Selecting Forwarding Neighbors in Wireless Ad Hoc Networks

Gruia Călinescu* Ion Măndoiu† Peng-Jun Wan‡ Alexander Z. Zelikovsky§

Abstract

Broadcasting is a fundamental operation which is frequent in wireless ad hoc networks. A simple broadcasting mechanism, known as *flooding*, is to let every node retransmit the message to all its 1-hop neighbors when receiving the first copy of the message. Despite its simplicity, flooding is very inefficient and can result in high redundancy, contention, and collision. One approach to reducing the redundancy is to let each node forward the message only to a small subset of 1-hop neighbors that cover all of the node's 2-hop neighbors. In this paper, we propose two practical heuristics for selecting the minimum number of forwarding neighbors: an $O(n \log n)$ time algorithm that selects at most 6 times more forwarding neighbors than the optimum, and an $O(n \log^2 n)$ time algorithm with an improved approximation ratio of 3, where n is the number of 1- and 2-hop neighbors. The best previously known algorithm, due to Bronnimann and Goodrich [2], guarantees $O(1)$ approximation in $O(n^3 \log n)$ time.

1 Introduction

Wireless ad hoc networks can be flexibly and quickly deployed for many applications such as automated battlefield, search and rescue, and disaster relief. Unlike wired networks or cellular networks, no wired backbone infrastructure is installed in wireless ad hoc networks. A communication session is achieved either through a single-hop radio transmission if the communication parties are close enough, or through relaying by intermediate nodes otherwise. In

*Computer Science Department, Illinois Institute of Technology, Chicago, IL 60616, calinesc@cs.iit.edu. Research partially performed while visiting the Department of Combinatorics and Optimization at University of Waterloo, where supported by NSERC research grant.

†Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114, mandoiu@cs.ucsd.edu.

‡Computer Science Department, Illinois Institute of Technology, Chicago, IL 60616, wan@cs.iit.edu.

§Department of Computer Science, Georgia State University, University Plaza, Atlanta, Georgia 30303, alexz@cs.gsu.edu. Supported in part by NSF Grant CCR-9988331 and Award No. MM2-3018 of the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF).

this paper, we assume that all nodes in a wireless ad hoc network are distributed in a two-dimensional plane and have an equal maximum transmission range of one unit.

Broadcasting is a fundamental networking operation in wireless ad hoc networks. It is widely and frequently performed in many networking tasks such as paging a particular host, sending an alarm signal, and finding a route to a particular host [1][9][17]. A simple broadcasting mechanism, known as *flooding*, is to let every node retransmit the message to all its 1-hop neighbors when receiving the first copy of the message. Despite its simplicity, flooding has a serious drawback, known as the *broadcast storm* [16]. First, because the radio propagation is omnidirectional and a physical location may be covered by the transmission ranges of several nodes, many retransmissions are redundant. Second, heavy contention could exist because retransmitting nodes are probably close to each other. Third, collisions are more likely to occur because the RTS/CTS dialogue is inapplicable and the timing of retransmissions is highly correlated.

The following simple technique was recently exploited [13] (see also [12]) and [18] to reduce redundant retransmissions: By virtue of beaconing, each node maintains a local topology of its 2-hop neighborhood, and relays the message only to a small subset of 1-hop neighbors which cover (in terms of radio range) all nodes that are two hops away. The subset of 1-hop neighbors selected by each node is referred to as *forwarding set* [18] or *multipoint relaying set* [13] (see also [12]). In this paper we consider the problem of finding a forwarding set of minimum size.

Minimum Forwarding Set Problem: Given a source A, let $\mathcal{D}$ and $\mathcal{P}$ be the sets of 1- and 2-hop neighbors of A. Find a minimum-size subset $\mathcal{F}$ of $\mathcal{D}$ such that every node in $\mathcal{P}$ is within the coverage area of at least one node from $\mathcal{F}$.

1.1 Previous work

Laouiti et al. [13] (see also Jacquet et al. [12]) and Sinha et al. [18] considered the Minimum Forwarding Set problem assuming no knowledge of the geographic location of the nodes. In this case, the Minimum Forwarding Set problem is essentially the well-studied Set Cover problem. Not surprisingly, the heuristic proposed in [13] (see also [12]) is a translation of Chvátal's greedy algorithm [4] for Set Cover, and thus guarantees an approximation factor of $O(\log m)$, where m is the maximum neighborhood size. The greedy algorithm iteratively selects a 1-hop neighbor covering the maximum number of 2-hop neighbors not yet covered, and terminates when all 2-hop neighbors have been covered. The greedy algorithm does not take into account the geometric properties of the Minimum Forwarding Set problem, and in fact Figure 1 shows a family of instances for which the size of the solution found by the greedy algorithm is larger than the optimum by a logarithmic factor.

Under the assumption that the nodes in the wireless network are distributed in a two-dimensional plane and each node has unit transmission range, the topology of the network is modeled as a *unit-disk graph* [5]. In this graph, there

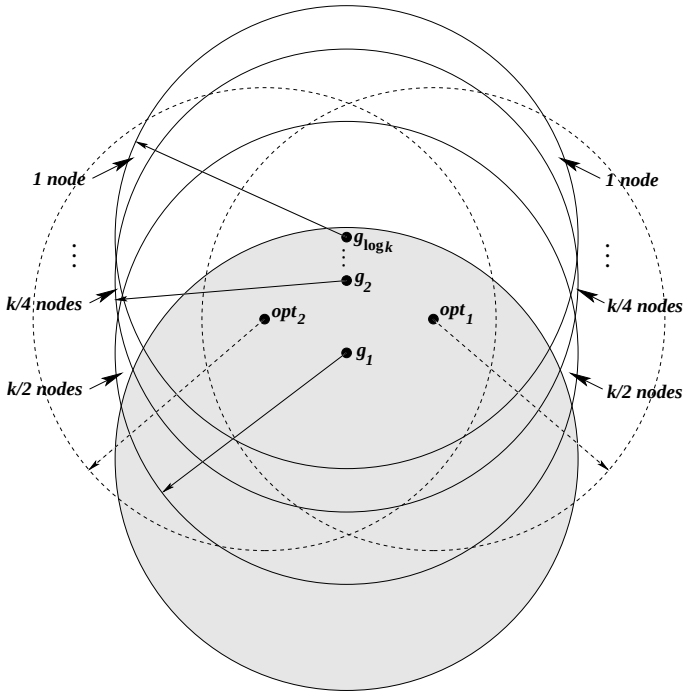

Figure 1: Instance for which the size of the solution computed by the greedy algorithm, $\{g_1, \ldots, g_{\log k}\}$, is larger than the optimum solution, $\{opt_1, opt_2\}$, by a logarithmic factor.

is an edge between two nodes if and only if their distance is at most one. The Minimum Forwarding Set problem for a given source node s asks for a minimum size set of 1-hop neighbors of s *dominating* 2-hop neighbors of s in the unit-disk graph. The related Dominating Set problem in unit-disk graphs [5] asks for a subset of nodes dominating (i.e., adjacent to) all the other nodes. The Dominating Set problem in unit-disk graphs is NP-hard [5] but admits a PTAS [11]. The Minimum Forwarding Set problem does not reduce to the Dominating Set problem in unit-disk graphs since dominators are restricted to the set of 1-hop neighbors.

