H$ is of size $O(\sqrt{N_H})$. Also, the number of vertices in $A_2$ (resp. $B_2$) equals the weight of $A_1$ (resp. $B_1$), which is at most $\frac{2N_H}{3}$.

By the construction of the auxiliary graph $T$, if no two black vertices in $T$ are joined by an edge or by two edges with a red vertex in the middle, then there is no edge connecting vertices of $H$ in those two corresponding cells. $A_1$ and $B_1$ are not adjacent in $T$, and $S'_T$ has no red vertex. So $A_2$ and $B_2$ obtained in Step 5 are not adjacent in $H$, and $S_H$ separates $A_2$ and $B_2$ in $H$.

It is easy to see that any subgraph of $H$ can be used as the input to Algorithm **Separator**, and the above arguments will still hold. It follows that $H$ is $(\sqrt{n'}, \frac{2}{3})$-separable. □

In some applications, a perfectly balanced separator is desirable. By using the same technique described in [13], we can construct a separator of size $O(\sqrt{N_H})$ that separates $H$ into two parts, each of which has no more than $\frac{N_H}{2}$ vertices. The idea is to separate the larger part in the outcome of the algorithm recursively. Therefore, we have:

**Corollary 1** *Let $N_H$ be the number of vertices in the grid graph $H$. Then $H$ is $(\sqrt{n'}, \frac{1}{2})$-separable.*

For the grid graph, we can develop a shortest-path routing scheme based on its separability, using the idea of distance labeling [8]. We can then transform it into a compact routing scheme, with a small stretch factor, for the underlying quasi-UDG $G$. The following theorem summarizes the result. We defer the details of the routing algorithm, the proof of Theorem 4, and the extended results to Sect. 4.

**Theorem 4** *For any quasi-UDG $G$ of $N_G$ vertices, and any two vertices $u$ and $v$ in $G$, let $h(u, v)$ be the minimum hop distance between vertices $u$ and $v$. There is a routing protocol that, for any two vertices $u$ and $v$ in $G$, guarantees a routing path from $u$ to $v$ of at most $2h(u,v) + 1$ hops. Moreover, the size of the routing table at each node and the message overhead are both $O(\sqrt{N_G}\log N_G)$.*

## 2.3 Separability for degree/edge crossing bounded graphs

The technique described in the previous section can be used to find balanced separator for any graph $G$ deployed in the plane, whose degree is bounded by a constant, and in which the number of edges crossing any given edge is also bounded by a constant. The algorithm is outlined in Fig. 5.

The following theorem proves that the above algorithm finds a balanced separator for graph $G$ of size $O(\sqrt{n})$.

**Theorem 5** *Let $G$ be a graph whose degree is bounded by a constant $D$, and in which the number of edges crossing any given edge is bounded by a constant $C$. Then the algorithm* **Separator-Gen** *constructs a* $(\sqrt{n}, O(1))$-separator for $G$.

*Proof* It is easy to see that the auxiliary graph $T$ has at most $D{\cdot}n/2$ edges, and $(\frac{C\cdot D}{4}+1)n$ vertices. Hence, the separator $S$ found in step 2 by the Lipton–Tarjan's algorithm has size $O(\sqrt{n})$, and separates the graph $T$ into two equal parts, each of size $N$, such that $n/2 - O(\sqrt{n}) \le N \le (\frac{C\cdot D}{4}+1)n/2$. The third step adjusts the separator so that the separator does not contain any virtual nodes. Let $T'$ be the resulting graph after step 3 of the algorithm. The separator obtained separates $T'$ and has size $O(\sqrt{n})$. The graph $T'$ also has $O(n)$ nodes, and the

```
Algorithm Separator-Gen
INPUT:    $G = (V_G, E_G)$: a graph with bounded degree and
          bounded edge-crossings
OUTPUT:   A separator for G
  1. Add a virtual vertex at each edge-crossing to obtain a plane
     auxiliary graph $T$;
  2. Use Lipton-Tarjan's algorithm to find a balanced separator $S$ for
     $T$;
  3. If a virtual vertex $u$ is in $S$, replace it by all four endpoints
     (original nodes) of the two edges crossing at $u$ and remove $u$
     from $T$;
  4. return $S$.
```

**Fig. 5** Separator-Gen($G$

separator returned separates $T'$ into two parts $T_1$, $T_2$ (each of size $(N \pm O(\sqrt{n}))$. More importantly, every virtual vertex in $T_1$ and $T_2$ is surrounded by real (original) nodes. Since each original node can create at most $C \cdot D$ many virtual vertices, each of $T_1$ and $T_2$ must contain at least $[(N - O(\sqrt{n})]/(C \cdot D + 1) > \frac{n}{2(C \cdot D+1)}$ many real nodes. Clearly $S$ also separates $G$ into two parts, each of size at least $\frac{n}{2(C \cdot D+1)}$ (at most $\frac{n[2(C \cdot D+1)-1]}{2(C \cdot D+1)} = O(n)$). Hence, this separator is a $(\sqrt{n}, O(1))$-separator for $G$. □

Applying the same technique described in [13], we can obtain a perfectly balanced separator for graphs with bounded degree and edge-crossings.

**Corollary 2** *A graph $G$ of $n$ nodes, whose degree is $O(1)$ and in which the number of edges crossing any given edge is $O(1)$, is $\left(\sqrt{n}, \frac{1}{2}\right)$-separable.*

## 3 Backbone with constant stretch factor (spanners)

We denote by a *backbone* of a given graph any spanning subgraph. Examples of backbones are spanning trees. Backbones, particularly those with small stretch factors (also called *spanners*) and node degrees, have very important applications in wireless communication because they help reduce signal interferences and simplify algorithms (such as routing algorithms).

