DISS. ETH NO. 16800

Understanding Ad hoc Networks
From Geometry to Mobility

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of

Doctor of Sciences

presented by

REGINA O'DELL

Dipl. Inf.-Ing., ETH Zürich

born 26.04.1980

citizen of

Germany

accepted on the recommendation of

Prof. Dr. Roger Wattenhofer, examiner
Prof. Dr. Rajmohan Rajaraman, co-examiner
Prof. Dr. Dorothea Wagner, co-examiner

2006

для Довани

Abstract

With the increase in number and decrease in size of computing devices, the bulk of networking research has shifted its focus away from stationary, heavy-duty computer networks such as the Internet toward wireless ad hoc networks composed of small, cheap, and in many ways limited battery-operated devices. This paradigm shift unveils both unexpected limitations as well as unprecedented latitudes. The network designer has to deal with possibly severe size, energy, communication ability, and cost constraints while at the same time he must account for uncertain mobility of the nodes.

This dissertation looks into two particular aspects of the consequences of the aforementioned paradigm shift. A number of applications for wireless networks require the nodes to know their position, a key example is given by sensor networks where it is vital to associate a location to the reported sensor data. Given their size and energy restrictions, often it is not possible to equip the nodes with the necessary hardware to allow them to directly deduce their positions. Consequently, it becomes important to develop algorithms which compute the coordinates of the network nodes in software based on what limited connectivity and range information is available to them. This geometric aspect of wireless networks is the object of scrutiny in the first part of this thesis, culminating in the first – to the best of our knowledge – non-trivial approximation algorithm for the network embedding problem.

The second part of this dissertation investigates the newly-found freedom enabled by battery-powered devices: mobility. One of the primary operations drastically affected by mobility is routing, or, more generally, the task of disseminating information from one part of the network to another. As such, this dissertation provides preliminary results for analytically studying the effect of mobility on routing protocols in both highly and moderately dynamic networks. In the former case, we present several flooding and routing algorithms which reliably deliver a message from the source to the rest of the network or the destination, respectively, and which do not need infinite energy resources to accomplish this. In the latter case, we provide some specific bounds showing the tradeoffs between proactive and reactive routing approaches.

Zusammenfassung

Sowohl die rapide Miniaturisierung von heutigen Rechnern als auch deren allgegenwertige Präsenz haben zur Folge, dass sich die Wissenschaft vermehrt weg von stationären, hochleistungsfähigen Netzwerken wie dem Internet und hin zu drahtlosen Ad-hoc-Netzwerken bestehend aus kleinen, billigen und in vieler Hinsicht eingeschränkten batteriebetriebenen Geräten wendet. Dieser Paradigmenwechsel deckt sowohl unerwartete Einschränkungen wie auch beispiellose Freiheiten auf. Neuerdings muss sich der Netzwerkentwickler mit teilweise schwerwiegenden Grössen-, Energie-, Kommunikations- und Kostenbeschränkungen befassen, und gleichzeitig muss die ungewisse Mobilität der Knoten in Betracht gezogen werden.

Diese Dissertations betrachtet zwei spezielle Aspekte der Folgen vom obengenannten Paradigmenwechsel. Zum einen ist es für viele Anwendungen von drahtlosen Netzwerken unerlässlich, dass die Knoten ihre Positionen kennen. Ein Musterbeispiel bilden Sensornetzwerke, wo es wesentlich ist, Messdaten von den Sensoren mit deren Koordinaten zu verknüpfen. Allerdings ist es oftmals durch die vorhandenen Grössen- und Energiebeschränkungen nicht möglich, die Knoten mit der notwendigen Hardware, mittels welcher die Position direkt bestimmt werden kann, auszustatten. Folglich ist es wichtig, dass man, basierend auf Topologie- und Reichweiteinformation, die Knotenkoordinaten auch in Software, also mit Hilfe von Algorithmen, bestimmen kann. Es ist diese geometrische Sichtweise auf drahtlose Netzwerke welche den ersten Teil der Arbeit dominiert und in einem, unseres Wissens nach ersten, nichttrivialen Approximationsalgorithmus für das Netzwerkeinbettungsproblem gipfelt.

Der zweite Teil dieser Dissertation befasst sich mit der neugewonnen Freiheit, welche erst durch batteriebetriebene Geräte ermöglicht wird: Mobiliät. Deren drastische Auswirkung spürt man schon in einem der essentiellsten Netzwerkoperationen, dem des Routing. Letzteres kann man auch allgemein mit der Aufgabe beschreiben, Informationen von einem Teil des Netzwerks in einen anderen weiterzugeben. Diese Arbeit untersucht in einem analytischen Sinne die Auswirkung von Mobilität auf Routingprotokolle in sowohl mässig als auch hochgradig dynamischen Netzwerken. In letzterem Fall werden mehrere Flooding- bzw. Routingalgorithmen vorgestellt. Ein wichtiges Merkmal dieser Algorithmen ist, dass sie nicht unendliche Energieressourcen für den Zweck, eine Nachricht zuverlässig von der Quelle zum Rest des Netzwerkes bzw. zum angegebenen Ziel zu liefern, aufbrauchen. Im Falle von gemässigter Mobilität werden konkrete Grenzen zwischen proaktiven und reaktiven Routingansätzen ausgelotet.

Contents

Chapter 1

Introduction: Wireless Ad hoc Networks

Who am I? Where am I; and where am I going? Three deep and ancient philosophical questions of mankind. But perhaps it is a bit presumptuous to think that only human minds seek answers to these mysteries. In the very least, if we take them on face value and apply the term "think" rather liberally, then these questions become relevant for a multitude of living and even non-living beings. For instance, in order to survive, a tiger needs to be acutely aware of its predator status, and it needs the ability to locate itself in relation to its prey, at which point it must exert the necessary control to move towards it. We can delve even lower in the complexity hierarchy of organisms down to non-sentient physical entities such as human-built sensor nodes. Such a node, too, needs to "know" of its role as an information gathering, processing, and relaying unit; it needs to be "aware" of its position so as to make the information useful; and it needs to "act" appropriately in order to forward the information.

The primary source of inspiration for this dissertation is just such collections of artificially created wireless entities along with the types of questions and problems they face. In the broadest sense, we call them *wireless ad hoc networks* because they consist of nodes which are deployed in a potentially arbitrary and likely unpredictable manner and whose communication ability is restricted to a wireless broadcast medium. A common application and subclass of these types of networks are sensor networks which gather data about some aspect of their environment and report back the information to a central sink attached to a database. The future of wireless networks (whether sensor or not) is often envisioned as a large aggregation of small, if not to say tiny, low-power devices. The challenges that come with this paradigm are twofold. On the one hand, engineers are trying to build hardware that conforms to such a "smart dust" ideal. On the other hand, researchers need

to construct models and algorithms that are suited to the very stringent requirements that these devices impose: low energy consumption, low storage and computational capabilities, and basic radio transmission hardware.

We will study the fundamentals of wireless ad hoc networks by examining their geometry and seeing how the concept of mobility impacts performance. Even though it may seem that the aforementioned issues are highly practical in nature, we will look at wireless ad hoc networks through a theoretical lens. One of the reasons for doing so is that, for the time being, actual wireless ad hoc and sensor networks are based on what we consider to be "transient technology" in that the used hardware is still in its infant stage. In contrast to chip design and the domination of Moore's Law at the closing of the twentieth century, we believe that it is virtually impossible to reliably predict wireless network hardware development, not to mention the time scales involved for when breakthrough advancements in scale, energy, and communication ability will be made. Instead, in this dissertation we adopt the view of studying first principles because we believe that we should not forgo a solid theoretical underpinning for short-term practical goals dictated by evolving market values.

Coming back to our initial three questions, this dissertation does not dare touch upon the first one; every one and every thing has to answer that for him- or itself. For the second one, however, we provide a constructive algorithm which, for the first time, gives non-trivial bounds on the error we can make in the case of a wireless node amidst a network of other such nodes. While we might not be able to fully handle the third question, we examine to what extent anything is possible regardless of what the answer to that question might be. Specifically in the case of wireless mobile ad hoc nodes, we provide efficient and reliable algorithms in the face of almost arbitrary mobility of all of the nodes. We further open up a path for analytically dealing with mobile routing concepts in the pro- versus reactive debate.

Part I

Geometry

Chapter 2

The Geometry of Wireless Networks

Roughly speaking, the geometry of a wireless network is the result of putting together the topology and the metric[1] information of the network. The basic problem comes in many guises in the literature where it is known under terms such as localization, positioning, or virtual coordinates computation. They handle different aspects of the problem or deal with different input requirements, but basically all are concerned with the same question: How does one assign representative coordinates to the nodes in the network? In the virtual coordinates formulation, the task is to assign each node to a point in the plane such that the wireless links are accurately represented. The coordinates need to be only consistent *relative* to each other. The positioning variant asks for points that are consistent with respect to an *absolute* coordinate system, defined by several anchor points with known positions. The term localization has been used in both contexts. Collectively, we describe these aspects – the geometry of a network – in terms of an embedding: an assignment of nodes to points in the two-dimensional Euclidean plane.

The reasons for studying the geometry of wireless networks, apart from its intellectual appeal, stem from very practical concerns. The recent flurry of activity in the field of *sensor networks*, basically a subclass of wireless ad hoc networks, highlights a number of important research areas. One quintessential service of sensor networks is the availability of location information: In most cases it is imperative to attach position information to the sensed data. The network designer is faced with the decision of how to provide this information. In some cases, it might be possible to hardcode the position into the sensor nodes at or before deployment time, such as in building monitoring. When this is not feasible, the designer might have the option to equip

[1] that is, the link and non-link weights

some or all of the nodes with localization hardware, be it GPS or some other specialized hardware for indoor positioning, for instance. In this case, the network architect needs to weigh the benefits against the costs introduced by the positioning hardware. The tradeoff here is between the accuracy of the positions determined with aid of the hardware versus the burden placed on the nodes due to increased size, energy drain, and costs. In extreme situations, it might not be possible for any nodes to obtain absolute position information, in which case the problem is known as computing virtual coordinates. A prominent application of virtual coordinates, also in general ad hoc networks, is to enable what is known as *geographic routing* [88, 122, 145], a successful, simple, and well-documented routing paradigm based on the use of coordinate information. Lastly, another important usage of network coordinates is for *visualization* purposes. Be it in the debugging or during the deployment stage of wireless ad hoc networks, an accurate and timely display of the nodes' location and their interconnections can prove to be an invaluable monitoring device.