The Minimum Forwarding Set problem is also related to the Unit-Disk Cover problem [10], which asks for the minimum number of unit disks covering a given set of points in the plane. The Unit-Disk Cover problem is also NP-hard [5] and admits a PTAS [10]. Since in the Unit-Disk Cover problem disk centers can be chosen arbitrarily in the plane, the algorithms for this problem do not apply to the Minimum Forwarding Set problem where disks must be centered at 1-hop neighbors only.

The Minimum Forwarding Set problem is a special case of the NP-Hard Disk Cover problem [2], which asks for a minimum size subset of a given set of disks covering a given set of points. The complexity of Minimum Forwarding Set problems is not known. A constant-ratio approximation algorithm for Disk Cover, and therefore also for Minimum Forwarding Set, was given by Bronnimann and Goodrich [2] However, their algorithm – which is a special case

of a sophisticated algorithm for spaces with bounded VC-dimension – has impractical running-time and its proven approximation ratio is a very large constant.

1.2 Our contributions

- A 6-approximation algorithm for the Minimum Forwarding Set problem running in $O(n \log n)$ time, where n is the total number of 1- and 2-hop neighbors.
- A 3-approximation algorithm for the Minimum Forwarding Set problem running in $O(n \log^2 n)$ time.
- An exact $O(n \log^2 n)$ time, and a 2-approximation $O(n \log n)$ time algorithm for the special case of the Minimum Forwarding Set problem when all 2-hop neighbors are in the same quadrant with respect to the source node.
- A constant-factor approximation for the Minimum Disk Cover problem with disks of the same radius, based on rounding the optimal solution of a linear programming relaxation.

A preliminary version of this paper [3] presents the same results, except for the 3-approximation algorithm and the exact algorithm for the special case above, where the running time in [3] is $O(n^2)$.

The paper is organized as follows. In next section we reformulate the Minimum Forwarding Set problem in geometric terms, give a high-level algorithm based on decomposition into quadrants, and establish basic geometric properties of the partitioned sets of 1- and 2-hop neighbors. The next three sections deals with covering 2-hop neighbors in a quadrant. We first describe an $O(n \log n)$ 2-approximation algorithm (Section 3), then we give an $O(n^2)$ exact algorithm (Section 4) and finally describe details of data structures needed to obtain the $O(n \log^2 n)$ implementation of the exact algorithm (Section 5). In Section 6 we give an extension of our techniques to the Disk Cover problem of [2], and conclude in Section 7.

2 Partition based algorithm

Throughout this paper a *unit disk*, or just *disk* for short, refers to a closed disk of radius 1. The boundary of a region R of the Euclidean plane is denoted by ∂R, e.g., the boundary circle of a disk D is denoted by ∂D. Under the assumption that each network node has unit transmission range, we reformulate the Minimum Forwarding Set problem as follows.
1-Hop Disk Cover Problem: Given a unit-disk A, a set $\mathcal{D}$ of unit disks centered inside A, and a set of points $\mathcal{P}$ outside A such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{D}\}$, find a minimum-size subset $\mathcal{F}$ of $\mathcal{D}$ such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{F}\}$.

Algorithm 1: 1-Hop Disk Cover

Input: Unit-disk A, set of unit disks $\mathcal{D}$ centered inside A, set of points $\mathcal{P}$ outside A such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{D}\}$

Output: Subset $\mathcal{F} \subseteq \mathcal{D}$ such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{F}\}$

---

1. Partition the exterior of A into four quadrants Q_1–Q_4 by two orthogonal lines, not containing points in $\mathcal{P}$, through the center of A (see Figure 2).
2. For $q = 1, \ldots, 4$, compute a disk cover, $\mathcal{F}_q$, for the points in $\mathcal{P} \cap Q_q$.
3. Output $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$.

Our high-level algorithm (Algorithm 1) partitions the points of $\mathcal{P}$ according to the four quadrants defined by two orthogonal lines through the center of A, and then independently solves the 1-Hop Disk Cover problem for each quadrant. The union of these four disk covers is then a disk cover for all the points in $\mathcal{P}$. As usual, the approximation ratio of an algorithm $\mathcal{A}$ for a minimization problem Π is the supremum, over all instances of Π, of the ratio between the output value of $\mathcal{A}$ and the optimal value. The following theorem relates the approximation ratio of Algorithm 1 to the approximation ratio that can be guaranteed for the 1-Hop Disk Cover restricted to points in a single quadrant.

Theorem 1 *If disk covers $\mathcal{F}_q$ computed in Step 2 are within a factor of α of optimum, then Algorithm 1 has an approximation ratio of at most 3α for the 1-Hop Disk Cover problem.*

Proof. Let *OPT* be the optimal set of disks, and denote by OPT_q, $q = 1, 2, 3, 4$, the subset of disks in *OPT* having centers in the q^{th} sector of disk A. The key observation is that points in the quadrant Q_q cannot be covered by disks in $OPT_{q+2 (\text{mod } 4)}$. Therefore, points in $\mathcal{P} \cap Q_1$ must be covered by disks in $OPT_4 \cup OPT_1 \cup OPT_2$, and thus, by the assumption that $\mathcal{F}_q$'s are within a factor of α of the respective optimum solutions,

$$|\mathcal{F}_1| \leq \alpha(|OPT_4| + |OPT_1| + |OPT_2|).$$

Similarly,

$$|\mathcal{F}_2| \leq \alpha(|OPT_1| + |OPT_2| + |OPT_3|),$$
$$|\mathcal{F}_3| \leq \alpha(|OPT_2| + |OPT_3| + |OPT_4|),$$
$$|\mathcal{F}_4| \leq \alpha(|OPT_3| + |OPT_4| + |OPT_1|).$$

Thus, the output of the algorithm has size

$$|\mathcal{F}_1| + |\mathcal{F}_2| + |\mathcal{F}_3| + |\mathcal{F}_4|$$

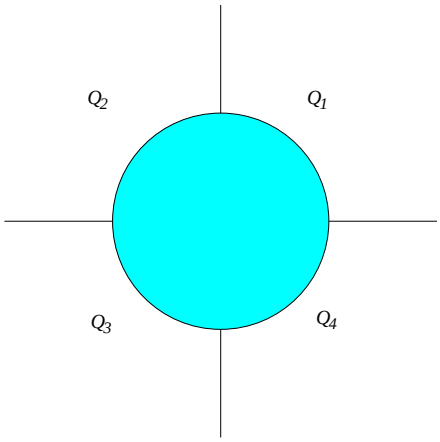

Figure 2: The four quadrants in Algorithm 1.

$$\begin{aligned} &\leq 3\alpha(|OPT_1| + |OPT_2| + |OPT_3| + |OPT_4|) \\ &= 3\alpha|OPT|. \end{aligned}$$

■

We will show that $\alpha = 2$ can be achieved in $O(n \log n)$ time (see Section 3), and $\alpha = 1$ can be achieved in $O(n \log^2 n)$ time (see Section 4). Hence, Algorithm 1 achieves an approximation factor of 6, respectively 3, within the same time bounds. It is natural to ask if these approximation ratios can be improved by partitioning the set of points according to $k < 4$ equal sectors defined by half-lines starting at the center of A. The proof of Theorem 1 can be generalized to show that partitioning into k sectors gives an approximation ratio of $(\lceil k/2 \rceil + 1)\alpha$ for the 1-Hop Disk Cover problem if the disk cover for each sector is approximated within a factor of α. Thus, using decomposition into 3 equal sectors does not lead to an approximation ratio better than that obtained by decomposition into quadrants. Improvements using decomposition into 2 equal sectors are possible provided that we can find an algorithm for covering the points in a $180°$ sector with an approximation ratio of less than $3/2$. The ideas used in Section 4 to solve exactly the problem for a quadrant do not extend to $180°$ sectors, since these lack the second of the essential topological properties established for the quadrants in the following lemma.