A distributed algorithm is said to be $k$-local [17], if it runs in $k$ communication rounds. A distributed algorithm is *local* if it is $k$-local for some constant $k$. Local distributed algorithms, in general, tend to be more scalable and more robust to topological changes.

In this section, we present a distributed local construction of a backbone for a quasi-UDG with constant stretch factor, constant node degree, and a small number of edge crossings. We will show in Sect. 4 how these backbones can help reduce the routing table size in routing schemes.

### 3.1 Backbone construction

Energy is a major limitation in wireless networks. Accordingly, the stretch factor of a backbone is often defined based on energy consumption. We start with the formal definition.

**Definition 3** Denote by $|uv|_G$ the Euclidian distance between any two nodes $u$ and $v$ in $G$. Let $(u = u_1, u_2, \ldots, u_k = v)$ be a path from $u$ to $v$ in the graph $G$. The **communication cost** between $u$ and $v$ along the above path is defined as:

$$c_G(u,v) = \sum_{i=1}^{k-1} \alpha |u_i u_{i+1}|_G^{\beta},$$

where $\beta$ is the power exponent satisfying $2 \leq \beta \leq 5$, and $\alpha$ is a scaling factor that is linear in the number of bits sent. If there is no path from $u$ to $v$ then $c_G(u, v)$ is defined to be $+\infty$.

**Definition 4** Given a graph $G = (V, E)$ and a backbone $B$ of $G$, the **stretch factor** of $B$ is defined as:

$$\max_{u,v \in V} \left\{ \frac{c_{B,min}(u,v)}{c_{G,min}(u,v)} \right\},$$

where $c_{B,min}(u, v)$ and $c_{G,min}(u, v)$ denote the *minimum* communication cost (over all paths) between $u$, $v$ in the graph $B$ and $G$, respectively.

The stretch factor defined above is also called the *power stretch factor*. We say that a backbone is *energy efficient* if its power stretch factor is bounded by a constant.

We next present a distributed local algorithm that, given a quasi-UDG $G$, constructs a backbone $B$ whose maximum degree is $O\left(\frac{R^2}{r^2}\right)$, the average number of edges crossing a given edge is $O\left(\frac{R^4}{r^4}\right)$, and the power stretch factor is bounded by $3 + \epsilon$, for any constant $\epsilon > 0$.

We classify the edges in the quasi-UDG $G$ into two types: **short edges** whose length is not greater than $r$, and **long edges** whose length is strictly larger than $r$.

Let $E_{short}$ the set of short edges. Then the graph induced by $E_{short}$ is a UDG of unit length $r$. Denote this graph by $U_{short}$, and note that $U_{short}$ may not be connected.

The results in [9] describe a distributed local algorithm that, given a UDG $U$ and a positive integer $k \geq 9$ as a parameter, constructs a planar power spanner of the graph with degree at most $k + 5$ and stretch factor $1 + (2\sin(\pi/k))^{\beta}$, where $\beta$ is the power exponent. The algorithm described in [9] is very simple and is easy to implement. It consists of two phases: a *sparsification* phase and an *edge-selection* phase. In the sparsification phase a sparse (having a linear number of edges) spanning subgraph of $U$ is constructed. This subgraph is the *Gabriel subgraph* of $U$ [6]. In addition to its sparseness, the Gabriel subgraph is planar and has power stretch factor equals to 1. Moreover, the Gabriel subgraph can be constructed by a distributed local algorithm very efficiently. However, the Gabriel subgraph may have unbounded degree. To bound the degree, a subgraph of the Gabriel graph is then constructed by dropping edges from the Gabriel subgraph, while not hurting the stretch factor by much; this constitutes the edge-selection phase. To do that, a variant of the *Yao construction* [18] is applied in which the area around a point of $U$ is divided into cones, and then edges are selected from these cones based on some appropriate criteria. As shown in [9], the resulting spanner satisfies all the required properties, and can be easily and efficiently constructed by a local distributed algorithm. The results in [9] cannot be applied directly to $G$ since $G$ is not a UDG.

However, since each component in $U_{short}$ is a UDG, we can apply the algorithm in [9] to each component of $U_{short}$ to construct a plane power spanner of the component of degree bounded by $k + 5$ and stretch factor $1 + (2\sin(\pi/k))^{\beta}$. Let $B$ be the set of edges in the spanners of all these components.

Now impose a grid of cell-size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ on the plane. Note that any two points in the same cell are connected in $G$, and that any long edge must connect points in two different cells. For each pair of cells, add to $B$ the shortest edge between those two cells (i.e., the shortest edge having one endpoint in one of the two cells and the other endpoint in the other cell). Observe that determining the shortest edge between two cells can be done in a local fashion since the points in a cell form a clique. This completes the construction of $B$. Note that after adding the long edges to $B$, $G'$ may no longer be planar. However, as we will show below, the average number of edges crossing a given edge will be bounded by a constant. The algorithm is summarized in Fig. 6.

Call two grid-cells *adjacent* if there is an edge between a node in the first cell and another node in the second cell.