There are two main ways for additional hardware to support localization efforts of the network. Nodes that are equipped with a GPS receiver provide a global and absolute orientation of the network. Another form of hardware allows for inter-node distance or angle measurements, providing relative information. As with most of the current wireless network technology, also these hardware components are far from being perfected and it is not foreseeable at which point all of the factors cost, size, and energy will be sufficiently minimized. Some recent analytical considerations have shown that inaccurate measurements can in fact hinder localization efforts. Simulations in [23] show that starting at a range measurement error of about 30%, the resulting localization is worse than by topology alone. Calculations and simulations in [110] for a specific algorithm show how large the error in link distance and angle measurements can be until the extra information becomes useless. It turns out that a large part of the parameter space of when the localization error is minimal is occupied by the algorithm using only hop information. This is the primary reason why we look at localization in a "range free," also known as topological or connectivity-only, model. What this means is that the knowledge of the nodes is restricted to the presence and absence of links. These, in turn, are determined by the distances between the nodes and possibly their surroundings in forms of various obstructions. Simply put, instead of looking at the equation

$$\text{topology} + \text{metric} = \text{geometry}$$

we look at how much we can learn from

$$\text{topology} + (\text{simplified metric}) \Rightarrow \text{geometry}$$

where the simplified metric stands for the distance constraints implied by the existence (or lack thereof) of a wireless link between two nodes. Note

that, because of this abstraction, the geometry information must now be necessarily incomplete.

The status quo of network localization research, aside from hardware advances, is a collection of heuristics with various levels of refinements which we will discuss in detail later on. What is missing from a theoretical point of view are some insights into the general problem of network embedding. In order to tackle the problem analytically, our first layer of abstraction is modelling wireless networks as unit disk graphs (UDGs). Unfortunately, already recognizing whether a given graph is a UDG is NP-hard [29]. Consequently, we seek to find good approximation algorithms. If we only look at positioning, then the quality of a positioning algorithm could be the largest deviation of a node's computed location from its actual location. In the absence of absolute reference points such as in the case of computing virtual coordinates, this measure is meaningless. Thus an important step in the process of treating network localization analytically is formalizing how to assess the quality of an embedding, be it absolute or relative. A closing question which begs itself after so much abstraction is what is a good model for wireless networks? We will turn to that at the end of this part in Chapter 6 to look at possible alternatives and their utility.

2.1 Model

The entire area of network localization, in one form or another, can be captured by the idea of graph embedding, formalized below.

Definition 2.1 (Embedding). *An* embedding *of a graph $G = (V, E)$ in the Euclidean plane is a mapping $f : V \to \mathbb{R}^2$, i.e., each vertex v is identified with a point (x, y) in the plane.*

Denote by $\|\cdot\|$ the l_2 norm, that is, the Euclidean length of a vector in the plane. In a slight abuse of notation, by $\mathbb{R}^2$ we implicitly mean the Euclidean plane, that is, $\mathbb{R}^2$ equipped with the l_2 metric, since this is the main object of interest. It will be stated explicitly when we discuss more general metric spaces. Note that this is in contrast with the graph metric $d_G(\cdot, \cdot)$ which gives the length of the shortest path between two vertices in the graph G. Since we look at network localization mainly from an analytical perspective, the unit disk graph has proven itself a popular network abstraction.

Definition 2.2 (Unit Disk Graph). *A graph $G = (V, E)$ is called a* unit disk graph (UDG) *if there exists an embedding f such that $\{u, v\} \in E \Leftrightarrow \|f(u) - f(v)\| \leq 1$ holds for any $u, v \in V$. Such an f is called a* realization *of G.*

A realization is also called a representation in the literature. Attaching virtual coordinates to wireless ad hoc and sensor nodes corresponds to finding

a realization of a given unit disk graph. As mentioned above, however, finding such a realization is NP-hard. Therefore, we resort to finding algorithms which compute an approximate realization, i.e., an embedding which may violate some unit disk constraints, but does not do so too much.

Problem 1 (Virtual Coordinates). *An algorithm for the* virtual coordinates problem *takes as input a unit disk graph G and outputs an embedding f for G.*

The question is how to quantify how "close" an embedding is to a realization. The motivation for UDGs and thus the goal of an approximation algorithm is to map adjacent nodes to close-by coordinates and non-adjacent nodes to distant coordinates. This intuition naturally leads to a quality measure based on the ratio between the longest edge to the shortest non-edge in the embedding. Therefore, we formally define the *quality of an embedding* as follows.

Definition 2.3 (Quality). *Let f be an embedding of UDG $G = (V, E)$. Let $\rho(u, v) = \|f(u) - f(v)\|$ denote the (Euclidean) distance between nodes u and v in f. We define the* quality *of the embedding $f(G)$ as*

$$Q(f(G)) := \frac{\max_{\{u,v\}\in E} \rho(u, v)}{\min_{\{\hat{u},\hat{v}\}\notin E} \rho(\hat{u}, \hat{v})}. \tag{2.1}$$

Let $\mathcal{G}$ denote the family of all unit disk graphs. We consider algorithms which, given an input graph $G \in \mathcal{G}$, compute an embedding $f_{ALG}(G)$. We say that a virtual coordinates algorithm achieves approximation ratio α if $Q(f_{ALG}(G)) \leq \alpha$ for all $G \in \mathcal{G}$. Note that the approximation ratio, i.e., the "quality", is something we wish to minimize and any realization r of G is an embedding where $Q(r(G)) \leq 1$. In place of the general definition in Equation (2.1) we could also have considered an alternative formulation. Consider scaling the embedding such that non-edges are required to be greater than one, then minimize the edge length. This is a more standard version of an optimization problem and the quality refers to how well we can compress the edges to the optimum unit value.

An easy upper bound on the approximation ratio is $O(\sqrt{n})$, n being the number of nodes. To see that, place the nodes arbitrarily on the points of a grid with $\sqrt{n}$ per side, the space between two neighboring grid points being (about) one unit. The possibly longest edge is the diagonal of the grid, or in the order of $\sqrt{n}$.

Much of the early literature on network localization was interested in determining the absolute positions for the case where several rigid nodes, called anchors, were present. These anchors are part of the network and know their exact positions, information which they distribute to the other nodes. Below, we can think of p as a partially revealed embedding.

Problem 2 (Positioning). *An algorithm for the* positioning problem *collects, in a distributed manner, information $V' \subseteq V$, $E' \subseteq E$ about a given unit disk graph $G = (V, E)$ where a subset $A = \{a_1, \ldots, a_k\} \subset V$, called anchors, have fixed positions $p(a_1), \ldots, p(a_k)$. The algorithm is given a set $P \subset V$ of nodes for which it outputs an embedding $f : P \to \mathbb{R}^2$.*

Since sufficiently many anchors will help to "pin down" the rest of the network, it is more straightforward to measure the quality of a positioning algorithm. Note that, however, even if the entire topology is known but the graph is sparse, the error of any positioning algorithm must necessarily be large. For instance, if the network consists of a path with anchors at the end points, then the error for the nodes in the middle is in the order of the distance separating the anchors without further available information.

Definition 2.4 (Error). *Let f be an embedding of $P \subset V$ of a UDG $G = (V, E)$ with anchor set $A = \{a_1, \ldots, a_k\} \subset V$ and anchor positions $p : A \to \mathbb{R}^2$. We define the* error *of the embedding $f(P)$ as*

$$E(f(P)) := \max_{r} \max_{v \in P} \|f(v) - r(v)\| \tag{2.2}$$

where the first maximization is over all realizations $r : V \to \mathbb{R}^2$ such that $r(a_i) = p(a_i)$ for all $a_i \in A$.

An optimal positioning algorithm will wish to minimize this maximum possible error. Observe the difference between our general quantification of error and that used in most prior work. Since all algorithms discussed in Section 2.4 are evaluated by simulations, the error of the simulated algorithm is the absolute difference to the position generated by the simulated instance. However, for a given set of anchors and a unit disk graph, there may be several realizations and the crucial point is that they may be indistinguishable to any algorithm. This should be taken into account when computing the embedding.

A similar point has been raised very recently in [21]. The authors advocate the use of *regions* instead of *points*. The regions represent a subset (strong deployment region) or superset (weak deployment region) of the total set of possible point locations consistent with the input data. The above definition adheres to that idea by maximizing over all possible realizations; reducing the error function to a single value merely gives a bound on the size of the region, since this is what is ultimately of interest. If the embedding is optimal, then the error is proportional to the greatest diameter of the strong deployment region. In other words, while some works concern themselves with determining when a graph is globally rigid, that is, there is a unique solution (possibly up to translation and rotation), we look at the general case where there might not be a unique embedding. In that case we are interested in how far apart the nodes in all such possible embeddings can be, or, in other words, find a metric for how far apart two embeddings are.

2.2 Inherent Difficulties

As often encountered in mathematical theory, the motivation for the definition of unit disk graphs has a realization in mind, but the graph which is distilled from that geometric picture is a combinatorial object in its own right. While going from the geometric to the combinatorial view is easy, quite the opposite is true for the other way. Given a graph G, it was first shown by Breu and Kirkpatrick [29] that determining whether G is a unit disk graph is NP-hard. Consequently, finding a realization of a unit disk graph is NP-hard. A whole flood of recent hardness results has shown that a number of variants and relaxations of the problem, including approximations, remain difficult, see Table 2.1 for a quick overview. With the approximation ratio defined as above, [97] showed that no polynomial-time approximation scheme (PTAS) exists unless P = NP, and, independently, [84] strengthened this to give a $\sqrt{3/2}$ lower bound on the approximation ratio. In some scenarios, it might be possible not only to detect connectivity but also to measure the edge lengths. Then the problem of embedding the graph G with given edge lengths and non-edges at distance greater than 1 remains NP-hard. This was shown by an easy reduction from the partition problem in [14] and, independently, by a more elaborate reduction from boolean circuit satisfiability in [7]. The latter result has two corollaries: The NP-hardness can be extended to randomized algorithms and, again, there is no PTAS for this problem. Recently, it was shown [21] that even if we add relative angle information[2] to the input, then it is still NP-hard. This was a follow-up paper to [31] which showed that the UDG embedding problem without edge lengths but knowing the absolute angles between any two edges is also NP-hard. If we had the absolute angles between all pairs of nodes, then the embedding is determined up to scaling. It is also straightforward that if both edge lengths and absolute angles are given, an embedding can easily be reconstructed. If, however, the input in both length and angle measurement is arbitrarily small but positive, then the problem becomes NP-hard again [21]. All the hardness results using angle information are a variation on the original recognition proof [29] using a reduction from 3SAT.

One can attempt to assess where the conceptual difficulties arise in the embedding problem. A key issue certainly stems from the fact that the graph metric (of the nodes) greatly deviates from the target metric (of the points, i.e., the embedded nodes). In other words, there exists an UDG such that the shortest path in the graph connecting two nodes needs k hops while the nodes need to be embedded such that their Euclidean distance is only slightly greater than 1. In fact, k can be as large as $\Theta(\sqrt{n})$, n being the number of nodes.

In order to construct an example where this is the case, it is enough

[2] that is, the magnitude, but not the sign of the angle is known

G and	edge lengths	edge angles	in
[29]	-	-	NP
[7, 14]	✓	-	NP
[31]	-	absolute	NP
trivial	✓	absolute	P
[21]	✓	relative	NP
[21]	ϵ	δ-absolute	NP

Table 2.1: Hardness of embedding a unit disk graph G with extra information. Absence of information is indicated by a - and presence by a ✓. The ϵ and δ refer to the error in the input information and can be arbitrarily close to 0.