Lemma 2 *Let Q be an exterior quadrant of A, $J = \partial D$ be its border, and $\mathcal{D}$ be a set of disks intersecting the interior of Q. Then:*

(a) For any disk $D \in \mathcal{D}$, $|\partial D \cap J| = 2$.

(b) For any two disks $D, D' \in \mathcal{D}$, $|\partial D \cap \partial D' \cap Q| \leq 1$.

(c) No two disks in $\mathcal{D}$ are tangent in Q.

Proof. Without loss of generality, we may assume that the unit-disk A is centered at the origin and that Q is defined

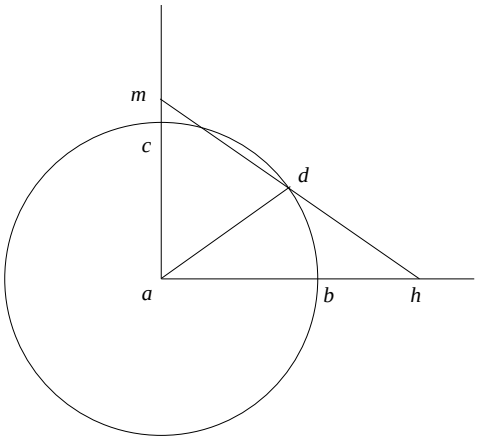

Figure 3: The extreme configuration in the proof of Lemma 2(b).

by the positive x- and y-axes. Then, the boundary of the quadrant Q, J, consists of the two half-lines from $(0,1)$ to $(0,\infty)$, and from $(1,0)$ to $(1,\infty)$, together with a quarter-circle of ∂A. Let a, b, c be the points with coordinates $(0,0)$, $(1,0)$, and $(0,1)$, respectively. We will use $\widehat{bc}$ to denote the quarter-circle of A enclosed in J.

(a) Since every $D \in \mathcal{D}$ has non-empty intersection with the interior of Q, every circle ∂D has at least two intersection points with J. The closed simple Jordan curve ∂D and the infinite simple Jordan curve J must intersect an even number of times (unless they are tangent, but this cannot happen), and thus cannot intersect three times. Thus, to complete the proof of part **(a)** we need to show that ∂D does not intersect J four or more times.

Let d denote the center of disk D. Then $0 < |da| \leq 1$, since d is inside A. Note that ∂D can intersect the x-axis in at most two points, of which only one can have x-coordinate bigger than 1. Similarly, D can intersect the y-axis in at most two points, of which only one can have y-coordinate bigger than 1. Furthermore, D intersects $\widehat{bc}$ at most once. Indeed, when two unit-circles with centers within distance of at most 1 intersect, the two intersection points are at least $2\pi/3$ apart on each of the circles, and hence a quarter-circle may contain only one of them.

(b) Assume, for a contradiction, that D and D' are two distinct disks in $\mathcal{D}$ that intersect at points h and l, with both h and l in $Q \cup J$. Let d and d' be the centers of D and D', respectively. We will change the configuration a bit, to obtain a more extreme case. First, translate d, d', h, and l to the right until d or d' hits $\widehat{bc}$, and assume, by symmetry, that d is on $\widehat{bc}$. We still have $h, l \in R \cup J$. Assume also that h is to the right of the point l. Now start rotating the rhombus $hd'ld$ clockwise around d until h hits either the x-axis or $\widehat{bc}$, whichever happens first (see Figure 3). This procedure also keeps d' inside the unit-disk A and l in Q. Let m be the point where the line hd intersects the y-axis. As $|d'h| = |dh| = 1$, d' must lie in the same side of the line hm as a. As the angle $\widehat{ham} \leq \frac{\pi}{2}$, m must be within the diameter of the unit-disk centered at d that contains h. Therefore $|dm| \leq 1$. Thus l, which is in Q, must be outside the triangle ahm, and consequently d' must be on the other side of the line hm than a, which is a contradiction.

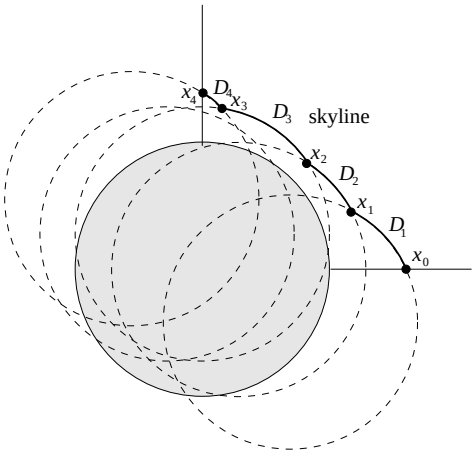

Figure 4: The skyline of a set of disks in a quadrant.

(c) Let D and D' be two disks from $\mathcal{D}$. Then ∂D and $\partial D'$ cannot be tangent from the interior since they have the same radius. If ∂D and ∂D are tangent from the exterior, then the distance between their centers is 2, and the common point can only be the origin a, which is not in Q. ■

3 Fast geometric disk covering in a quadrant

In this section we give a fast 2-approximation algorithm for the 1-Hop Disk Cover problem with all points of $\mathcal{P}$ coming from an exterior quadrant Q of unit disk A.

The *skyline* $S = (x_0, x_1, \ldots, x_k)$ of $\mathcal{D}$ is the upper envelope of $Q \cap (\cup\{D \in \mathcal{D}\} \cup A)$ (see Figure 4). The skyline consists of arcs $\overset{\frown}{x_{i-1}x_i}$ on the border of disks $D_i \in \mathcal{D} \cup \{A\}$, $i = 1, \ldots, k$, such that $x_0 \in \partial Q \cap \partial D_1$, $x_i \in \partial D_{i-1} \cap \partial D_i$ ($i = 1, \ldots, k-1$), and $x_k \in \partial D_k \cap \partial Q$. The algorithm (Algorithm 4) starts by computing the skyline S with x_i's numbered in counter-clockwise order, i.e., with polar coordinates (ρ_i, r_i) of points x_i satisfying $\rho_0 \leq \rho_1 \leq \rho_2 \leq \cdots \leq \rho_k$. As established in Lemma 8 below, the skyline disks D_i covering a point $p \in \mathcal{P}$ form an interval in the sequence $D_1, \ldots, D_k$. The algorithm computes these intervals for each point of $\mathcal{P}$, them outputs a minimum size set $\mathcal{F}$ of skyline disks D_i hitting all intervals. Clearly, the hitting set $\mathcal{F}$ computed by Algorithm 4 is a disk cover for the points in $\mathcal{P}$. Furthermore, we have:

Theorem 3 *Algorithm 4 runs in $O(n \log n)$ time, and has an approximation ratio of 2 for the 1-Hop Disk Cover problem in a quadrant.*

Theorems 1 and 3 immediately give:

Algorithm 2: Geometric 1-Hop Disk Covering in a quadrant

Input: Unit-disk A, set of unit disks $\mathcal{D}$ centered inside A, set of points $\mathcal{P}$ in the exterior quadrant Q of A such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{D}\}$

Output: A subset $\mathcal{F} \subseteq \mathcal{D}$ such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{F}\}$

---

1. Find the skyline $S = (x_0, x_1, x_2, \ldots, x_k)$ of $\mathcal{D}$, where the polar coordinates of x_i are (ρ_i, r_i) and $\rho_0 \leq \rho_1 \leq \rho_2 \leq \cdots \leq \rho_k$. Let D_i be the disk containing arc $\overset{\frown}{x_{i-1}x_i}$.
2. For each $p \in \mathcal{P}$ with polar coordinates (ρ, r), find the interval $[D_{first(p)}, D_{last(p)}]$ of skyline disks D_i that cover p, via three binary searches:
   (a) find $i \in \{1, \ldots, k\}$, such that $\rho \in [\rho_{i-1}, \rho_i]$
   (b) $first(p) \leftarrow \min\{j : 1 \leq j \leq i, p \in D_j\}$
   (c) $last(p) \leftarrow \max\{j : i \leq j \leq k, p \in D_j\}$
3. Find the minimum set $\mathcal{F}$ of skyline disks hitting each interval $[D_{first(p)}, D_{last(p)}]$, $p \in \mathcal{P}$, using the following *Interval Hitting Algorithm*:
   (a) Sort the set of all intervals $I = \{[D_{first(p)}, D_{last(p)}], p \in \mathcal{P}\}$ in ascending order of their right end, $D_{last(p)}$
   (b) $\mathcal{F} \leftarrow \emptyset$
   (c) While $I \neq \emptyset$ do
      Add to $\mathcal{F}$ the right end disk $D_{last(p)}$ of the first interval
      Remove from I all intervals hit by $D_{last(p)}$
4. Output $\mathcal{F}$

Corollary 4 *Combined with Algorithm 4, Algorithm 1 runs in $O(n \log n)$ time and has an approximation ratio of 6 for the Minimum Forwarding Set problem.*

The rest of the section is devoted to the proof of Theorem 3.

Lemma 5 *A point $q \in Q$ belongs to a disk $D \in \mathcal{D}$ if and only if the half-line L from the center a of A through a point q intersects $\partial D \cap Q$ at a point q' such that q belongs to the segment $[a, q']$.*

Proof. Every disk $D \in \mathcal{D}$ contains a. Thus, the segment $[a, q']$ is fully contained in D, and every point of L outside of this segment is in the exterior of D. ■

Lemma 6 *If point $p \in \mathcal{P}$ has polar coordinates (ρ, r) such that $\rho \in [\rho_{i-1}, \rho_i]$, then $p \in D_i$.*

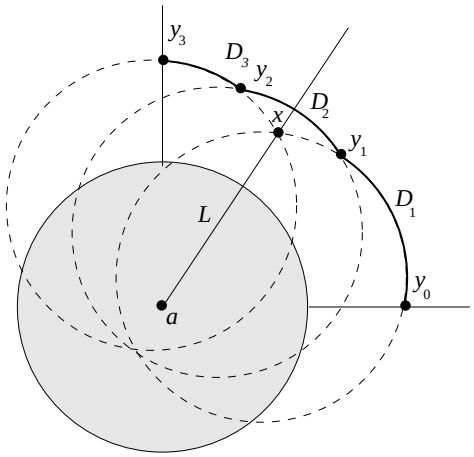

Figure 5: The skyline of $\{D_1, D_2, D_3\}$ in Lemma 7.

Proof. Follows immediately from Lemma 5. ■

Lemma 7 *Let D_1, D_2, D_3 be three disks of $\mathcal{D}$ appearing in this order in the skyline of $\{D_1, D_2, D_3\}$. Then $D_1 \cap D_3 \cap Q \subseteq D_2 \cap Q$.*

Proof. Assume that $D_1 \cap D_3 \cap Q \neq \emptyset$, and let $S' = (y_0, y_1, y_2, y_3)$ be the skyline of $\{D_1, D_2, D_3\}$ (see Figure 5). Since $y_1 = \partial D_1 \cap \partial D_2 \cap Q$, $y_2 = \partial D_2 \cap \partial D_3 \cap Q$, and $y_1, y_2 \notin D_1 \cap D_3$, Lemma 2(b) implies that $\partial D_2 \cap \partial(D_1 \cap D_3 \cap Q) = \emptyset$.

To complete the proof, it suffices to show that D_2 contains some point of $D_1 \cap D_2 \cap Q$. Let $x = \partial D_1 \cap \partial D_2 \cap Q$, and let L be the half-line from a through x. Since $a \in D_1$, L intersects ∂D_1 exactly once, at x. Thus, L does not intersect the arc $\overset{\frown}{y_0 y_1}$ of the skyline. Similarly, L does not intersect $\overset{\frown}{y_2 y_3}$. It follows that L intersects $\overset{\frown}{y_1 y_2}$, and, by Lemma 5, $x \in D_2$. ■

The following is a straightforward corollary of Lemma 7:

Lemma 8 *For every $p \in \mathcal{P}$, the skyline disks D_i covering p form an interval $[D_{first(p)}, D_{last(p)}]$ in the sequence $D_1, \ldots, D_k$.*

Lemma 9 *The optimum cover of $\mathcal{P}$ with disks from the set $\{D_1, \ldots, D_k\}$ of skyline disks contains at most 2 times more disks than the optimum cover of $\mathcal{P}$ with disks from $\mathcal{D}$.*

Proof. It suffices to prove that, for every $D \in \mathcal{D}$, $D \cap Q$ is covered by at most two skyline disks. Furthermore, since Lemma 5 implies that any set of disks covering $\partial D \cap Q$ fully covers $D \cap Q$, we only need to show that $\partial D \cap Q$ is covered by at most two skyline disks.

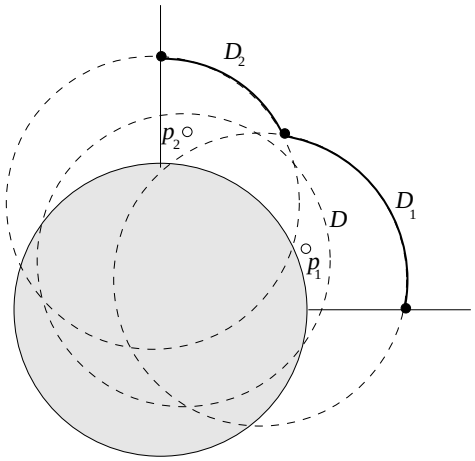

Figure 6: A tight example for the approximation ratio of Algorithm 4.