**Lemma 4** *The number of cells that are adjacent to a given cell is* $O(R^2/r^2)$.

*Proof* Fix a cell $C$ and let $C'$ be an adjacent cell to $C$. Let $O$ be the center of $C$, i.e., the point located at the intersection of the diagonals of $C$. Since $C$ and $C'$ are adjacent, there exists a point $M$ in $C$ that is adjacent to a point $M'$ in $C'$, and hence, $|MM'| \leq R$. It is easy to verify that any point in $C'$ is at a distance of at most $O(R + r)$ from $O$. Therefore, all cells adjacent to $C$ must be completely contained within a disk $D$ of center $O$ and radius $O(R + r)$. Since the disk $D$ has area $O((R + r)^2)$ and each cell has area $O(r^2)$, the number of such cells that completely reside within $D$ is $O(R^2/r^2)$. It follows that the number of cells adjacent to $C$ is $O(R^2/r^2)$. □

```
Algorithm  QuasiUDG-Backbone
INPUT:   G: a quasi-UDG with parameters R and r; an integer
         parameter $k \geq 8$
OUTPUT:  B: a backbone of G
  1. let $U_{short}$ be the UDG induced by the set of edges in $G$ of length
     at most $r$;
  2. let $B$ be the set of edges computed by applying the algorithm
     in [19] to each component of $U_{short}$ to compute a spanner of the
     component;
  3. impose a grid of cell-size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ on the plane;
  4. for every two grid-cells do
         add to $B$ a shortest edge between the two cells (in case such
         an edge exists);
  5. return $B$;
```

**Fig. 6** Construction of a backbone for a given quasi-UDG

**Lemma 5** *The backbone B constructed by the algorithm* **QuasiUDG-Backbone** *is a spanning subgraph of G.*

*Proof* Since the algorithm in [9] constructs a backbone for each component in $U_{short}$, $B$ contains a spanning subgraph for each component in $U_{short}$. Since all the vertices in a given grid-cell form a clique, and hence belong to the same component in $U_{short}$, all the vertices in a given cell remain connected in $B$. From step 4 of the algorithm, every two adjacent cells that were adjacent in $G$, remain adjacent in $B$ by virtue of adding the shortest edge between the two cells to $B$. It follows from the above, and from the connectivity of $G$, that $B$ is a spanning subgraph of $G$. □

**Lemma 6** *The number of long edges crossing a given short edge in B is* $O(R^2/r^2)$.

*Proof* Let $e_{short}$ be a short edge in $B$. Then $e_{short}$ must join two nodes in the same grid-cell $C$. Any long edge in $B$ that crosses $e_{short}$ must join two nodes, each located in an adjacent cell to $C$. By Lemma 4, cell $C$ has $O(R^2/r^2)$ adjacent cells. Therefore, $C$ has $O(R^4/r^4)$ pairs of adjacent cells. Since exactly one edge between any two adjacent cells is kept in $B$, the total number of long edges crossing $e_{short}$ is $O(R^4/r^4)$. □

**Lemma 7** *The number of edges crossing a given long edge in B is* $O(R^4/r^4)$.

*Proof* Let $e_{long}$ be a long edge. Suppose that the endpoints of $e_{long}$ reside in the two cells $C$ and $C'$. Any long edge crossing $e_{long}$ must join a node in a cell that is adjacent to either $C$ or $C'$ to another node in a cell adjacent to $C$ or $C'$. Since there are $O(R^4/r^4)$ pairs of cells that are adjacent to either $C$ or $C'$, and since exactly one edge between any two adjacent cells is kept in $B$, the total number of long edges crossing $e_{long}$ is $O(R^4/r^4)$. □

**Theorem 6** *For any integer parameter $k \geq 9$, the algorithm* **QuasiUDG-Backbone** *constructs a backbone of the given quasi-UDG G whose maximum degree is $O\left(\frac{R^2}{r^2}\right)$, average number of edges crossing a given edge is $O\left(\frac{R^4}{r^4}\right)$, and power stretch factor is $3 + 2^{\beta+1}\sin^{\beta}(\pi/k)$ (which, for any $\epsilon > 0$, is bounded by $3 + \epsilon$ for large enough k). Moreover, the algorithm is local and exchanges $O(m)$ messages, where m is the number of edges in G.*

*Proof* To prove the bound on the degree, note that since each node in the spanner of $U_{short}$ constructed by the algorithm in [9] has degree bounded by $k + 5$, each node in $B$ has at most $k + 5$ short edges incident on it in $B$. By Lemma 4, for any node in $B$ the number of cells adjacent to the cell containing the node is $O(R^2/r^2)$. Therefore, any node in $B$ can have at most $O(R^2/r^2)$ long edges incident on it in $B$. It follows that every node has $O(R^2/r^2)$ incident edges in $B$.

To prove the bound on the number of edge-crossings, note that since the algorithm in [9] constructs planar spanners of the components of $U_{short}$, no two short edges in $B$ cross. Therefore, the number of edge-crossings is the number of crossings between short edges and long edges, plus the number of crossings between long edges. By Lemma 6, the number of edges crossing a given short edge is $O(R^4/r^4)$. Therefore, the total number of edge-crossings involving short edges is $O(m{\cdot}R^4/r^4)$. By Lemma 7, the number of long edges crossing a given long edge is $O(R^4/r^4)$. Therefore, the total number of edge-crossings involving long edges is $O(m{\cdot}R^4/r^4)$. It follows that the total number of edge-crossings is $O(m\ {\cdot}R^4/r^4)$, and hence the average number of edges crossing a given edge is $O(R^4/r^4)$.