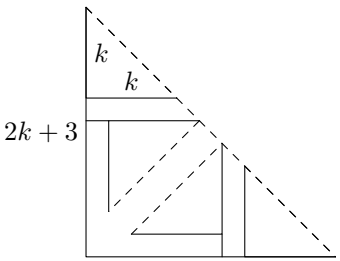

Figure 2.1: Recursive construction of a full tree. The small inner triangle trees have depth k, the outer one $2k + 3$. The root of the trees is the corner of the triangle.

to look at a tree which can be realized with all vertices placed on a grid. Consider the recursive construction schematically shown in Figure 2.1. We build a tree T_k of depth k out of four trees of depth $(k-3)/2$. (The tree could be mirrored to all four quadrants, but we show only one for simplicity.) From a simple area observation, the total number of nodes in T_k is in $O(k^2)$. We are concerned with the number of leaves $L(k)$ which are at the maximum depth k in T_k. By construction, $L(k) = 4L((k-3)/2)$, which, up to some tedious factors[3], solves to give $L(k) = \Theta(k^2)$. Thus both the total number of nodes and the number of nodes which have graph distance k are in the order of k^2. Since the area in which to embed the nodes of a tree with depth d is in $O(d^2)$, there is necessarily a leaf node u from the root v with $d_G(u,v) = k = \sqrt{n}$ such that $\|r(u) - r(v)\| = O(1)$ in any realization r of T_k. We can phrase this in general terms to have the following.

Lemma 2.1. *There are unit disk graphs $G = (V, E)$ with $|V| = n$ such that in any realization r of G, there exist $u, v \in V$ such that*

$$\frac{d_G(u,v)}{\|r(u) - r(v)\|} = \Theta(\sqrt{n}) .$$

The question now is whether the discrepancy between the embedded and the graph metric is the only hurdle. It seems not. A glance at the construction in the NP-hardness proofs shows that the graphs there are not dense in the way that the above trees are. To further investigate this issue, we can define a subtype of unit disk graphs which a priori eliminates the problem in Lemma 2.1. Consider unit disk trees with the following property. Let $V(v,k) = \{u \in T_v \mid d_G(u,v) = k\}$ be the set of nodes at depth k in the subtree rooted at v. Then the unit disk tree $T_r = (V, E)$ is a *sparse UDT* if

$$\forall\, v \in V \quad \forall\, 1 \le k \le \mathit{depth}(T_r) : \quad |V(v,k)| \le c \cdot k$$

for some constant parameter c. The motivation for such a definition is that there is enough room for all the nodes on the ring of radius k around a node v. We have not been able to find an algorithm to embed even sparse UDTs with constant quality, where the crux of the problem lies in proving that any DFS arrangement of the nodes gives a constant-ratio embedding. Thus, although it is not a concrete statement as Lemma 2.1, it seems that the *relative arrangement* of the subtrees might also play a significant role in finding a good embedding. In summary, it appears that there are two major components in the hardness of finding a constant-approximation embedding algorithm for unit disk graphs: deducing the correct Euclidean lengths based on graph distance information and finding a consistent global arrangement. To worsen the picture, we have reason to believe that the embedding of unit disk trees is already NP-hard.

[3] since it is basically $L(k) = 4L(k/2) = 4^{\log k} \cdot L(1) = \Theta(k^2)$

2.3 Graph Drawing and Embedding

Mapping graphs into the Euclidean plane or other metric spaces has a rich history before the recent attention the ad hoc networking community has lavished on it. There has been the more practical approach of *graph drawing* which concerns itself with finding algorithms which embed a graph in two or three dimensional space[4]. In a public lecture about the MetaPost language, Donald Knuth said he was thrilled when he learned that there is an entire annual conference devoted to the sole purpose of drawing graphs (see `http://gd2006.org/` for the latest installment) and that it would be one of the more pleasant jobs to get paid for the study of how to make aesthetic visualizations of graphs. That is also why it is such a rich source for research: Defining the quality of a drawing is not always straightforward and many different aspects need to be considered. For instance, one could try to produce as little edge crossings as possible, or expose as many symmetries as possible.

For the interested reader, the books [41] and [74] can serve as an introduction to the topic of graph drawing. The sub-field of straight-line drawings is of particular interest for the embedding problem as it takes into account the lengths of the embedded edges (and non-edges). See also [37] for an overview of graph-drawing methods where the placement of nodes should reflect their measured or graph distances. A method which has proven popular in the recent form of network embedding is that of *spring embedders*, a heuristic which simulates the network as a physical system. Originally, as introduced by Eades in [45], the vertices are rings (generally: point masses) and the edges are replaced by springs. The spring of two nodes which are placed too closely will exert a repulsive force and an attractive force if two connected nodes are too far apart. Two popular implementations are the Kamada-Kawai [73] and Fruchterman-Reingold [53] layout algorithms. In [73] all node pairs are connected by a spring with rest length (and thus supposed optimum distance) proportional to the length of the shortest path connecting the two nodes. In [53], the springs are replaced by a more general *force-directed model*. An electrostatic force repulses all nodes from each other, balanced by an attractive force along edges. Since the forces between nodes do not need to be nature inspired, the energy of the system can be generalized to model a property of the desired final layout, such as in [39]. All these methods have in common that they find a solution iteratively by the numerical method of choice, ideally converging to the global optimum, with the distinct possibility of stranding in a local energy minimum.

Another approach which also inspired an ad hoc network embedding algorithm is based on the statistical method known as *multidimensional scaling* (MDS). This was first applied in [79] to graph drawing where vertices are placed such that their Euclidean distances approximate their graph theoretic

[4] note that this does not have to be Euclidean, a hyperbolic map of the Internet [104] has proven to be a nice visualization tool

distances, which falls exactly into the domain described below.

The more abstract approach is taken by the theory of (finite) *metric embedding* out of which many algorithmic applications were distilled. The object of study is general weighted graphs, or finite metrics, and the measure is the *distortion* between pairs of points, not our peculiar UDG quality measure. Let (S, ρ) denote a finite metric space (see also Section 5.1.1 for a detailed review of these definitions). An embedding $f : (S, \rho) \to (S', \rho')$ has distortion at most $\delta = \delta_c \cdot \delta_e$ if

$$\frac{1}{\delta_c} \cdot \rho(x, y) \leq \rho'(f(x), f(y)) \leq \delta_e \cdot \rho(x, y) \tag{2.3}$$

holds for all pairs $x, y \in S$. Intuitively, such an embedding has *contraction*

$$\delta_c = \max_{x,y \in S} \frac{\rho(x,y)}{\rho'(f(x),f(y))}$$

and expansion

$$\delta_e = \max_{x,y \in S} \frac{\rho'(f(x),f(y))}{\rho(x,y)}$$

giving a total distortion $\delta = \delta_c \delta_e$. Usually, one of the sides is fixed, i.e., $\delta_c = 1$ covers non-contracting embeddings. Later on, in Chapter 5, we will see how this definition extends to encompass not only pairs, but any subset of points.

We will not even attempt to give an overview about this fascinating and challenging research area and refer the reader to the book chapters [100] or [67]. Instead, we will briefly summarize some main results and application areas. Although much of the classical literature on metric embeddings is not especially concerned with the dimension of the embedded space, a noteworthy and surprising result is the Johnson-Lindenstrauss Flattening Lemma [72]. It states that any n-point Euclidean metric can be embedded into Euclidean space of dimension $O\left(\varepsilon^{-2} \log n\right)$ with distortion $(1 + \varepsilon)$ for a given $\varepsilon \in (0, 1]$. If we do not place any restrictions on the dimension but are only concerned with embeddability into Euclidean space, then another famous set of results states that any n-point metric space can be embedded in l_2 with distortion $O(\log n)$ (Bourgain [28]) and a matching lower bound is given by Linial, London and Rabinovich [96] a decade later. The latter also emphasizes how metric embeddings can play an important role in algorithmic results, in particular for graph theory.

When it comes to the virtual coordinates problem, we can view these related areas as a source of inspiration, but the problem remains that the measures of success are different. As the discussion of Section 2.2 showed, the discrepancy between the graph metric and the target Euclidean metric can be insurmountable while a realization of the unit disk graph with ideal quality exists. Stated another way, if we embed the graph metric into two-dimensional Euclidean space, then the best possible distortion by Equation

2.3 can become very bad, while an embedding which gives perfect quality according to Definition 2.3 does exist. Further, most of the traditional literature is not concerned with two or other low-dimensional embeddings, emphasis has been placed more on the properties of the target metric. Only recently a trend of low-dimensional embeddings is emerging [15, 13].

2.4 Approaches to Localization

The problem of localizing an ad hoc network has not been overlooked by the recent boom of general wireless networking research literature. Consequently, a vast number of approaches tackling either the positioning or the virtual coordinates problem or both have been suggested. The majority of these are heuristic in nature, or give reasonable answers only for certain well-behaved types of input graphs. We will attempt to give an introduction into the current state of knowledge of the topic broadly described by the term *localization*, albeit from an algorithmic perspective and disregarding some results that fall more into the domain of routing rather than embedding. There are two points of view to keep in mind when discussing the localization problem in wireless networks. One direction is interested in finding a best possible unit disk or other graph embedding, while another line of research tries to find some coordinates which can be used in conjunction with routing and where emphasis is placed on finding fast effective heuristics. Another distinction we will make is, when unit disk graph embedding is considered, how much additional measurement information an algorithm assumes, such as edge lengths or angles.

2.4.1 Convex Constraints

One class of algorithms is what we label the *convex constraints* or linear programming approach. One of the earliest papers to propose solving the position estimation problem by convex constraints is [42], and later work, discussed below, produces more fine-grained relaxations. If we formulate the UDG embedding problem, with or without anchors, as a set of constraints which the variables (node positions) have to obey, then these constraints are not convex, thus we need an appropriate relaxation. The relaxations of the distance and angle constraints given in [42] yield a set of convex constraints, but their method works well only with anchors nicely placed on the outer perimeter of the network. This approach is explored further in [26] in regard to the question of when a network has a unique realization. Assume that there are $m > 0$ anchors and n sensor nodes. If at least $2n + n(n+1)/2$ distance pairs are known and all other reasonable constraints are feasible, then there is a unique solution and a semi-definite programming approach will find it. Note that the anchors are necessary as the above number is greater than $\binom{n}{2}$. In a follow-up paper considering anchor-free realization

(i.e., virtual coordinates instead of positioning) but in three dimensions [25] the authors observe that no similarly strong claim can be made in the case of $m = 0$. Unique realizability, or global rigidity, is further explored in [6, 137]. In [6], the authors study the conditions for when a network with anchors has a unique realization. Based on rigidity theory, they formulate when a point formation is generically globally rigid, which is equivalent to the corresponding graph having a unique realization. This is extended by [137] with a semidefinite programming approach.

The next set of linear programs for UDG embedding worked with additional information. In addition to showing that it is NP-hard to embed a UDG even if the absolute angles between edges is given (see Section 2.2), [31] also gives a linear-programming-based heuristic embedding for that case. The variables are the edge lengths and the constraints, in addition to bounding the edge and (some) non-edge lengths from above and below, respectively, are formulated for cycles and crossing edges. The embedding is then constructed via the computed edge lengths and the given angle information. Simulations in [31] show that this heuristic produces good results, yet an analysis giving bounds on the error which this embedding produces is missing. One could imagine that for tree networks where the cycle and crossing constraints are not applicable the embedding will produce less satisfactory results.