Let d_1 and d_2 be the two points of intersection of ∂D with the boundary of the central disk A. By Lemma 2(b), any skyline disk D_i intersecting $D \cap Q$ contains at least one of the points d_1 and d_2. The key observation is that, for any two skyline disks D_i and D_j both containing d_1 (or both containing d_2), the arc $\partial D \cap \partial D_i \cap Q$ is contained in the arc $\partial D \cap \partial D_j \cap Q$ or vice versa. Therefore the minimal set of skyline disks covering $\partial D \cap Q$ has at most two disks. ■

Proof of Theorem 3. The approximation ratio of Algorithm 4 follows from Lemma 9. Step 1 of the algorithm can be implemented in $O(n \log n)$ time using, e.g., an adaptation of the divide-and-conquer algorithm in [15] for computing the Manhattan skyline. The binary searches in Step 2 also take $O(n \log n)$ time.

Finally, the Interval Hitting Algorithm can be implemented in $O(n \log n)$. Indeed, an interval is hit by the disk $D_{last(p)}$ if its left covering disk is before $D_{last(p)}$ in the skyline S. Therefore, by traversing all intervals sorted in ascending order of their left ends $D_{first(p)}$, we can delete each hit interval in constant time per interval. ■

Remark. The approximation ratio of 2 in Theorem 3 is tight: Figure 6 gives an instance when the optimum disk cover consisting of skyline disks has size 2, while there is a single disk covering the two points of $\mathcal{P}$.

4 Exact combinatorial disk covering in a quadrant

In this section we present an $O(n^2)$ exact algorithm for the 1-Hop Disk Cover problem with all points of $\mathcal{P}$ coming from an exterior quadrant Q of unit disk A. In the next section we describe a faster $O(n \log^2 n)$ implementation of this algorithm based on efficient data structures.

In [3], a different $O(n^2)$ algorithm is presented for the 1-Hop Disk Cover problem with all points of $\mathcal{P}$ coming from Q. That algorithm seems easier to implement, but the proof of its correctness is more invloved and it does not

appear to have an implementation faster than $O(n^2)$.

The algorithm (see Algorithm 3) starts with sorting and renumbering all disks D_i with respect to the intersection points of ∂D_i with ∂Q, the boundaries of D_i and the quadrant Q. (see Step 1). The first and the last disk in this order covering each point are determined. Finally, a modified version of the Interval Hitting Algorithm (see Step 3 of Algorithm) finds the minimum disk cover.

Algorithm 3: Combinatorial Disk Covering

Input: Unit-disk A, set of unit disks $\mathcal{D}$ centered inside A, set of points $\mathcal{P}$ in the exterior quadrant Q of A such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{D}\}$

Output: Minimum size subset $\mathcal{F} \subseteq \mathcal{D}$ such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{F}\}$

---

1. For each $D_i \in \mathcal{D}$ find l_i and r_i, the two points of intersection between the boundaries ∂D_i with ∂Q. We assume that $l_j < r_j$ in a fixed orientation of ∂Q. Renumber the disks in $\mathcal{D}$ such that either $l_i < l_{i+1}$ or $l_i = l_{i+1}$ and $r_i < r_{i+1}$ for every $i = 1, \ldots, m-1$. Further, $D < D'$ denotes that the disk D has smaller index than the disk D'
2. For every point $p \in \mathcal{P}$, compute D_f^p and D_q^p, the first and last disks containing p.
3. Set $\bar{\mathcal{P}} \leftarrow \mathcal{P}$ and $\mathcal{F} \leftarrow \emptyset$
4. While $\bar{\mathcal{P}} \neq \emptyset$
   a) Find $p \in \bar{\mathcal{P}}$ with the minimum last disk $D = D_q^p$
   b) While there is $p' \in \bar{\mathcal{P}}$ with $D_f^{p'} < D$ and $p' \notin D$
      Replace D by the the last disk before D containing p
   c) Set $\mathcal{F} \leftarrow \mathcal{F} \cup \{D\}$
   d) Remove from $\bar{\mathcal{P}}$ all the points p' with $D_f^{p'} \leq D$
5. Output $\mathcal{F}$

We start the proof of correctness of the Algorithm 3 with the following definition and a crucial topological lemma. For $i < j < s$, we say that disks D_i and D_s *supercover* disk D_j if there is a point p in $\mathcal{P}$ such that $p \notin D_j$ and $p \in D_i \cap D_s$.

Lemma 10 *There is an optimum solution which uses no supercovered disk.*

Proof. Assume, for a contradiction, that *OPT* is an optimum solution such that the area of $\cup_{D \in OPT}(Q \cap D)$ is maximum, and that there is a disk $D_j \in$ *OPT* which is supercovered, by say D_i and D_s, where $i < j < s$. This means

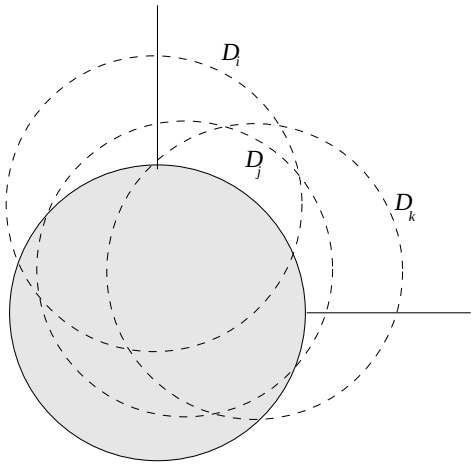

Figure 7: The existence of a nonempty region in $Q \cap ((D_i \cup D_k) \setminus D_j)$ implies that $(D_j \cap Q) \subseteq (D_i \cap D_k)$.

that there is a point $p \in \mathcal{P} \cap D_i \cap D_s$ which is not in D_j. There is a disk $D_k \in \mathit{OPT}$ such that $p \in D_k$. In the following we consider only the case when $k > j$, the other case being symmetric.

Lemma 2 implies that $(D_j \cap Q) \subseteq (D_i \cup D_k)$. See Figure 7 for an illustration.

Consider the walk on ∂D_i starting from l_i towards r_i. Let p_1 be the last point in this walk (p_1 might not be in $\mathcal{P}$) which is inside a disk D_g with $g < i$ and $D_g \in \mathit{OPT}$. Let p_2 be the first point in this walk (p_2 might not be in $\mathcal{P}$) which is inside a disk D_r with $i < r$ and $D_r \in \mathit{OPT}$. We have two cases.

If p_1 comes before p_2 on the walk on ∂D_i, then $\mathit{OPT} \setminus \{D_j\} \cup \{D_i\}$ is solution of the same size as *OPT*, but of bigger total area, as it includes the arc of ∂D_i in between p_1 and p_2. We obtained a contradiction to the assumption that *OPT* is an optimum solution such that the area of $\cup_{D \in \mathit{OPT}}(Q \cap D)$ is maximum.