To prove the bound on the stretch factor, first note that by Lemma 5, the backbone $B$ is a spanning subgraph of $G$. We now show that every edge in $G$ is stretched by no more that $3 + 2^{\beta+1} \sin^{\beta}(\pi/k)$ in $B$. Let $e = (u, v)$ be an edge in $G$. If $e \in B$ then the statement follows directly. Suppose now that $e \notin B$. If $e$ is a short edge, then since $B$ contains a power spanner of $U_{short}$ with stretch factor $1 + 2^{\beta} \sin^{\beta}(\pi/k)$, the result follows. If $e$ is a long edge, let $C_u$ and $C_v$ be the two cells containing $u$ and $v$, respectively. Let $e_{min} = (u', v')$ be the shortest edge between $C_u$ and $C_v$ that was included in $B$, where $u' \in C_u$ and $v' \in C_v$. Since $B$ contains a power spanner of $C_u$ of stretch factor $1 + 2^{\beta} \sin^{\beta}(\pi/k)$, there is a path $P_{uu'}$ in $B$ from $u$ to $u'$ of cost at most $1 + 2^{\beta} \sin^{\beta}(\pi/k)|uu'|^{\beta}$. Similarly, there is a path $P_{v'v}$ in $B$ from $v'$ to $v$ of cost at most $1 + 2^{\beta} \sin^{\beta}(\pi/k)|vv'|^{\beta}$. It follows that the path from $u$ to $v$ in $B$ consisting of the concatenation of $P_{uu'}, e_{min}$, and $P_{v'v}$, has cost bounded by $1 + 2^{\beta} \sin^{\beta}(\pi/k)|uu'|^{\beta} + 1 + 2^{\beta} \sin^{\beta}(\pi/k)|vv'|^{\beta} + |u'v'|^{\beta}$. Since $(u, u')$ and $(v, v')$ are both short edges, and hence, are shorter than the long edge $e = (u, v)$, and since $e_{min}$ is not longer than $e$, it follows that there is a path in $B$ of cost at most $3 + 2^{\beta+1} \sin^{\beta}(\pi/k)|uv|$. Noting that $2^{\beta+1} \sin^{\beta}(\pi/k)$ can be made arbitrarily small by choosing a sufficiently large parameter $k$, the the stretch factor can be bounded by $3 + \epsilon$ for any $\epsilon > 0$.

Finally, to analyze the number of messages exchanged by the algorithm, note first that algorithm in [9] exchanges $O(m)$ messages. To compute the shortest edge between two adjacent cells, each node in the cell computes the shortest edge incident on it and whose other point is in the other cell. Then vertices in one cell elect the shortest among all these edges. Since all the vertices in one cell form a clique, and since the number of adjacent cells to a given cell is $O(R^2/r^2)$, the total number of messages exchanged for computing the shortest edges between adjacent cells is $O(m)$. Moreover, all the computation can be done locally: every node only communicates with its neighbors. It follows that the total number of messages exchanged by the algorithm is $O(m)$. □

The following theorem is then straightforward by Corollary 2.

**Theorem 7** *The backbone constructed by the algorithm* **QuasiUDG-Backbone** *is* $(\sqrt{n}, 0.5)$*-separable where* $n$ *is the number of nodes in the network.*

## 4 Applications and performance evaluation

In this section, we first present a routing algorithm based on the separators and prove upper bounds on the stretch factor of the routing algorithm. In the second part of this section, we show the simulation results of the backbone constructions, and the routing performance of the routing algorithm.

### 4.1 A routing scheme based on the separators

As one of the applications of the small separators of the grid graphs, we present a routing scheme for quasi-UDG based on the grid graph and analyze its performance. Our routing scheme is suitable for systems in which the size of the messages itself is relatively large. We will give the simulation results later in this section.

Our routing scheme is based on the distance labeling scheme described in [8]. The basic idea of distance labeling is to assign each vertex a label such that the distance between two vertices can be computed using only their labels. A straightforward labeling scheme would be to store in each node a full table of the distances to all the other nodes. The goal of the distance labeling scheme in [8] is to find the labels of minimum length. The separability of the underlying graph is a key factor of how good a distance labeling scheme is. In [8], the authors proved that, for a graph which has a separator of size $k$, there is a distance labeling scheme of label size $O(k\log n + \log^2 n)$, and such that the distance between two nodes can be computed in time $O(\log n)$, where $n$ is the number of nodes in the network.

Although a quasi-UDG $G$ may not possess a small separator, we have proved that the grid graph $H$ with $n$ vertices, constructed for $G$, does have a balanced separator of size $O(\sqrt{n})$. Conceptually, our routing protocol utilizes two-level routing: virtually, the message is sent in the grid graph from the cell containing the source to the cell containing the destination, via a shortest path in the grid graph; in reality, the routing is implemented in the underlying quasi-UDG to route from cell to cell. (Note that in each cell, the quasi-UDG vertices are fully connected, so routing from one cell to an adjacent one takes at most two hops.) The basic idea behind achieving shortest-path routing in the grid graph, is to split $H$ into two non-adjacent parts using the small separator $S$. Each vertex of $H$ remembers the distance to all vertices in $S$. Thus, two vertices in the two parts (or $S$) can compute their shortest-path distance using that information, because any shortest path between them must go through some vertices in $S$. We recursively apply the same process to partition each part into small parts, to enable any two vertices to compute their shortest-path distance using their stored information (their labels). We stop partitioning a part when its size is below a certain constant. (We call such a part a *basic block*.) Since we use balanced separators, the process ends after $O(\log n)$ levels of partitioning.

For a vertex $W$ of $H$, let $v(W)$ be the set of quasi-UDG nodes of $G$ that reside in the cell corresponding to $W$. The following list contains the information that each node $u \in v(W)$ in $G$ stores:

- The distances (in $H$) from $u$ to all the separator vertices of $H$ that are on the boundaries of all the partitions that $W$ is in.
- If $W$ is in a separator $S$, $u$ also stores the distances in $H$ from $W$ to all other vertices in $S$.
- The neighboring quasi-UDG nodes through which it can get to other cells adjacent to $W$ in $H$.
- A shortest-path routing table for the vertices of $H$ in the basic block where $W$ resides.