More concrete results can be achieved if both angle and distance information on the edges is given, albeit with noise. Although [14] considers general graphs and embedding not only into two-dimensional l_2, but also l_1 and l_∞, their results are applicable to UDGs. The input they consider is all edge lengths with multiplicative error ε and additive angle error γ; that is, two nodes u and v are connected by edge e with $d_{uv} \leq \|e\| \leq (1+\varepsilon)d_{uv}$ and the absolute angle to a reference line is between $\alpha_{uv} - \gamma$ and $\alpha_{uv} + \gamma$. Assuming that an embedding exists within these error bounds, the information is enough to produce a set of constraints modelling the possible positions for all nodes. Fixing a node u, the length and angle of the edge to node v, along with the error values, defines a *feasible region* $FR_v(u)$ of v with respect to u. See Figure 2.2. Then the set of possible locations for a node v, its feasible region FR_v, is $FR_v = \bigcap_{u \in N(v)} FR_v(u)$. This can be formulated either as virtual coordinates, arbitrarily fixing one node, or with given anchors. Without additional information, this is best possible. We can relax the constraints for $FR_v(u)$ to give a convex region by adding the line from a_1 to b_1. If we consider the computed embedding, then the error in the angle remains α_{uv}, the maximum distance between u and v remains $(1+\varepsilon) \cdot d_{uv}$, and the minimum distance is $d_{uv} \cdot \cos\gamma$ at point c in Figure 2.2. Altogether, the relaxation to convex constraints results in an embedding with additive angle error γ and multiplicative distance error ε' with

$$\varepsilon' \leq \frac{(1+\varepsilon) \cdot d_{uv}}{d_{uv} \cdot \cos\gamma} - 1 = \frac{1+\varepsilon}{\cos\gamma} - 1. \tag{2.4}$$

To further obtain a set of linear constraints for faster computation, the upper arc needs to be replaced by a piece-wise linear function. Using a chain of $2k$, $k \geq 1$, equal-length segments, the farthest point is at the end of the first tangent where the angle is γ/k, giving the longest distance to be $(1+\varepsilon) \cdot d_{uv}/\cos(\gamma/k)$. This is depicted by point d in Figure 2.2 for $k = 1$. The distance error is now

$$\varepsilon'' \leq \frac{1+\varepsilon}{\cos\gamma \cdot \cos(\gamma/k)} - 1 = \frac{1+\varepsilon}{\cos\gamma} - 1 + O\left(\frac{\gamma^2}{k^2}\right). \tag{2.5}$$

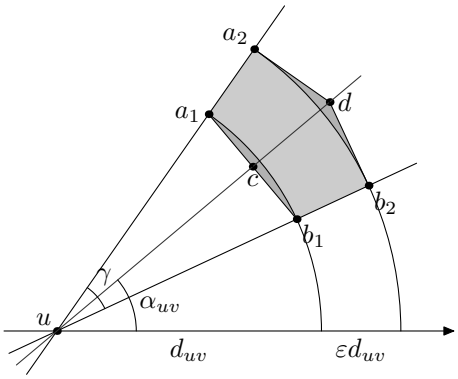

Figure 2.2: Feasible region of v with respect to u (lightly shaded) and its relaxation (additional darker shade).

Inspired by the above work, and as a follow-up to [31], the authors of [21] consider the embedding problem not one of computing point locations but entire regions of possible point locations for each node. The input is again such that distance and angle information is given on the edges, with possible noise. Since the feasible regions are not convex, they start out with considering only linear, convex shapes encompassing the non-convex feasible region. These can be trapezoids, or even axis-parallel rectangles. Then the authors define two types of "deployment regions" for each node. The *weak deployment region* W_i for node i is the maximal set of points W_i such that

$$\forall p \in W_i \; \forall j \neq i \; \exists q \in W_j \text{ constraints for } p \text{ and } q \text{ are consistent.} \tag{2.6}$$

Now fix a convex shape. The *strong deployment region* (of fixed shape) S_i for node i is the maximum-sized shape such that

$$\forall p \in S_i \; \forall j \neq i \; \forall q \in S_j \text{ constraints for } p \text{ and } q \text{ are consistent.} \tag{2.7}$$

Strong deployment (recall that we have a fixed shape) can be reduced to weak deployment by considering only the corners of that shape and taking the

intersection of the resulting feasibility regions. By construction, both cases can be solved efficiently by linear programming. [21] provides a distributed version instead of the mere formulation of the linear program and shows that it converges to the global solution after a finite number of iterations. The motivation for defining the weak and strong deployment regions is to give an upper and lower bound, respectively, on the uncertainty or error of the position. Unfortunately, no bounds on the *size of the regions* nor on the number of iterations are given analytically, which would provide insight beyond that of Eqs. (2.4) and (2.5), originally stated in [14]. The evaluation by simulations, however, show that already an angle range of $\pi/4$ provides considerable improvement over using only (noisy) distance information.

So far, to the best of our knowledge, the only algorithm to provide guarantees on the error bounds for unit disk graph embedding without any further information is given by [83] (correcting the earlier version [102]). Chapter 5 is devoted to presenting its details. A linear program on the edge lengths is only a small part of the solution, but nonetheless an essential step.

2.4.2 Global Structure

The physical models used in graph drawing have also found their way into network embedding. This class of algorithms lets the nodes iteratively compute their positions by responding to the forces exerted by a node's neighbors. In order for such an approach to work well and avoid local energy minima, it helps if the nodes start in initial positions already reflecting an optimal arrangement (see also [118] for a discussion of this phenomenon). Therefore, the algorithms based on the idea of simulating physical systems necessarily involve a two-phase process: first computing (parts of) a global structure of the graph, and then using a force-directed model for an iterative refinement of the nodes' positions. Below we will describe how the two popular physical models from graph drawing, the Kamada-Kawai and Fruchterman-Reingold layout methods, have been applied[5] to network embedding. The algorithmically interesting part is how they differ in the first, structural phase.

The work in [122] tries to pin down the global structure by deducing and placing the perimeter nodes of the network. The remaining nodes are placed loosely in the style of the spring embedder idea. The heuristic to detect the perimeter nodes works by choosing a broadcast node b which floods the network; if a node v has the greatest (or equal) hop distance to b within its two-hop neighborhood, then it sets itself as a perimeter node. The graph distances between all pairs of perimeter nodes are then used to approximately place the nodes such that the graph and embedded distances match as closely as possible. The perimeter node detection works well in dense graphs but is not amenable to worst-case analysis. One such problematic case is a tree

[5] independently but almost simultaneously in publication date

with a lot of little single-node branches along a path, such that the leaves of the short branches will consider themselves perimeter nodes. If we extend the search region to include the three-hop neighborhood, then we can also extend the branches by one hop. Another degenerate case is the ring where only a single node (opposite the beacon) claims to be a perimeter node.

Anchor-free localization (AFL) of [118] also tries to find the perimeter, but only four corners of the network plus a center node. Its refinement step is in analogy with the force-directed model of [53]. In contrast to the above implementation, AFL relies on distance measurements between neighboring nodes, it can thus also be used in a non-UDG context. The four periphery nodes n_1 through n_4 are computed using hop distance relations. Ideally, the goal is to have the lines n_1-n_2 and n_3-n_4 reflect two orthogonal diameters of the network. As before, we have an arbitrary beacon node n_0. Now n_1 is chosen such that it has maximum hop distance to n_0, breaking ties by node ID. Then n_2 is chosen farthest away from n_1 by the same principle. To find the other "diameter", n_3 is chosen to minimize $|d_G(n_1, n_3) - d_G(n_2, n_3)|$, with tie breaking by maximizing $d_G(n_1, n_3) + d_G(n_2, n_3)$ (and presumably ID again). Now n_4 is chosen so as to again minimize $|d_G(n_1, n_4) - d_G(n_2, n_4)|$, but ties are broken to maximize $d_G(n_3, n_4)$. The center node is chosen to minimize the difference between both pairs of opposing nodes. The result of this computation is to approximate a coordinate system such that a node v's polar coordinates are given by

$$\rho_v = d_G(v, n_5) \qquad \theta_v = \tan^{-1}\left(\frac{d_G(v, n_1) - d_G(v, n_2)}{d_G(v, n_3) - d_G(v, n_4)}\right).$$

The scheme is broken if the network does not have a "nice shape" and, in particular, the two main axes are missing. Consider constructing a graph out of a long path with dense clusters towards the ends, but no other nodes in the middle. Then n_1 and n_2 are chosen at the opposite ends of the path, but the remaining three reference nodes will cluster in the middle. Thus one direction of the network is correctly identified, but the other clusters degenerate onto the path.

2.4.3 Numerical Methods

The next set of algorithms for network localization uses numerical methods from other areas to find an embedding which tries to harmonize the target Euclidean and given graph distances. Recall that already in 1980 a graph drawing approach [79] tried to map graph to Euclidean distances using multidimensional scaling (MDS). Its "modern" version, applied to wireless network localization with or without anchors (and possibly including edge length measurements), is given in [136], along with its distributed updated version in [135]. The idea of the distributed approach is for each node to compute a map only of its local r-hop neighborhood and then the maps are

"glued" together (in [135] this is done by a linear transformation of the common nodes, minimizing the squared errors of the remaining nodes), one by one, until the entire network is mapped to the same coordinate system. The authors suggest using $r = 2$ as the improvement of using $r = 3$ is outweighed by the computational cost. The problem with this approach occurs again at sparse, non-regular networks. Consider a graph which is primarily a big cycle with some additional nodes. The iterative mapping procedure will produce a straight line until the last two maps are connected, at which point the quality of the embedding will become ruinous. The authors also propose an optional and computationally expensive global refinement step which then produces good results in simulation.

A method from the graph drawing community itself which falls in both this and the above category is [57]. Similar to the AFL algorithm, it has a two-phase process: First to try to find a fold-free layout, and then apply a better iterative refinement stage. It uses edge-length measurements and is specifically for unit disk graphs. The first part, however, is not based so much on the combinatorial structure but more on numerical methods so that we put the algorithm in this instead of the previous category. The idea to obtain a good initial layout is to have an embedding with an ideally minimum stress (energy) function, i.e., the difference between the embedded and measured distances for an edge is as close to 0 as possible. Since the goal is a fully distributed algorithm, using forces between non-neighbors is prohibitively expensive. Thus the authors design an energy function which is minimized by such a layout using a heuristically determined similarity measure w_{ij} only between adjacent nodes i and j as a function of their measured edge length. The global minimum of the energy function can be expressed in terms of the eigenvalues of a related matrix, and these can be approximately calculated in a distributed fashion. They also state that the number of iterations is in the order of the diameter of the graph. Evaluation is again by simulations, which show improvement over AFL as well as robustness to noise in the input data.

The main drawback of the above methods is that no analytical bounds on the resulting errors can be given. In the majority of the thus far described methods one or more parameters, such as w_{ij} above, exist which can be chosen freely and it is not clear whether they have an optimum value and if so, where that optimum lies and what its effect on the overall embedding has. Moreover, most of these methods work well in reasonable and usually (at least locally) dense graphs, as they are generated by a random process. To overcome these drawbacks, it is necessary to design more comprehensive simulation or real-world experiments or develop algorithms with a greater combinatorial component more effectively capturing the network layout.