Now assume p_2 comes no later than p_1 on the walk on ∂D_i. Let $D_g < D_i$ and $D_r > D_i$ be the disks in *OPT* such that $p_1 = \partial D_i \cap \partial D_g$ and $p_2 = \partial D_i \cap \partial D_r$. The fact that there is a point in $(Q \cap D_i \cap D_k) \setminus D_j$ and Lemma 2 imply that $\partial D_i \cap \partial D_k$ comes before $\partial D_i \cap \partial D_j$ on the walk, and therefore $r \neq j$. As $\partial D_i \cap \partial D_r$ comes no later than $\partial D_i \cap \partial D_g$, Lemma 2 implies that $(D_i \cap Q) \subseteq (D_g \cup D_r)$. Then $(D_j \cap Q) \subseteq (D_g \cup D_r \cup D_k)$, implying that $\mathit{OPT} \setminus \{D_j\}$ is a solution. This contradicts the fact that *OPT* is an optimum solution, and completes the proof. ■

From now on, we assume that *OPT* is a fixed optimum solution which contains no supercovered disk.

Based on the previous lemma, an $O(n^4)$ exact algorithm for disk cover is immediate: eliminate all the supercovered disks. After that, for every point $p \in \mathcal{P}$, the set of disks which cover p forms an interval. Then the Interval Hitting Algorithm finds the optimum solution.

To prove that the runtime of the Algorithm 3 is $O(n^2)$ we need to show that the total time for checking while-condition in step 4.b. is $O(n)$. This fact follows from the following lemma.

Lemma 11 *A disk cannot appear as D, inside the inner* **while** *loop (Step 4.b), for two different points.*

Proof. Assume, for a contradiction, that two points, p_1 and p_2, selected in this order in Step 4.a, use the D in Step 4.b. Since p_2 is not removed from $\bar{\mathcal{P}}$ while processing p_1, we have $D_f^{p_1} < D_f^{p_2}$. There is a disk $\bar{D}$ which is not supercovered and contains p_2. We have $\bar{D} \leq D$, since otherwise $\bar{D}$ would be selected by p_2 in Step 4.c, and D would not be processed by p_2. Thus $D_f^{p_1} < D_f^{p_2} \leq \bar{D} \leq D \leq D_q^{p_1}$, and since $\bar{D}$ is not supercovered, we deduce that $\bar{D}$ contains p_1.

Let D_1 be the disk selected in Step 4.c while processing p_1. We cannot have $D_1 < \bar{D}$, since otherwise, while processing p_1, $\bar{D}$ would have been considered as D' before D_1, and since $\bar{D}$ is not supercovered, $\bar{D}$ would be selected by p_1 in Step 4.c. So we have $D_f^{p_2} \leq \bar{D} \leq D_1 \leq D \leq D_q^{p_2}$. As D_1 is selected while processing p_1, the check in the **if** statement ensures that D_1 contains p_2. So p_2 is eliminated from $\bar{\mathcal{P}}$ in Step 4.d while processing p_1, a contradiction. ■

Lemma 12 *The runtime of Algorithm 3 is $O(n^2)$.*

Proof. Using straightforward **for** loops, every inner step, except for the inner **while** loop (Step 4.b), of the algorithm takes time $O(n)$. The replacement of a disk by a previous disk (inside the inner **while** loop) takes time $O(n)$ for a given point p. Checking the condition in the inner **while** loop (Step 4.b) takes $O(n)$ time, and Lemma 11 ensures that such a check is done at most once per disk. ■

Before proving the correctness of Algorithm 3, we give the intuition which motivates its steps. $\bar{\mathcal{P}}$ is the current set of uncovered points. In each iteration of the **while** loop 4, as in the Interval Hitting Algorithm, we pick the point with the first last-covering disk. Then we select a disk which includes this point, preferably one closer to the last-covering disk (in order to include a larger number of points), but we exclude certain supercovered disks. Note that the inner **while** loop 4.b jumps over disks which are supercovered.

Theorem 13 *Algorithm 3 gives an optimal solution for the 1-hop Disk Cover problem in a quadrant.*

Proof. First we show that every point is covered. We look at the current situation at the beginning of Step 4: the **while** loop. By the time Step 4.b (the inner **while** loop) is finished, p is removed from $\bar{\mathcal{P}}$. Indeed, in *OPT*, there is a disk D_{opt}^p which is not supercovered and which covers p, and $D_f^p \leq D_{opt}^p \leq D_q^p$. Every disk ignored by the algorithm in Step 4.b is supercovered, since by the choice of p in Step 4.a, we have $D_f^{p'} \leq D' \leq D_q^p \leq D_q^{p'}$. Therefore the inner **while** loop will stop at D_{opt}^p, if not earlier.

To prove optimality, let $p_1, p_2, \ldots, p_s$ be the set of points selected by the algorithm in Step 4.a. We claim that no disk D of OPT can include p_i and p_j, where $i < j$. Let D_i be the disk selected to cover p_i in Step 4.c. Note that while processing p_i, the algorithm cannot ignore D in Step 4.b, as D is not supercovered, and therefore $D \leq D_i$. Assuming

by contradiction that D contains p_j, we obtain that $D_f^{p_j} \leq D$. But then $D_f^{p_j} \leq D_i$, and p_j should have been removed from $\bar{\mathcal{P}}$ in Step 4.d.

It follows that $|OPT| \geq s$, and the theorem follows from the fact that the algorithm also selects s disks. ■

Theorems 1, 13, and Lemma 12 imply:

Theorem 14 *Combined with Algorithm 3, Algorithm 1 runs in $O(n^2)$ time and has an approximation ratio of 3 for the Minimum Forwarding Set problem.*

5 An $O(n \log^2 n)$ implementation of the combinatorial disk covering

In this section we describe an enhanced data structure based on Voronoi diagrams which allows to implement Algorithm 3 in time $O(n \log^2 n)$

Step 1 can obviously be executed in $O(n \log n)$. Next we describe the data structure which we use for Step 2. Construct a balanced binary search tree T, with the centers of the individual disks as leaves, in the sorted order obtained in Step 1. For a node $v \in T$, denote by A_v the set of the centers in the subtree rooted at v. For every node v in T, construct the Voronoi diagram of A_v, and preprocess the diagram for membership queries. A membership query returns, for a give point p, the face of the Voronoi diagram where p lies. The preprocessing time for constructing the Voronoi diagrams is $O(n \log n)$ per level in T (see, for example, Chapter 20.2 of [8]), for a total of $O(n \log^2 n)$. The space requirement is $O(n)$ per level of T, for a total of $O(n \log n)$. Preprocessing a node v for membership queries also takes time $O(|A_v| \log |A_v|)$ and space $O(|A_v|)$, and a query can be answered in time $O(\log |A_v|)$ (see, for example, Chapter 30.3 of [8]). The total time is $O(n \log^2 n)$, and the total space is $O(n \log n)$.

With this data structure, given any node v in T, and a point p, one can find whether a disk in the subtree rooted at v covers p in time $O(\log n)$, by finding out in which cell of the Voronoi diagram of A_v p lies, and computing the distance from p to the center of that cell. Using this observation, a binary search in T can find for a point p the disks D_f^p and D_q^p in time $O(\log^2 n)$.

Before proceeding to Step 3, in time $O(n \log n)$, sort the points of $\mathcal{P}$ with respect to D_f^p and put them in a list L_1. To represent $\bar{\mathcal{P}}$, we only use the position in L_1 of the first point of $\bar{\mathcal{P}}$; this is enough since the points removed from $\bar{\mathcal{P}}$ in Step 4.d are consecutive in L_1, and include the first uncovered point of L_1. This representation allows the implementation of Step 4.d in total time $O(n)$: to remove points from $\bar{\mathcal{P}}$, simply move forward in L_1, checking at every step if $D_f^{p'} \leq D$.