Part (1) and part (2) of the above list are then considered as the label $L(W)$ of the vertex $W$ in $H$. The label of $u$ is the triplet $(u, W, L(W))$.

The routing protocol assumes that the source knows the label of the destination. This piece of information can be obtained from location service. Since location service is not directly related to our topic, we skip the details here.

If the destination is not in the same cell as the source, the message will follow a shortest path in $H$ from the source cell to the destination cell. By utilizing the second part of the list, a node can send a message to any of its neighboring cells in two hops. Within a basic block, the third part of the routing table points out the shortest path between cells directly. Once the message reaches the basic block containing the destination, it can be relayed through the shortest path to the destination.

The routing protocol compares favorably with shortest-path routing algorithms and compact routing algorithms, for general networks, for its significantly-smaller routing table size and constant stretch factor.

In practice, there are two slightly different realization of this scheme: probing and non-probing. The one we described above is the probing case: each node will talk to node in its neighboring cells to find out which one of them is closer to the destination. Since the probing messages is of size $O(\log n)$ (only the destination cell ID and the node ID are required), and can be ignored by comparison to the size of the real message. The probing case, on the other hand, requires a little bit more space, but can avoid the probing messages. The node can also remember all the labels of the neighboring cells. Since there are constant many (at most $O\left(\frac{R^2}{r^2}\right)$) such cells, the order of the routing table will stay unchanged.

Now we present the proof of Theorem 4.

*Proof* (Theorem 4) In the routing protocol described above, the first part of a node's routing table is of size $O\left(\sqrt{N}\log N\right)$. The second and third parts of the routing table both consist of a constant number of entries, because the number of neighboring cells, and the number of cells in each basic block, are both constants. The size of the routing table is then $O\left(\sqrt{N}\log N\right)$. Inside each message we need only to carry the label of the destination vertex; therefore, the overhead in the message size is also bounded by $O\left(\sqrt{N}\log N\right)$.

Given a path $p$ from $u$ to $v$, let $d(p)$ denote its number of hops, and let $c(p)$ denote the number of times the path $p$ travels from one cell to another. Let $p_{opt}$ be the shortest path from $u$ to $v$, and let $p'$ be the routing path of our protocol. Clearly, $c(p_{opt}) \leq d(p_{opt})$, and $c(p') \leq c(p_{opt})$ because our protocol uses shortest path routing in the grid graph. The path $p'$ travels from one cell to the next in at most two hops, so $d(p') \leq 2c(p') + 1$. It follows that $d(p') \leq 2d(p_{opt}) + 1$. □

Sometimes we are more concerned about the energy consumption than the hop distance if the wireless nodes are able to adjust their communication range to save power. Let the communication cost be as defined before.

In reality, it is infeasible for a node to reduce its communication range to an arbitrarily-small number. There is always a constant range $\delta$ below which the wireless node cannot reduce its communication range. With this assumption, we prove the following theorem.

**Theorem 8** *Assume that the minimum communication range is $\delta$. (Therefore, the communication cost of an edge of Euclidean length $d$ is $\alpha \cdot (\max\{d, \delta\})^{\beta}$.) Then the communication cost of a routing path from a node $u$ to a node $v$ generated by the above routing protocol is upper bounded by a constant times the minimum communication cost from $u$ to $v$.*

*Proof* Let $p_{opt}$ be the optimal path from $u$ to $v$ with the minimum communication cost $C_{opt}$, and let $p'$ be the routing path of our algorithm with cost $C'$. If $u$, $v$ are in the same cell of the grid graph $H$, then $C_{opt} \geq \alpha \cdot \delta^{\beta}$, and $C' \leq \alpha \cdot r^{\beta}$, since vertices in the same cell form a clique. Therefore, $C' \leq (r^{\beta}/\delta^{\beta}) \cdot \alpha \cdot \delta^{\beta} \leq (r^{\beta}/\delta^{\beta}) \cdot C_{opt} = C_{opt} \cdot O(1)$.

Now assume that $u$, $v$ are in different cells of $H$. Let $l_{opt}$ and $l'$ denote, respectively, the number of hops in $p_{opt}$ and $p'$. By Theorem 4, $l' \leq 2l_{opt} + 1$. So $C' \leq l' \cdot \alpha \cdot R^{\beta} \leq (2l_{opt} + 1) \cdot \alpha \cdot R^{\beta} \leq \frac{2l_{opt}+1}{l_{opt}} \cdot \frac{R^{\beta}}{\delta^{\beta}} \cdot l_{opt} \cdot \alpha \cdot \delta^{\beta} \leq 3 \cdot \frac{R^{\beta}}{\delta^{\beta}} \cdot C_{opt} = C_{opt} \cdot O(1)$.

## 4.2 Simulations

We conduct extensive simulations based on the following quasi-UDG model: when the distance $x$ between two nodes $u$ and $v$ is such that $r \leq x \leq R$, then with probability

$$1 - \frac{x - r}{R - r}$$

there will be a direct link between $u$ and $v$.

The performance of our backbone construction algorithm and routing protocol are evaluated for sensor networks based on different configurations of the quasi-UDG model.