2.4.4 Hop-Based Positioning

Moving on to purely positioning systems for wireless networks, there is a class of algorithms which we can loosely term *hop-based*. The reason for this is that the nodes determine their hop distance to the anchor nodes and base their position calculation solely on that. We give two prominent examples. The first is a set of algorithms collectively known as APS, or ad hoc positioning, first described in [107]. In its initial form, nodes know only connectivity information, that is, graph distances. The basic connectivity-only approach first finds the hop distance to all anchors (called landmarks). The anchors, based on the knowledge of the other anchors' positions, estimate the average length of a hop and propagate this to the nodes. Using these distance estimates a node can perform trilateration (much like in GPS) to calculate a position. Simulation results from [107] show that this scheme works well only under high-density conditions. DV-Hop needs more than 20% of the nodes to be anchors, only to stabilize at an average error of about a third of the radio range (unit distance). No data is given for less than 5% anchor nodes. These schemes have been extended to include information about edge lengths [109] and angles [108]. Interestingly enough, the same authors give calculations and simulations in [110] for APS in all combinations with angle and range measurements to show how large the error can be until the extra information becomes useless. For a specific set of network parameters, they give exact conditions on the uncertainty (standard deviations) which the angle and distance measurement hardware can have so that algorithms using either one or both of those measurements perform better than the hop-based approach alone. The majority of their parameter space, however, is occupied by the connectivity-only variant, motivating our study of this version of the unit disk graph embedding problem.

Another scheme is the Amorphous Computing system [105]. The idea is basically similar to APS in that the nodes determine their hop distances (called gradients) to the anchors (called seeds), but then use the Kleinrock-Silvester [76] formula to calculate the average length of a hop. A key ingredient in this formula is the global average density, measured in terms of nodes per unit disk. Simulation results here show satisfactory performance at node densities of more than 15 nodes per unit disk, but it suffices to have only one tenth of the nodes to be anchors.

The power of hop-based systems is the theme of Chapter 3, based on [24, 113], where we will explore how well knowing only the hop distances compares to centralized algorithms knowing the entire connectivity topology. It turns out that when the networks are not uniform and highly dense anymore, using only hop information can lead to arbitrarily bad performance. In one dimension, such as a model of a highway or other long road, it suffices to know only a little more information to achieve optimal performance.

In fact, these type of hop-based systems can be used as an initial step,

computing a first rough layout, and later refining the nodes' positions by an iterative process, much like the two-phase approach of the physical systems of Sections 2.4.2 and 2.4.3. These iterative steps are the subject of the following section.

2.4.5 Iterative and Other Positioning

In parallel to the spring embedder ideas, several papers have proposed iterative procedures of refining the coordinates based on updates in the neighbors' positions. The concept of *refinement* was used in [129]. The main idea there is to iterate the position estimation process. Once the nodes have an estimate of their position along with a confidence or error interval (which starts at 0 at the anchors), the information is exchanged among neighbors and the positions are recomputed.

A sizeable list of papers on network localization that we do not discuss remains. The majority of them is either along the same lines as what has so far been described, or contain one or more ingredients of the methods detailed above. A significant portion of the early location system literature focuses on single-hop systems where each node can hear a minimum number of powerful anchor nodes. Even the idea of a mobile robot serving as the anchor has been proposed. We have also omitted any discussion of hardware papers, the most famous of which are probably the Cricket location system [117] and RADAR [16]. Furthermore, distance embedding has also become popular for the Internet [106, 38] where inter-host latencies are mapped to the two or higher-dimensional plane. This can be seen as an application or extension of the metric embedding work for general graphs.

2.5 Roadmap

Concluding this introductory chapter, we provide an outline of what is to come in the remainder of the first part of the thesis. First, we will examine the problem of positioning in more detail with emphasis on what we call hop-based algorithms. The main drawback of iterative systems which use any type of refinement concept is the lack of an analysis bounding the number of steps needed to achieve a certain maximum error. Therefore, we look at algorithms which are essentially only given one shot: Obtain the necessary connectivity information in time proportional to the graph distance from the anchors and determine the position based solely on that. This is the theme of Chapter 3 which is based on [24, 113], but containing a more detailed analysis. We can also see this chapter more in line with the unit disk graph embedding problem by observing that it is all about determining more about the local structure of a UDG. A one-shot positioning algorithm is then merely an application of that.

We then make a quick excursion in Chapter 4 to take a glimpse outside the ivory tower of theoretical analysis and return with two results of real-world sensor networks. The positive one is that there does exist some correlation between real data and theoretical models. The negative one states that this correlation is not nearly enough to produce accurate embeddings. These results were originally published in [111] based on hardware from 2004 and thus one would expect current and future technology to show a significant improvement.

Returning to the safe terrain of abstract mathematical models, we give, to the best of our knowledge, the first and thus far only non-trivial poly-logarithmic approximation algorithm for the UDG embedding problem in Chapter 5, originally formulated in [102] but containing the major revisions of [83].

Inspired and humbled by the measurement results, in Chapter 6 we look at alternative wireless network models and discuss how the UDG can be generalized and examine when our proposed algorithm still works with the poly-logarithmic approximation ratio guarantee, giving some perspective on our results.

Chapter 3

Fast Positioning

In order to approach unit disk graph embedding, we first look at the subproblem of positioning where a part of the embedding is revealed in the form of anchors with known absolute coordinates. The most popular positioning heuristics in the literature contain an iteration phase in which the error is (potentially) successively minimized. The problem with this approach is that no provable guarantees are provided, neither with respect to runtime nor accuracy. Since one goal in wireless ad hoc network algorithm design is to have fast, distributed algorithms, we look at positioning from a one-shot perspective. The nodes are allowed to collect information from the anchors in time proportional to their distance from the anchors and then base their coordinate computation on that information alone. Thus, for a definite time bound, we can give bounds on the resulting error. Of course, a subsequent refinement phase can still be employed which might improve the accuracy in average-case networks. From the analytical perspective, this view forces us to identify which structural properties of unit disk graphs contain information leading to better embeddings.

3.1 Preliminaries

Model In the positioning problem, the network is a unit disk graph and the goal is to compute an embedding f respecting the absolute positions of given anchor nodes $a_1, \ldots, a_k$, see Problem 2 and Definition 2.4. We adopt the customary concepts from distributed algorithms design. Since our primary concern is wireless networks, we consider only the local broadcast model in which a single message transmission is heard by all the neighbors of a node. In the synchronous message passing model, communication proceeds in rounds. One round consists of a node receiving and processing messages from all of its neighbors, and then sending a message. The time complexity is the number of rounds from start until completion of a given task. In the

asynchronous message passing model, messages can be delayed arbitrarily long and are received as events. In the analysis, we assume that the longest message travels for one time unit. The time complexity is then the maximum number of time units in a worst-case execution and scheduling scenario. For the message complexity of an algorithm, we count the maximum number of messages sent by any single node.

Notation Apart from the Euclidean distance between two points in $\mathbb{R}^d$ given by the l_2 norm $\|\cdot\|$, there is a distance in graphs independent of any embedding. The term *hop* is synonymous with edge. A *path* P of length k is a sequence of nodes $P = v_0 v_1 \dots v_k$ where $v_i \neq v_j$ for $i \neq j$ and $\{v_i, v_{i+1}\} \in E$ for $0 \leq i < k$. Recall that the *graph distance* $d_G(u, v)$ between two nodes $u, v \in V$ is the length of a shortest path between u and v in the graph G. We immediately get, for any realization r of G,

$$\|r(u) - r(v)\| \leq d_G(u, v) \tag{3.1}$$

as a direct consequence of the definition of unit disk graphs.

In our algorithm analysis, we will frequently make use of the set of nodes which are a given graph distance away from another node.

Definition 3.1. *The set of nodes at graph distance exactly h from node $a \in V$ is*

$$N_h(a) := \{v \in V \mid d_G(a, v) = h\}\,. \tag{3.2}$$

Typically, a will be an anchor node and, when it is clear from context, we will simply write N_h.

General Outline of Algorithms The positioning algorithms we consider in this chapter consist of two parts: the gathering of connectivity information and a local calculation which computes the position (the embedding $f(v)$) based on that. Roughly speaking, the graph information collected at v outlines an interval of possible positions for v and our algorithms take the center of that interval for $f(v)$ in order to minimize $E(f(v))$. The main difference then lies in the information gathering phase.

3.2 The HOP Algorithm

3.2.1 The Algorithm

Since the forthcoming algorithms will build upon the basic structure of a general hop algorithm which we term HOP, we will present it in perhaps greater detail than necessary. Algorithm 3.1 gives the pseudo-code of HOP. To start the algorithm, an anchor node a transmits the message $[a : pos(a),1]$; at every other node h_a is initialized to $h_a \leftarrow \infty$.

```
1: receive [a : pos(a),h]
2: if h < h_a then
3:    h_a ← h
4:    send [a : pos(a),h_a + 1]
5: end if
```

Algorithm 3.1: A simple hop-counting algorithm HOP at each node v, given as a response to a message receipt event in an asynchronous model.

We first show the general properties of HOP and later discuss the actual computation of the embedding $f(v)$ from h_a for a particular node v.

Lemma 3.1. *The* HOP *algorithm finds the graph distance h from anchor node a to node v in time h.*

Proof. We will use induction on the graph distance h. Say that the longest time it takes for any single message to travel from one node to another, including processing time, is 1 time unit. All nodes in $N_1(a)$ will eventually receive the transmission from a, thereby correctly setting h_a to 1. This will be at time $t_1 \leq 1$ from the point where a sends the first message. For the induction step, assume that all nodes in $N_{h-1}(a)$ have received their distance at time $t' \leq h-1$. Then, by the definition of $d_G(\cdot,\cdot)$, $v \in N_h(a)$ has at least one neighbor $u \in N_{h-1}(a)$ (and none in $N_{h-2}(a)$). Since all $w \in N_{h-1}$ transmit exactly $[a : pos(a),h)]$, v will receive this message at least once (from u) and set h_a to h. The transmission from u to v will take at most 1 time step so that v determines its distance by time $t \leq t' + 1 \leq h$. □

The message complexity of HOP depends on whether the execution model is synchronous or asynchronous. The description in Algortihm 3.1 is given for the asynchronous case. In a synchronous execution model, a node first receives all of its messages, processes them, and then sends its messages. For HOP this means that node $v \in N_h(a)$ will receive all the messages from its neighbors in $N_{h-1}(a)$ before sending a message itself.

Lemma 3.2. *In the asynchronous model, the* HOP *algorithm has message complexity $n-1$, where n is the number of nodes in the graph. In the synchronous model, message complexity is* 1.

Proof. For message complexity, recall that we look at the maximum number of messages sent by any node. If there are n nodes in the graph (including the anchor a), then the maximum h which a node v can receive initially in an asynchronous model is $n-1$. Thereafter, v will accept and transmit only lower-count messages, in the worst case (where all non-anchors are in $N_1(a)$) down until its own hop counter is at 1. Thus v sends a total of $n-1$ messages.