Also, in time $O(n \log n)$, sort the points of $\mathcal{P}$ with respect to D_q^p and put them in a separate list L_2. This allows us to implement Step 4.a to run in total time $O(n)$, by keeping track of where point p is in L_2, and moving only forward

in L_2, ignoring the points which are not in the current $\bar{\mathcal{P}}$.

In the following, we describe how to use the data structure T to implement one **replace** statement inside the inner **while** loop Step 4.b in time $O(\log^2 n)$. Start at the leaf of T which has the center of D. Let v be the current node and $p(v)$ be its parent in T. If v is a left child, then replace v by its parent, and repeat. Assume now that v is a right child, and let v' be its sibling. In time $O(\log n)$, we can check whether there is a center in $A_{v'}$ whose disk includes p. If yes, starting from v', find the last center in $A_{v'}$ whose disk covers p (this is the same binary search procedure used to compute D_q^p). If no, then replace v by its parent, and repeat. Theorem 13 implies that there always is a disk before D which contains p, and thus the above procedure is correct.

The fact that all executions of Step 4.b take time $O(n \log^2 n)$ follows from Lemma 11.

The condition of the inner **while** statement Step 4.b is checked by using a data structure T' which we describe below. T' is a balanced binary tree, whose leaves are the points of $\mathcal{P}$, sorted as in L_1 (the smaller D_f is first). For a node $v \in T$, denote by A'_v the set of points in the subtree rooted at v. For every node v in T, construct the furthest-site Voronoi diagram of A'_v, and preprocess the diagram for membership queries. A cell of the furthest-site Voronoi diagram contains points which have the same furthest site in A'_v. The furthest-site Voronoi diagram can be constructed in $O(|A'_v| \log |A'_v|)$ and uses $O(|A'_v|)$ space (see, for example, Chapter 20.3 in [8], and with the same time and space bounds, it can be preprocessed for membership queries (Chapter 30.3 of [8]). The total preprocessing time is $O(n \log^2 n)$ and the total space is $O(n \log n)$. Given the center of a disk D, and a node $v \in T'$, finding if all the points of A'_v are contained in D can be done in time $O(\log n)$ by a membership query in the furthest-site Voronoi diagram of A'_v.

Now we describe how exactly to check if there is a point $p' \in \bar{\mathcal{P}}$ with $D_f^{p'} < D$ and $p' \notin D$. Given D, the set of points of $\bar{\mathcal{P}}$ with $D_f^{p'} < D$ are consecutive in L_1. We denote by p_1 the first such point, and note it is the first in the remaining part of L_1. We denote by p_2 the last such point, and note it can be found by a binary search. Checking the condition for p_1 and p_2 takes constant time.

Binary search can also locate p_1 and p_2 in the leaves v_1 and v_2 of T', and let v be the least common ancestor of v_1 and v_2 in T'. Let v' be v_1, and $p(v')$ be its parent in T'. As long as $p(v') \neq v$, do the following: if v' is a left child, and if v'' is its sibling, check if D contains $A'_{v''}$. Then let v' be $p(v')$.

Similarly, let v' be v_2. As long as $p(v') \neq v$, do the following: if v' is a right child, and if v'' is its sibling, check if D contains $A'_{v''}$. Then let v' be $p(v')$. If any of the checks above fails, then there is a point $p' \in \bar{\mathcal{P}}$ with with $D_f^{p'} < D$ and $p' \notin D$; otherwise there is no such point. The total time for one disk D is $O(\log^2 n)$, as there are at most $2 \log n$ queries of the type: check whether all the points of A'_v are contained in D.

Finally, Lemma 11 implies that a disk D will need at most once such processing. Thus Step 4.b takes $O(n \log^2 n)$

Based on the discussion above, we have:

Theorem 15 *Algorithm 3 can be implemented to run in time* $O(n \log^2 n)$.

6 The general minimum disk cover problem

In this section we describe a constant-factor approximation algorithm for the following

Minimum Disk Cover Problem. Given a set of unit disks $\mathcal{D}$ and a set of points $\mathcal{P}$ in the Euclidean plane, find a minimum-size subset $\mathcal{F} \subseteq \mathcal{D}$, such that $\mathcal{P} \subseteq \cup\{D \in \mathcal{F}\}$.

This problem is NP-Hard since it contains as a special case Dominating Set in unit-disk graphs, a problem shown to be NP-Hard in [5]. A polynomial-time algorithm with constant approximation ratio for Minimum Disk Cover was first provided by [2].

If we can obtain a constant ratio for covering an equilateral triangle with sizes equal to 1, we can obtain a constant ratio for the whole plane, by tiling the plane into triangles and separately covering all the triangles, and using the fact that one disk in the optimum can only cover points in a constant number of triangles.

Let ABC be such a triangle. If no point of $\mathcal{P}$ is in the triangle, there is nothing to be done. Also, if there is a disk $D \in \mathcal{D}$ whose center is in the triangle, then D covers all the triangle. So, in the following, we assume all the points are in the triangle, and all the centers of disks in $\mathcal{D}$ are outside the triangle.

The algorithm has four phases:

1. After removing those disks that do not intersect the triangle, partition the remaining disks into three sets $\mathcal{D}_1$, D_2, and $\mathcal{D}_3$, such that all the centers of the disks in D_1 are on the other side of the line AB than C, all the centers of the disks in D_2 are on the other side of the line BC than A, and all the centers of the disks in D_3 are on the other side of the line AC than B. If a disk could be put in more than one D_i, pick one arbitrarily.

2. For $i = 1, \ldots, 3$, let $Q_i = B$ be the triangle and let J_i be the line which separates the centers of the disks of $\mathcal{D}_i$ from the interior of the triangle. Find the skyline as in Section 3, and compute $\mathcal{F}_i$, the set of disks containing some arc of the skyline.

3. Write the natural Integer Programming formulation involving only the disks in $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3$. Solve the Linear Programming relaxation.

4. Round the linear programming optimum to an integer solution, as described in Subsection 6.1.

Later we prove Theorem 16, which claims that the algorithm described above has approximation ratio at most 6 for the problem of covering the points inside the triangle.

First, we note that Lemma 2 holds easily when J_i is a straight line. For each $\mathcal{F}_i$, Lemma 8 also holds. Let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3$, and assume $\mathcal{F}$ is sorted with $\mathcal{F}_1$ (which is sorted) followed by the sorted $\mathcal{F}_2$, and followed by the sorted $\mathcal{F}_3$. Lemma 9 also holds, and therefore $\mathcal{F}$ contains a solution at most twice opt, the size of an optimum solution.

6.1 Rounding

We use the natural IP, with variables x_D, for $D \in \mathcal{F}$:

$$\text{minimize} \sum_{D \in \mathcal{F}} x_D$$

$$\text{subject to} \sum_{D \,:\, P \in D} x_D \geq 1 \quad \forall P \in \mathcal{P} \tag{1}$$

$$x_D \in \{0,1\} \quad \forall D \in \mathcal{D}. \tag{2}$$

Let LP be the linear programming relaxation of IP, obtain by replacing the constraints 2 by

$$x_D \geq 0 \quad \forall D \in \mathcal{D}. \tag{3}$$

Let Z^*_{IP} the value of the IP optimum. As argued above, we have $Z^*_{IP} \leq 2\ opt$.