The performance has been stable and consistent. In the following experiment, we randomly deploy $N$ quasi-UDG nodes in a 2-D space of size 15,000 $\times$ 15,000. We increase the number of nodes, $N$, in the system from 1,000, 1,500 to 2,000 to verify the effects of density change on the performance. We range the value of $R/r$ from 1, 2, 3, ..., 10 to see the performance of our algorithms for different wireless connectivity models. To mimic nontrivial network topologies, we randomly generate holes of radius randomly picked in the range $[R, 2R]$ in the sensor field. The number of holes ranges from 0, 1, 2, 4. The average degree of our networks ranges from 6, 7, ..., 10. For a given value of $R/r$, we adjust the absolute values of $R$ and $r$ to achieve the expected average degree. For each configuration, we run the simulation 50 times and take the average of the performance metrics.

We would like to point out that the performances of our routing algorithm and backbone construction are relatively independent of the size of the network. Both our theoretical bounds and the simulation results show that the quality of the backbone constructed and the stretch of the routing paths are closely related to the ratio of $r$ to $R$.

### *4.2.1 Backbone construction*

In the backbone construction simulations, we measure the power stretch factor, maximum degree, the average degree, and the average number of edge-crossings in the backbone constructed, and compare them to those of the original graph.

The results are shown in Fig. 7, and are for backbones constructed by only performing the first step and the last step in Algorithm **Backbone** on sensor networks with four holes.

Our results show that, for all configurations, the backbones have very small power stretch factors, and much smaller maximum degree than the original network (in most cases, we can bring the average degree to below 6, which is the upper bound of the average degree for planar graphs). Even when $R/r = 10$, the average degree of our backbones is no more than 5. As for the number of

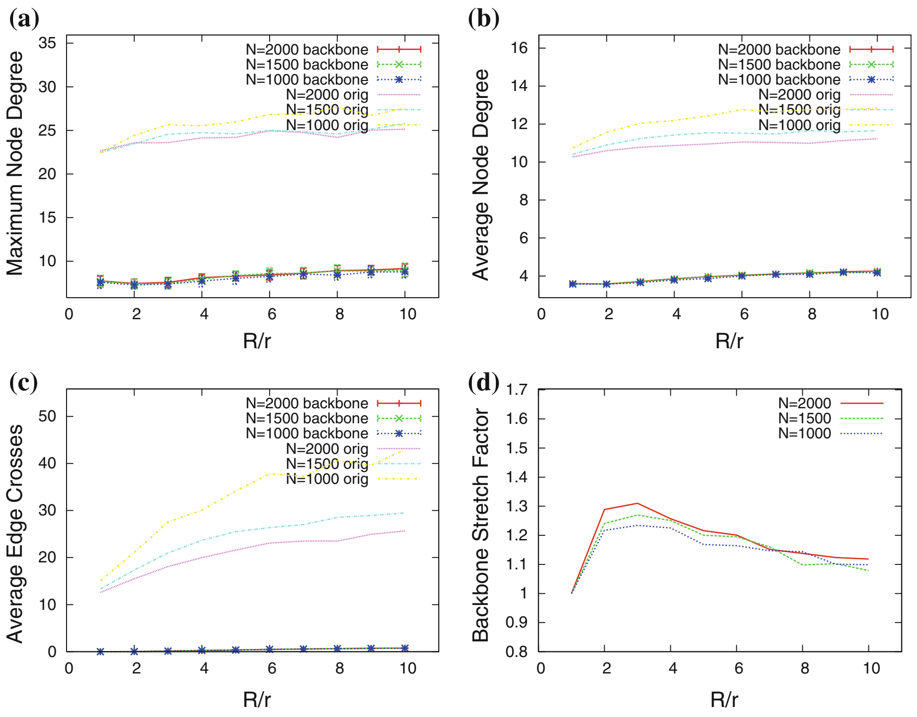


**Fig. 7** Statistics (average value and its standard deviation) of the backbones constructed for sensor networks with four holes, compared to that of the original networks: **a** maximum degree; **b** average degree; **c** average number of edge crossings; **d** power stretch factor

edge-crossings, our algorithm reduced it by at least 80% in all cases.

Figure 7(a) shows the maximum node degree for the backbones compared to that of the original networks. In all cases, the maximum node degree is below 10, which is much smaller than that of the original network. Similarly, Fig. 7(b) shows that the average degrees of the backbones are all below 5. In Fig. 7(c), we see that the average number of edge-crossings in the backbone is very close to 0: the backbones we constructed are very close to planar graphs. Figure 7(d) shows the power stretch factor of our backbones. We notice that when *R*/*r* gets larger, the power stretch factor becomes smaller. This is because in our backbone constructions, we tend to remove less edges if *R* is much larger than *r*, hence causing less distortion to the stretch factors.