In the synchronous case, v first hears from nodes $u \in N_{h-1}(a) \cap N(v)$, taking v to be in $N_h(a)$. In the next round, v transmits a hop count of $h+1$

and never another messages in a later round since h reflects its true hop distance to a. □

Using HOP, a node v collects the hop information h_{a_i} from all the anchors a_i to which it is connected. In one dimension, letting $h = d_G(u, v)$, we can extend Eq. (3.1) to

$$h/2 < \|r(u) - r(v)\| \leq h \tag{3.3}$$

because we cannot compress two hops to length 1 or less on the Euclidean line. This associates two intervals $I^+_{a_i} = pos(a_i) + (h_{a_i}/2, h_{a_i}]$ and $I^-_{a_i} = pos(a_i) - (h_{a_i}/2, h_{a_i}]$ (for $h_{a_i} > 1$) with each a_i. The final interval at v is $I(v) = \bigcap_{a_i}(I^+_{a_i} \cup I^-_{a_i}) =: [l_v, r_v]$ and the position of v is given by

$$f(v) = (r_v - l_v)/2 \tag{3.4}$$

in order to minimize the maximum error given that, based on the information h_{a_i} at v, all positions in $I(v)$ are possible.

We postpone the discussion of higher-dimensional mid-points to Section 3.4.

3.2.2 Competitive Analysis of HOP

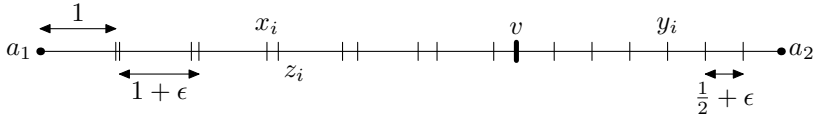

Figure 3.1: Instance of a UDG G where the HOP algorithm is significantly outperformed by an optimal algorithm.

We want to compare the HOP algorithm to an optimal one. To that end, we first need to define optimality which we do in terms of the minimum possible achievable error.

Definition 3.2. *Let f_{OPT} be an embedding of $G = (V, E)$ such that $E(f_{\text{OPT}}(V))$ is minimized. Let $f_{\text{ALG}}(P)$ be the positions computed by a positioning algorithm* ALG *for vertices $P \subset V$. Then the* competitive ratio *of* ALG *is c if*

$$E(f_{\text{ALG}}(P)) \leq c \cdot E(f_{\text{OPT}}(P)) + c'$$

for constants c, c' for any input unit disk graph G. We say that ALG *is c-competitive.*

Let D be the Euclidean distance between anchors a_1 and a_2. We construct an example where HOP's error is about $D/6$ for a node v and an optimal algorithm can determine v's position within one unit. In other words, we have the following.

Lemma 3.3. *The* HOP *algorithm is not competitive.*

Proof. Consider the unit disk graph $G = (V, E)$ with a realization r, depicted in Figure 3.1. Let h be the graph distance of a node v to both anchors a_1 and a_2. G has $n = 3h - 1$ nodes. There are h nodes that form the only shortest path from a_1 to v (excluding v), we call them $x_0 = a_1, x_1, \ldots, x_{h-1}$; there are h nodes from a_2 to v, $y_0 = a_2, y_1, \ldots, y_{h-1}$; and there are $h - 1$ nodes $z_1, \ldots, z_{h-2}, z_{h-1} = v$ for which $N(z_i) = \{x_i, x_{i+1}\}$ (for $i = 1, \ldots, h-2$), $N(v) = \{x_{h-1}, y_{h-1}\}$, and $z_i \in N_{i+1}(a_1)$. With $r(a_1) = 0$, $r(a_2) = D$, the actual coordinates are

$$r(x_i) = i \qquad r(y_i) = D - (\tfrac{1}{2} + \epsilon)i$$
$$r(z_i) = (1 + \epsilon)i \qquad r(v) = (1 + \epsilon)(h - 1)$$

for some small ϵ with $\frac{1}{h-1} > \epsilon > 0$. This means that $D = (h-1)(1+\epsilon) + h(\frac{1}{2} + \epsilon) = \frac{3}{2}h + ((h-1)\epsilon - 1)$.

The HOP algorithm will receive the information $r(a_1) = 0$ and $r(a_2) = D$ as well as $h_{a_1} = h_{a_2} = h$. Let f be the (partial) embedding computed by HOP. By the symmetry of the hop information, any hop-based algorithm minimizing the maximum possible error will put $f(v) = D/2 = \frac{3}{4}h + \frac{1}{2}((h - 1)\epsilon - 1)$. The error of HOP is

$$E(f(v)) = \|r(v) - f(v)\| = \tfrac{h}{4} + \tfrac{1}{2}((h-1)\epsilon - 1) > \tfrac{h}{4} - \tfrac{1}{2} \qquad (3.5)$$

or almost $D/6$.

An optimal algorithm will be able to deduce that $\|r(z_i) - r(z_{i+1})\| > 1$ from the connectivity information that $\{z_i, z_{i+1}\}$ is not an edge in G. Thus also $\|r(z_1) - r(z_{h-1})\| = \|r(z_1) - r(v)\| > h - 2$. Since $a_1 \notin N(z_1)$, the optimal algorithm can conclude that $\|r(a_1) - r(v)\| > h - 1$. Using the full connectivity information we can deduce that $h - 1 < r(v) \leq h$. Placing v anywhere within that interval will lead to $E(f_{\text{OPT}}(v)) < 1$. Therefore,

$$E(f(v)) > \frac{h}{4} - \frac{1}{2} > E(f_{\text{OPT}}(v)) + \frac{h}{4} - \frac{3}{2}$$

which is unbounded as $h \to \infty$. □

Note that although the counter example against HOP is one-dimensional, the fact that HOP is not competitive holds for all dimensions.

3.3 The HS Algorithm

Based on the bad performance of HOP, we can ask whether there exists a better efficient algorithm at least for the one-dimensional case.

3.3.1 Preliminaries

In order to improve the naive algorithm based on the above observations, we will introduce the notion of a *skip*.

Definition 3.3 (Skip). *For a graph $G = (V, E)$, two nodes $u, w \in V$ form a* skip *if $\{u, w\} \notin E$ and $\exists v$ such that $\{u, v\}, \{v, w\} \in E$.*

Equivalently, $d_G(u, w) = 2$. In other words, a skip is a little longer than a hop.

Definition 3.4 (Skip Distance). *A sequence of nodes $SP = v_0 v_1 \dots v_k$ is a* skip path *of length k if*

(i) $\{v_i, v_{i+1}\}$ *is a skip for* $0 \leq i < k - 1$,

(ii) $d_G(v_0, v_i) < d_G(v_0, v_{i+1})$ *for* $0 \leq i < k$, *and*

(iii) $\exists u_i$, $0 < i \leq k$, *such that either* $P = v_0 u_1 v_1 \dots v_{k-1} u_k v_k$ *or* $P' = v_0 u_1 v_1 \dots v_{k-1} v_k$ *is a path.*

The length of the longest skip path between $u, v \in V$ is the skip distance *$ds_G(u, v)$ between u and v in the graph G.*

The motivation behind the idea of skip distance is to impose a better lower bound on the distance between two nodes on the Euclidean line. Note that this is why we take the *longest* path as opposed to the shortest path as in the definition of graph distance. Further, while the use of skips only makes sense in the one-dimensional UDG case (see also Section 3.4), we have kept the definition general in terms of graphs. To warm up to the idea of skip distance, we conclude this subsection with the following lemma.

Lemma 3.4. *Let $h = d_G(u, v)$ and $s = ds_G(u, v)$. Then, in one dimension,*

$$\lceil h/2 \rceil \leq s \leq h. \tag{3.6}$$

Proof. If there is exactly one path from u to v, $P = u x_1 \dots x_{h-1} v$, and assuming for simplicity that h is even, then $SP = u x_2 x_4 \dots x_{h-2} v$ is a longest skip path. (If h is odd, it goes up to x_{h-1}). Any additional nodes can only lengthen SP. For a maximum skip path, there is at most one node $u_j \in N_j$ for $1 \leq j \leq h$ in the path for a total of h nodes from u_1 to $u_h = v$. The lemma now follows because two successive nodes in SP must be greater than one unit apart in one dimension (since, by the definition of a skip, they are not connected). □

```
 1: receive [a : pos(a), h | u, s] from neighbor x
 2: if h = h_a + 1 then
 3:    if u = x then
 4:       send [a : pos(a), h | u, s]
 5:    else if u ∉ N(v) then
 6:       send [a : pos(a), h | v, s + 1]
 7:    end if
 8:    s_a ← max{s_a, s + 1}
 9: else if h ≤ h_a then
10:    if u ∈ N(v) then
11:       send [a : pos(a), h + 1 | u, s]
12:    else
13:       send [a : pos(a), h + 1 | v, s + 1]
14:    end if
15:    if h < h_a then
16:       s_a ← s + 1
17:    else
18:       s_a ← max{s_a, s + 1}
19:    end if
20:    h_a ← h
21: end if
```

Algorithm 3.2: The HS algorithm at each node v, given as a response to a message receipt event in an asynchronous model.

3.3.2 The Algorithm

Algorithm 3.2 gives the pseudo-code of HS, a hop-skip algorithm. To start the algorithm, an anchor node a transmits the message $[a : pos(a), 1 \mid a, 0]$. Every other node initializes $h_a \leftarrow \infty$ and $s_a \leftarrow -1$.

Theorem 3.5. *In one dimension, the* HS *algorithm finds the graph and skip distances, h and s, respectively, from anchor node a to node v.*

Proof. Observe that the basic structure of the HOP algorithm is kept (Lines 1, 9, 11/13, 20) and merely augmented to include skip information.

To prove the correctness of the skip distance, we will use induction on the number of hops h as well. We claim that a node v at distance h will eventually know its correct hop and skip counts.

Going from $h - 1 \rightarrow h$, we assume that all N_{h-1} nodes will know their correct hop and skip distances. By Lemma 3.1, we know that then the N_h nodes will learn their hop distance as well. Based on that, we claim that the N_h nodes will obtain their correct skip distance. There are two things we need to show: (i) that s will not be erroneously too large and (ii) that it will be as large as it is supposed to be. The crucial property of one-dimensional unit

disk graphs is that increasing hop count corresponds to increasing Euclidean distance, thus we always make progress (away from the anchor) with each new hop.

First, observe that we can ignore all s_a values *before* the time that a node sets the correct h_a value since at that point s_a is reset (Line 16). Since we know that the nodes in N_{h-1} and N_h eventually obtain their correct hop distances, we will consider only the messages sent after that point.

The problem with (i) is that we need to show that a valid skip counter cannot travel away from a and then back towards it, falsely incrementing itself in the process. Line 9 makes v consider only messages from nodes in N_{h-1}. Line 2 allows messages from nodes in N_h. Observe that all nodes in N_h (on the same side of a) are neighbors, otherwise they would be farther or closer from a. We have to distinguish the two possibilities in the upper if-statement. If v forwards the message in Line 4, then any receivers in N_h will ignore the message (since $v \neq u$ and $u \in N(w)$ for receiving node $w \in N_h$) and only the legitimate receivers in N_{h+1} consider it. If, on the other hand, v has changed the message (legally, since u is at distance $< h$) in Line 6 and subsequently another node $w \in N_h$ has picked it up, then we are back on Line 4 and all the next nodes in N_h will drop it. Observe that this also guarantees the termination of the algorithm, since eventually all lesser-hop nodes will have sent off their messages and same-hop nodes will ignore irrelevant information after two passes.