Let y be a (fractional) solution to LP. For a point $P \in B$, the set of disks covering it consists of at most three intervals, say I^P_1, I^P_2, and I^P_3. For one of the three intervals, which we call simply I^P, we have: $\sum_{D \in I^P} y_D \geq 1/3$.

Consider the following integer program, which we call IP', with variables x_D, for $D \in \mathcal{F}$:

$$\text{minimize} \sum_{D \in \mathcal{F}} x_D$$

$$\text{subject to} \sum_{D \in I^P} x_D \geq 1 \quad \forall P \in \mathcal{P} \tag{4}$$

$$x_D \in \{0,1\} \quad \forall D \in \mathcal{D}. \tag{5}$$

Let LP' be the linear programming relaxation of IP'. The matrix of IP' is totally unimodular (see [6], Theorem 6.28, page 223, and Example 3 on the next page), and $3y$ is a solution to LP'. Therefore IP' has a solution of size at most $3 \sum_{D \in \mathcal{F}} y_D$, and an optimum for IP' can be found easily by the greedy algorithm, as described at the end of the proof of Theorem 3. Now, if y is an optimum solution to LP, then $\sum_{D \in \mathcal{F}} y_D \leq Z^*_{IP} \leq 2\ opt$, and therefore the solution found by the greedy algorithm has size at most $6opt$.

Rounding consists of finding for each point $P \in B$ the interval I^P, and then using the greedy algorithm to hit each I^P with elements of $\mathcal{F}$. In conclusion, we proved:

Theorem 16 *The algorithm described in this section has approximation ratio at most 6 for the covering points inside an equilateral triangle with sizes equal to 1 with unit-disks from a fixed set $\mathcal{D}$.*

Since a single disk from the optimum solution can cover points in at most 17 triangles of the tiling we conclude

Corollary 17 *There is a 102-approximation algorithm for the Minimum Disk Cover problem.*

7 Conclusions

In this paper we presented a geometric $O(n \log n)$ 6-approximation algorithm and a combinatorial $O(n \log^2 n)$ 3-approximation algorithm for selecting forwarding neighbors in wireless ad-hoc networks, significantly improving both the running time and the approximation ratio of the best previously known algorithm. An extension of our method can be used to obtain an alternative constant-ratio polynomial-time algorithm for the Minimum Disk Cover problem.

We mention that Theorem 13 is true in the following more general setting. Let J be an infinite simple Jordan curve which separates the plane into exactly two regions, and let B be one of these two regions. Let all points $\mathcal{P}$ be in B, and each D_j be a region bordered by a simple closed Jordan curve $\partial\mathcal{D}_j$. Also, each $\partial\mathcal{D}_j$ intersects the infinite curve J in exactly two points, and, for any two regions D_j and D_k, $\partial\mathcal{D}_j \cap \partial\mathcal{D}_k \cap B$ has at most one point. Moreover, whenever two of the curves above intersect, they cross each other. Then Lemma 10 holds, and therefore a polynomial-time exact disk covering algorithm exists.

Similar techniques can also be applied to a fractional solution to the natural linear program LP to obtain a rounding procedure with a ratio of 2 when $\mathcal{P}$ and the centers of $\mathcal{D}$ are separated by a straight line. Then, as in Section 6, it follows that the linear program LP has constant integrality ratio for the general problem. However, when the disks in $\mathcal{D}$ are weighted, we do not know the integrality ratio of the corresponding integer and linear programs. The linear program is given below:

$$\text{minimize} \sum_{D \in \mathcal{D}} w_D x_D$$

$$\text{subject to} \sum_{D\,:\,P \in D} x_D \geq 1 \quad \forall P \in \mathcal{P} \tag{6}$$

$$x_D \geq 0 \quad \forall D \in \mathcal{D} \tag{7}$$

Student Yuchen Wu, whom we wish to thank, implemented the geometric algorithm and a slower ($O(n^2)$) preliminary version of the exact combinatorial algorithm. On random instances, the much simpler geometric algorithm has solutions larger by 17–44% than the exact combinatorial algorithm.

References

[1] J. Broch, D. B. Johnson, and D. A. Maltz, The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks, IETF Internet Draft, draft-ietf-manet-dsr-05.txt, March 2001.

[2] H. Bronnimann and M.T. Goodrich, Almost Optimal Set Covers in Finite VC-Dimension. *Proc. 10th ACM Symp. on Computational Geometry (SCG)*, 1994, 293–302.

[3] G. Calinescu, I. Mandoiu, P-J. Wan, and A. Zelikovsky, "Selecting Forwarding Neighbors in Wireless Ad Hoc Networks," *Proc.* 5^{th} *International Workshop on Discrete Algorithms and Methods for Mobility*, 2001, 34–43.

[4] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operation Research*, 4(3):233–235, 1979.

[5] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit Disk Graphs", *Discrete Mathematics*, 86:165–177, 1990.

[6] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver *Combinatorial Optimization*, Wiley-Interscience, 1998.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest *Introduction to Algorithms*, McGrow Hill, 1990.

[8] J. E. Goodman and J. O'Rourke (editors), *Handbook of Discrete and Computational Geometry*, CRC Press, 1997.

[9] Z. J. Haas, M. R. Pearlman, and P. Samar. The Interzone Routing Protocol (IERP) for Ad Hoc Networks, IETF Internet Draft, draft-ietf-manet-zone-ierp-00.txt, January 2001.

[10] D. S. Hochbaum and W. Maass, "Approximation schemes for covering and packing problems in imageprocessing and VLSI", *Journal of the ACM*, 32(1): 130–136, 1985.

[11] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, "NC-Approximation schemes for NP- and PSPACE-hard problems for geometric graphs", *Journal of Algorithms*, 26(2):238–274, 1998.

[12] P. Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L. Viennot, and T. Clausen, Optimized Link State Routing Protocol, IETF Internet Draft, draft-ietf-manet-olsr-04.txt, March 2001.

[13] A. Laouiti, A. Qayyum, and L. Viennot, "Multipoint Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks", 35th Annual Hawaii International Conference on System Sciences (HICSS'2001)

[14] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi and D. J. Rosenkrantz, "Simple Heuristics for Unit Disk Graphs", *Networks*, Vol. 25, 1995, pp. 59–68.

[15] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP, Volume I: Design and Efficiency*. Benjamin/Cummings, 1991.

[16] S.-Y. Ni, Y.-C. Tseng, Yuh-Shyan Chen, and J.-P. Sheu, The Broadcast Storm Problem in a Mobile Ad Hoc Network", *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, 1999, Pages 151–162.

[17] C. E. Perkins, E. M. Royer, and S. Das, Ad Hoc On Demand Distance Vector (AODV) Routing, IETF Internet Draft, draft-ietf-manet-aodv-08.txt, March 2001.

[18] P. Sinha, R. Sivakumar, and B. Vaduvur, Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures, to appear in *IEEE INFOCOM* 2001.