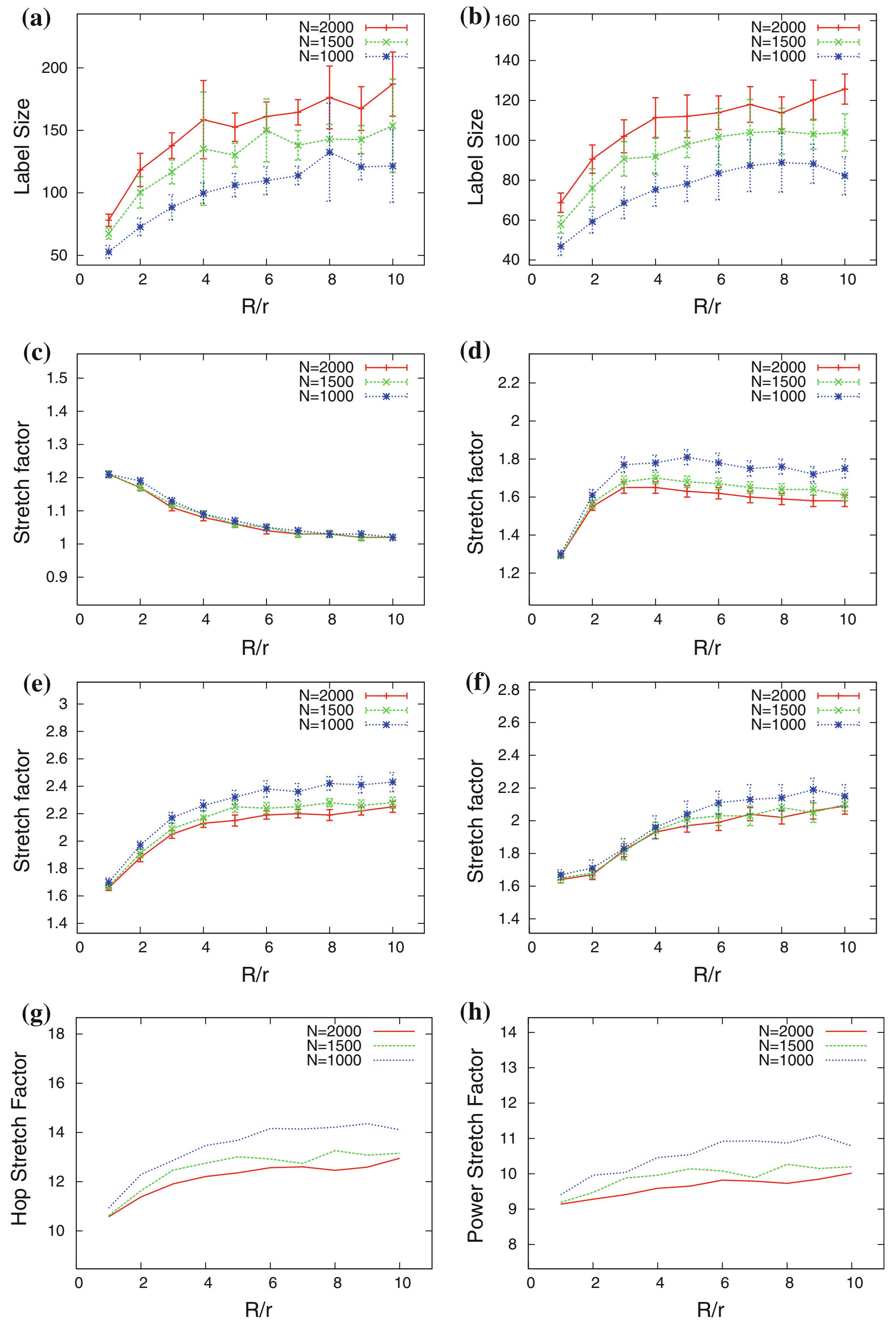


**Fig. 8** Routing performance in networks with four holes: **a** average label sizes and their standard deviations based on the original network; **b** average label sizes and their standard deviations on the backbone; **c** hop stretch factors for compacting routing on the original network; **d** hop stretch factors for compacting routing on the backbone; **e** power stretch factors for compacting routing on the original network; **f** power stretch factors for compacting routing on the backbone; **g** hop stretch factors for greedy forwarding + local flooding routing on the original network; **h** power stretch factors for greedy forwarding + local flooding routing on the backbone

### 4.2.2 Routing performance

We apply our routing protocol not only to the original quasi-UDGs, but also to the backbones we obtain. To study the performance, we measure the average label size and the stretch factor of routing path. The length of the routing path in the original graphs is defined as the hop distance between two nodes, while in the backbones, we use the communication cost with $\beta = 2$ as the length of the path. In both cases we randomly pick 1,000 source-destination pairs in the graph, simulate the routing process, and compare the length of the path with the shortest.

Figure 8(a) shows the average values of the average label size (with a node ID as a unit), along with their standard deviations, over the experiments for two cases. Figure 8(b) shows the statistics for the label sizes on routing base on the backbones. We observe that the label sizes, with the algorithm applied to the backbones, are smaller than those of the original graphs. This is mainly because the backbones are sparser than the original quasi-UDGs, and hence the grid graphs we get are also sparser and have smaller separators. We will see later that this advantage comes at a cost of slightly power larger stretch factors.

Figure 8(c) shows the average hop distance stretch factors of the routing path for the routing algorithm applied to the original graphs directly. In all cases, the path stretch factor is not larger than 1.3.

Figure 8(d) shows the hop distance stretch factors of the routing paths when the algorithm is applied to the backbones. The hop stretch factors shown in Fig. 8(d) are moderately larger than the ones shown in Fig. 8(c). It is the price we paid for the reduction in the size of the routing tables.

From Fig. 8(c) and (d), it looks interesting that when $R/r$ is large (10), the algorithm generally gets better. This is because to maintain the same average node degree of the graphs we have to decrease the value of $r$. In that case a grid graph actually describes the original graph more accurately, and with more details. Hence, the sizes of the labels are larger [see Fig. 8(a), (b)], but the paths we discovered are closer to the shortest paths.

Figure 8(e) shows the power stretch factors of the routing paths when applying our routing algorithm directly on the original network, while Fig. 8(f) shows the power stretch factors of the routing paths when applying the routing algorithm on the backbones.