We turn to resolving issue (ii). We restate the induction claim to say that, for any node $u \in N_h$ and for any valid skip path $u_0u_1 \dots u_{k-1}u$, then u will send the correct message (depending on whether $\{u_{k-1}, u_k\}$ is an edge or not). For the base case, observe that as in Lemma 3.1 we know that all N_1 nodes will eventually hear the message from a, setting their skip count to 1 and forwarding $[a, 0]$ (ignoring the first part of the message). Now consider a node $v \in N_h$ which has skip distance s. Any skip path $SP = av_1 \dots v_{s-1}v$ will have either $P = au_1v_1 \dots u_{s-1}v_{s-1}u_sv$ if $\{v_{s-1}v\}$ not an edge, or $P' = au_1v_1 \dots u_{s-1}v_{s-1}v$ as its associated path. In the case of P, we know that $v_{s-1} \in N_{h'}$ for $h' < h$. Thus, by the induction hypothesis, v_{s-1} eventually sends a message with $[v_{s-1}, s-1]$ which is forwarded by node u_s and v sets $s_a \leftarrow (s-1)+1 = s$ either in Line 8, 16, or 18 as these are the only possible cases. Since $v_{s-1} \notin N(v)$, v then sends $[v, s]$ in Line 6 or 13. In the case of P', we know that $\{v_{s-2}, v\}$ is not an edge and thus $v_{s-2} \in N_{h'}$ for $h' < h$. Again by the induction hypothesis, v_{s-2} sends a message with $[v_{s-2}, s-2]$, received by u_{s-1} and forwarded as is (since $v_{s-2} \in N(u_{s-1})$), which is updated by v_{s-1} to $[v_{s-1}, s-1]$ (since, by definition, $\{v_{s-2}, v_{s-1}\}$ is a skip). Therefore, v sets $s_a \leftarrow (s-1)+1 = s$ either in Line 8, 16, or 18. Since $v_{s-1} \in N(v)$, it will forward this information by Lines 4 or 11. □

The following theorem states that the time complexity for the improved HS algorithm is only twice as much as that of the simple HOP algorithm.

Theorem 3.6. *After time $2h$, a node v at distance h has received a message with the correct hop and skip count in the one-dimensional* HS *algorithm.*

Proof. For the time complexity, we will again lean on our analysis of the simple HOP algorithm of Lemma 3.2. Let the maximal time unit be 1. We know that a path of length k takes at most k time to reach the last node. Thus it takes at most h time for node v to learn its hop distance h and at most $2h$ time to learn the skip distance s because the associated path is composed of at most $2h$ hops. □

The interval for v is now bounded by

$$s - 1 < \|a - v\| \leq h \tag{3.7}$$

and the position is again computed as the mid-point of the intersection of all such intervals.

3.3.3 Competitive Analysis of HS

We want to show that an optimal algorithm cannot perform substantially better than an algorithm which only knows the graph and skip distances h and s, respectively. Specifically, we will prove that our positioning algorithm is optimal (1-competitive) up to a small additive constant. As a stepping stone for the main proof we first study the case of one anchor node.

Lemma 3.7. *Take a one-dimensional unit disk graph. Assume there is only one anchor node a and all nodes know they are to the right of a, that is, $r(v) \geq r(a)$ for all nodes v and realizations r. For the position f_{HS} of a node v as determined by the* HS *algorithm, we have*

$$E(f_{\text{HS}}(v)) \leq E(f_{\text{OPT}}(v)) + \frac{1}{2} + \epsilon$$

for all v and any $\epsilon > 0$.

In order to prove Lemma 3.7, we will need the following two lemmas.

Lemma 3.8. *Let $G = (V, E)$ be any one-dimensional UDG, $A = \{a\}$ with position $p(a) = 0$, and let $d_G(a, v) = h$ for a $v \in V$. Then, for any $\epsilon > 0$, there is a realization r of G such that $h - \epsilon \leq r(v) \leq h$.*

Proof. The idea of this lemma is to "stretch" the graph G to its maximum possible position at v. Let $N_h = \{v_0^h, \ldots, v_{n_h}^h\}$. Let the ordering be such that, in any realization, we have $r(v_{n_h}^h) \leq \cdots \leq r(v_1^h) \leq r(v_0^h)$, (i.e., v_0^h is the rightmost node in N_h). We can identify (the positions of) v_i^h with v_j^h if $N(v_i^h) = N(v_j^h)$ since they are indistinguishable from the combinatorial point of view. Renamed and relabeled, we have $r(v_{n_h}^h) < \cdots < r(v_1^h) < r(v_0^h)$. As noted before, all nodes in N_h are neighbors.

We will use induction on the number of hops h from anchor node a at $r(a) = 0$ to v. For $h = 1$ place the n_1 (different) nodes at positions $r(v_i^1) = 1 - i \cdot \epsilon_1$ for some sufficiently small $1 \gg \epsilon_1 > 0$, that is, $\epsilon_{i,1} = i \cdot \epsilon_1$ and $\epsilon_{i+1,1} - \epsilon_{i,1} = \epsilon_1$. See also Figure 3.2 for reference.

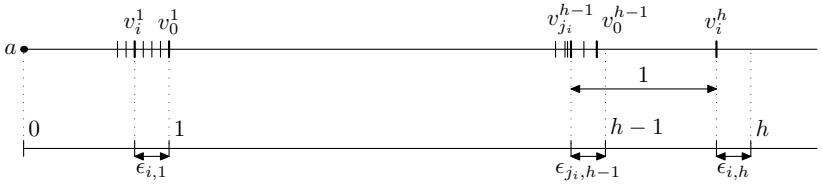

Figure 3.2: Schematic realization r of G from the proof of Lemma 3.8 on the top. The bottom line is a reference for the scale.

Assume now that we have placed all nodes up to N_{h-1} such that $r(v_i^{h-1}) = (h-1) - \epsilon_{i,h-1}$, and, with the labeling from above, $\epsilon_{i,h-1} < \epsilon_{i+1,h-1}$ for all $0 \le i < n_{h-1}$. In fact, let $\epsilon_{h-1} > 0$ be such that $\epsilon_{i+1,h-1} - \epsilon_{i,h-1} \ge \epsilon_{h-1}$ for $0 \le i < n_{h-1}$. For each of the v_i^h we consider the maximal (leftmost) j_i for which $v_{j_i}^{h-1} \in N(v_i^h)$ and $v_{j_i+1}^{h-1} \notin N(v_i^h)$. We start by placing

$$r(v_0^h) = r(v_{j_0}^{h-1}) + 1 = h + \epsilon_{j_0,h-1}$$

and for increasing $i > 0$ we let

$$r(v_i^h) = \min\left\{r(v_{j_i}^{h-1}) + 1, r(v_{i-1}^h) - \epsilon_h\right\} \tag{3.8}$$

in other words, $\epsilon_{i,h} = \max\{\epsilon_{j_i,h-1}, \epsilon_{i-1,h} + \epsilon_h\}$. The choice for ϵ_h is such that $0 < \epsilon_h < \epsilon_{h-1}/n_h$. By construction, we immediately have $\epsilon_{i,h} + \epsilon_h \le \epsilon_{i+1,h}$ for $0 \le i \le n_h$. It remains to be shown that all the neighbors of node $v = v_i^h$ in G are within distance 1 and only those.

Consider a $u \in N_h(a)$ which implies $u \in N(v)$. Then we can write $u = v_j^h$, meaning $r(u) = h - \epsilon_{j,h}$. Thus we get that

$$\|v - u\| = |\epsilon_{i,h} - \epsilon_{j,h}| < 1$$

by choosing all the $\epsilon_1, \ldots, \epsilon_h$ small enough.

For $u \in N_{h-1}(a)$ we can write $u = v_j^{h-1}$, $r(u) = h - 1 - \epsilon_{j,h-1}$, and get

$$\|v - u\| = 1 - (\epsilon_{i,h} - \epsilon_{j,h-1}) = 1 - \delta.$$

We distinguish two cases: (i) $u \in N(v)$ or (ii) $u \notin N(v)$. The first case (i) is equivalent to $j \le j_i$ by our choice of j_i. Since $\epsilon_{i,h} \ge \epsilon_{j_i,h-1}$ this implies $\delta \ge 0$ by the induction hypothesis and $j_i \ge j$. Thus, $\|v - u\| \le 1$.

Case (ii) is equivalent to $j > j_i$ and we again consider two possibilities. If $\epsilon_{i,h} = \epsilon_{j_i,h-1}$, then $\delta < 0$ by the same line of reasoning as above and $\|v - u\| > 1$. On the other hand, if $\epsilon_{i,h} = \epsilon_{i-1,h} + \epsilon_h > \epsilon_{j_i,h-1}$, then we can also write $\epsilon_{i,h} \leq \epsilon_{j_i,h-1} + i \cdot \epsilon_h$ by our numbering of the nodes, $j_i \geq j_{i-1}$. Then

$$\begin{aligned}\delta &\leq \epsilon_{j_i,h-1} + i \cdot \epsilon_h - \epsilon_{j,h-1} \\ &\leq i \cdot \epsilon_h - \epsilon_{h-1} \\ &\leq n_h \cdot \epsilon_h - \epsilon_{h-1} \\ &< 0\end{aligned}$$

where the second line follows from the induction hypothesis and $j > j_i$, the third line from $i \leq n_h$ and the last from our choice of ϵ_h. Again from $\delta < 0$ we have $\|v - u\| > 1$. □

Lemma 3.9. *Let $G = (V, E)$ be any one-dimensional UDG, $A = \{a\}$ with position $p(a) = 0$, and let $ds_G(a, v) = s$ for a $v \in V$. Then, for any $\epsilon > 0$, there is a realization r of G such that $s - 1 < r(v) \leq s + \epsilon$.*

Proof. Now we compress the graph as much as possible. We will again proceed by induction, this time on the number of skips s. Let $NS_s = \{w_0^s, w_1^s, \ldots, w_{n_s}^s\}$ be the distinguishable nodes at skip distance s from a and the order is such that $r(w_0^s) > r(w_1^s) > \cdots > r(w_{n_s}^s)$ in any realization, analogously to the proof of Lemma 3.8. Observe that their hop counts differ by at most one, and they are all neighbors, otherwise there would be a skip from $w_{n_s}^s$ to w_0^s.

For $1 \leq s' < s$, there always exists a pair $\{u, v\} \notin E$ such that $u \in NS_{s'-1}$ and $v \in NS_{s'}$ by the definition of skip distance. Set

$$v_{s'} = \max_i \{w_i^{s'} \mid \exists u \in NS_{s'-1} : u \notin N(w_i^{s'})\},$$

the leftmost node in $NS_{s'}$ which has a skip to a node in $NS_{s'-1}$. Let $l_{s'}$ be the index of $v_{s'}$ in $NS_{s'}$.