We have also implemented the well-known greedy-forwarding plus local-flooding routing algorithm, and performed the same number of experiments on the same set of graphs. The average stretch factors are shown in Fig. 8(g) and (h). Our results indicate that, when compared to that algorithm, the routing protocol based on separators has a much better stretch factor because of its robustness to the existence of holes.

## 5 Conclusion

In this paper, we have studied two structural properties of quasi-UDGs: separability and the existence of power efficient spanners. These results lead to a deeper understanding of the local properties of quasi-UDG networks, and to an improvement in the development of networking protocols.

## References

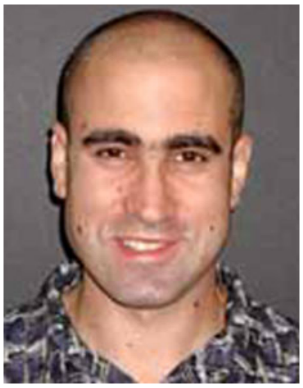
1. Alzoubi, K., Li, X., Wang, Y., Wan, P., & Frieder, O. (2003). Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems, 14*(4), 408–421.
2. Barrière, L., Fraigniaud, P., & Narayanan, L. (2001). Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *Proceedings of the 5th international workshop on discrete algorithms and methods for mobile computing and communications (DialM '01)* (pp. 19–27).
3. Bose, P., Morin, P., Stojmenovic, I., & Urrutia, J. (1999). Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on discrete algorithms and methods for mobile computing and communications (DialM '99)* (pp. 48–55).
4. Damian, M., Pandit, S., & Pemmaraju, S. (2006). Local approximation schemes for topology control. In *Proceedings of PODC* (pp. 208–217).
5. Fang, Q., Gao, J., & Guibas, L. J. (2006). Landmark-based information storage and retrieval in sensor networks. *Proceedings of INFOCOM'06*.
6. Gabriel, K., & Sokal, R. (1969). A new statistical approach to geographic variation analysis. *Systematic Zoology, 18*, 259–278.
7. Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., & Wicker, S. (2002). *Complex behavior at scale: an experimental study of low-power wireless sensor networks*. Technical Report UCLA/CSD-TR 02-0013, UCLA.
8. Gavoille, C., Peleg, D., Pèrennes, S., & Raz, R. (2004). Distance labeling in graphs. *Journal of Algorithms, 53*(1), 85–112.
9. Kanj, I. A., & Perkovic, L. Improved stretch factor for bounded-degree planar power spanners of wireless ad-hoc networks. In *To appear in the proceedings of ALGOSENSOR'06*.
10. Karp, B., & Kung, H. T. (2000). GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on mobile computing and networking* (pp. 243–254).
11. Kim, Y., Govindan, R., Karp, B., & Shenker, S. (2005, May). Geographic routing made practical. In *Proceedings of the 2nd USENIX/ACM Symposium on Networked System Design and Implementation (NSDI 2005), Boston, MA*.
12. Lillis, K. M., Pemmaraju, S. V., & Pirwani, I. A. (2007). Topology control and geographic routing in realistic wireless networks. In *ADHOC-NOW* (pp. 15–31).
13. Lipton, R. J., & Tarjan, R. E. (1979). A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics, 36*(2), 177–189.
14. Kuhn, F., & Zollinger, A. (2003). Ad-hoc networks beyond unit disk graphs. In *Proceedings of 2003 joint workshop on foundations of mobile computing* (pp. 69–78).

15. Kuhn, F., Watternhofer, R., Zhang, Y., & Zollinger, A. (2003). Geometric ad-hoc routing: Of theory and practice. In *Proceedings of PODC* (pp. 63–72).
16. Sohrabi, K., Manriquez, B., & Pottie, G. (1999). Near ground wideband channel measurement. *IEEE Vehicular Technology Conference, 1*, 571–574.
17. Wattenhofer, R. (2006). Sensor networks: Distributed algorithms reloaded—or revolutions?. In *Proceedings of SIROCCO* (pp. 24–28).
18. Yao, A. C.-C. (1982). On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing, 11*(4), 721–736.

## Author Biographies

**Jianer Chen** received Ph.D. degree in Computer Science from Courant Institute of Mathematical Sciences, New York University, 1987, and Ph.D. degree in Mathematics from Columbia University, 1990. He is currently a Professor in Computer Science and Engineering at Texas A&M University. His research interests include algorithms and complexity, networks optimization, computer graphics, and bioinformatics.

**Anxiao (Andrew) Jiang** received the B.S. degree in Electronic Engineering from Tsinghua University in 1999, and the M.S. and Ph.D. degrees in Electrical Engineering from the California Institute of Technology in 2000 and 2004, respectively. He is currently an Assistant Professor in the Computer Science and Engineering Department at Texas A&M University. He is a recipient of the NSF CAREER Award in 2008 for his research on information theory for flash memories. His research interests include information theory, data storage, networks and algorithm design.



**Iyad A. Kanj** obtained his Ph.D. from Texas A&M University in 2001. Since then he has been with DePaul University where he is currently an associate professor of Computer Science. His research interests are in the areas of parameterized complexity, graph theory and algorithms, combinatorial optimization, computational geometry, distributed algorithms and wireless computing.

**Ge Xia** received his Ph.D. in computer science from Texas A&M University in 2005. Since then, he has been with the Department of Computer Science at Lafayette College, where he is currently an Assistant Professor. His research interests include computational complexity and optimization, graph theory and algorithms, wireless networks, and computational biology.

**Fenghui Zhang** is a software engineer at Google. He received his Ph.D. from the Department of Computer Science in Texas A&M University in 2008. His advisor is Dr. Jianer Chen. He received his B.S. in Mathematics from Fudan University in 1997.