The positions for $s = 1$ are as follows. Set $r(v_1) = 1 + \delta_1$ for a small enough $\delta_1 > 0$. Then place

$$r(w_i^1) = 1 - (i - l_1 - 1)\epsilon_1 = r(v_1) - \delta_1 - (i - l_1 - 1)\epsilon_1$$

for $i > l_1$ and

$$r(w_i^1) = 1 + \delta_1 + (l_1 - i)\epsilon_1 = r(v_1) + (l_1 - i)\epsilon_1$$

for $i < l_1$ and small enough $0 < \epsilon_1 \leq \delta_1$. Then $r(w_i^1) - r(w_{i+1}^1) \geq \epsilon_1$.

We proceed by induction for $1 < s' < s$, assuming that $s' - 1 - \alpha_{s'-1} \leq r(w_i^{s'-1}) \leq s' - 1 + \beta_{s'-1}$ and $r(w_i^{s'-1}) - r(w_{i+1}^{s'-1}) \geq \epsilon_{s'-1}$. Find $v_{s'}$ and let

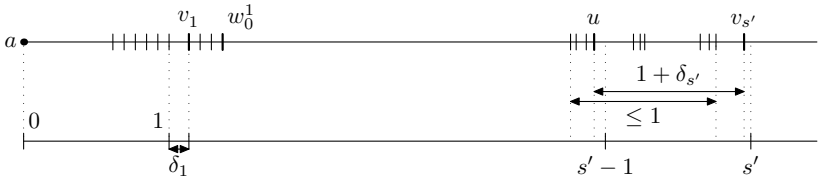

Figure 3.3: Schematic realization r of G from the proof of Lemma 3.9 on the top. The bottom line is a reference for the scale.

$u = \min_i\{w_i^{s'-1} \in NS_{s'-1} \mid w_i^{s'-1} \notin N(v_{s'})\}$, the rightmost non-neighbor of $v_{s'}$ to the left. Then

$$r(v_{s'}) = r(u) + 1 + \delta_{s'} \leq s' - \beta_{s'-1} + \delta_{s'}$$

and we know that $w_{n_{s'-1}}^{s'-1} \in N(w_i^{s'})$ for all nodes with $i > l_{s'}$ by the maximality of $l_{s'}$. Thus,

$$r(w_{l_{s'}+i}^{s'}) = r(w_{n_{s'-1}}^{s'-1}) + 1 - (i-1)\epsilon_{s'}$$

for $1 \leq i \leq n_{s'} - l_{s'}$. To the right of $v_{s'}$, let u_i be defined as above except for node $w_i^{s'}$, $i \geq l_{s'}$, instead of $v_{s'}$. Then

$$r(w_i^{s'}) = \max\left\{r(u_i) + 1 + \delta_{s'}, r(w_{i-1}^{s'}) + \epsilon_{s'}\right\}$$

for i from $l_{s'}$ to $n_{s'}$. Set $\epsilon_{s'}$ and $\delta_{s'}$ small enough such that $\delta_{s'} \leq \epsilon_{s'-1}$ and $\epsilon_{s'} \leq \delta_{s'}/n_{s'}$. Altogether, we get $\alpha_{s'} \geq \epsilon_{s'}$ and $\beta_{s'} < \beta_{s'-1} + \delta_{s'} + n_{s'} \cdot \epsilon_{s'}$.

As for the remaining connectivity relations, the nodes in $NS_{s'}$ are all neighbors by choosing the ϵ_i and δ small enough. The neighbors of nodes $w_i^{s'}$ for $l_{s'} \leq i \leq n_{s'}$ are guaranteed by construction. For $i < l_{s'}$, the non-neighbors are far away, also by construction and we need to show that all remaining neighbors are indeed within range. Here, the same reasoning as in the proof of Lemma 3.8 applies by our choice of $\epsilon_{s'}$ and $\delta_{s'}$ sufficiently small.

At skip distance s, we are not guaranteed the existence of a v_s with a skip to NS_{s-1} anymore. In that case, we set $r(v) \leq r(w_{n_{s-1}}^{s-1}) + 1 \leq s - 1 + 1 + \beta_{s-1} = s + \epsilon$ where $\epsilon = \beta_{s-1}$ can be made arbitrarily small by choosing the δ_i and ϵ_i appropriately. □

We are now ready for Lemma 3.7.

Proof of Lemma 3.7. Given a unit disk graph $G = (V, E)$, we can construct two realizations r_1 and r_2 of G such that $r_1(v) = h - \epsilon_1$ at the maximum and $r_2(v) = s + \epsilon_2$ at the minimum ($\epsilon_i > 0$). From the definition of skip distance, we know that $r(v) \geq s - 1 + \tilde{\epsilon}$ in any realization r. Therefore, an optimal

algorithm cannot distinguish between these extremes and is thus forced to return $\frac{h-s-1}{2} < r(v) \leq \frac{h-s}{2}$ which is off by less than $\frac{1}{2}$ from the position returned by HS who knows only h and s. □

Altogether, we end with the main result of this section.

Theorem 3.10. HS *is optimal in one dimension up to an additive constant.*

Proof. We have shown in Lemma 3.7 that Algorithm HS is optimal (up to an additive constant) whenever there is one anchor on a specified side of the nodes. It remains to be shown that this is the essential ingredient of the optimality of HS and that OPT cannot acquire too much more information in a general scenario. We do this by considering what happens when we add anchors to both sides of a node.

First, we argue that we can only lose a constant of at most 1 whenever there are anchors on both sides of v. In Lemma 3.7, we had considered the case of an anchor a to the left of v. If we place another anchor b to v's right, then we need to observe what happens when the two subgraphs "come together." Let the actual order of nodes from left to right be $a, u_1, \ldots, u_l, v, w_r, \ldots, w_1, b$, then the previous lemmas are applicable to the subgraphs of $V_a = \{a, u_1, \ldots, u_l, v\}$ and $V_b = \{v, w_r, \ldots, w_1, b\}$ independently (since they have no nodes in common except for v). The only problem which can occur is with nodes u_i and w_j which are within one hop of v. In this case, it could be that some of the u_i's are connected (or not connected) to the closer of the w_j, so that there needs to be a minor adjustment in v's position as well. Since this independence of the subgraphs is only violated at those nodes which are within one unit of the opposite subgraph, there will be an adjustment of at most one unit for the optimal position.

Next, we claim that multiple anchors to one side can again only shrink the interval by another additive constant of at most 1. To prove this, we will consider anchors a_i with given positions $p(a_l) \leq \cdots \leq p(a_1) < r(v)$ to the left of v, where again the coordinates increase to the right. Let $h_i = d_G(a_i, v)$ and $s_i = ds_G(a_i, v)$. Set

$$l_i = p(a_i) + s_i - 1 \qquad \text{and} \qquad r_i = p(a_i) + h_i$$

then the left and right boundaries of v's interval are

$$l = \max_i \; l_i \qquad \text{and} \qquad r = \min_i \; r_i$$

respectively. Note that we cannot have $l_i > r_j$ (i.e., the left boundary of a_i is to the right of the right boundary of a_j) for any i, j since otherwise the intervals of anchors a_i and a_j would not intersect, which is impossible (contradicting the assumption of the existence of a realization).

We claim that l_1 and r_1 are already good approximations of l and r (up to one unit). For the right boundary, consider the distances $d_i = p(a_1) - p(a_i)$

and $g_i = d_G(a_i, a_1)$. Then $h_i \geq g_i + h_1 - 1$ where the -1 is due to the fact that all shortest paths from a_i to v might not go through a_1 and would therefore be one hop less than if we take a detour through a_1. Altogether, using $p(a_i) = p(a_1) - d_i$, we get

$$\begin{aligned} r_i &\geq p(a_i) + g_i + h_1 - 1 \\ &= (p(a_1) + h_1) + \underbrace{g_i - d_i}_{\geq 0} - 1 \\ &\geq r_1 - 1 \end{aligned}$$

for all $i > 1$. The last inequality stems from the fact that the number of hops between two nodes is always an upper bound on their Euclidean distance.

The case for the left boundary is similar. Here, let $t_i = ds_G(a_i, a_1)$ and $s_i \leq t_i + s_1 + 1$. Then

$$\begin{aligned} l_i &\leq p(a_i) + t_i + s_1 + 1 - 1 \\ &= (p(a_1) + s_1 - 1) + \underbrace{t_i - d_i}_{\leq 0} + 1 \\ &\leq l_1 + 1 \end{aligned}$$

for all $i > 1$. Again, the skip distance is a lower bound on the actual distance and there might be a longer path circumventing a_1. Altogether, the interval bounds on each side can be decreased by at most 1 on each side, thereby increasing the maximum error of HS by at most 1.

Note that the same argument can be applied to anchors only to the right of a node v.

What remains is to consider the general case when v is located between several anchors to both sides. Altogether, we can consider the interval given by the rightmost anchor a_1 to the left of v and the leftmost anchor b_1 to the right of v, losing at most a constant of 1 unit. From the first claim, we know that if we have an anchor on both sides of v, then merging the two subgraphs results in the loss of at most another unit. We can conclude that the interval of HS compared to that of an optimal algorithm is greater by at most 2 in the general case, proving the theorem. □

3.4 The GHoST Algorithm

We now move on to higher dimensions. From Section 3.2.2 we know that we have to do more than the simple HOP algorithm in order to approach optimal position estimates. Moreover, HS does not apply directly, because in two or more dimensions, the minimum Euclidean distance for two nodes u and v separated by h hops is not $h/2$ anymore but merely 1, even for maximal skip distance. See for example Figure 3.4. The bad news is that if there

is no further information, then v has no way of determining whether it is slightly more than 1 or as much as h units away from u. The good news is that neither can an optimal algorithm so that the competitive ratio is not compromised in this case.

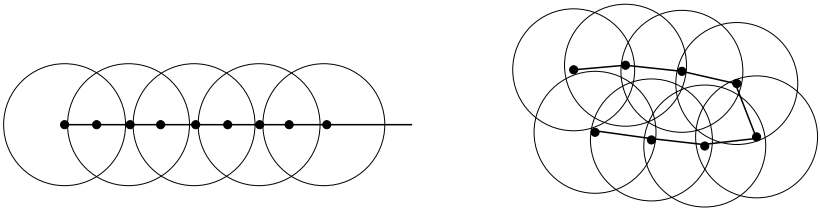

Figure 3.4: Minimum Euclidean distance between two nodes given their hop distance h. The circles have radius 1, the dots are the nodes. On the left is the one dimensional case where $\|\cdot\| > h/2$ (not all circles are shown). On the right, in two or more dimensions, only $\|\cdot\| > 1$ can be assumed.

Another issue is the construction of the "mid-point" of the interval intersections in two or more dimensions. Since we are studying the worst case, we want to find the point such that the *maximum error is minimized*. Going back to one dimension, one can consider the mid-point of a line segment as the center of the circle with the segment as its diameter. Similarly, in two dimensions, we can (locally) find the circle of minimal radius which encloses all points in the intersection, as in Figure 3.5. The center of that circle is then the point with least maximum distance to any other point in the area. In d dimensions, we find the smallest enclosing $(d-1)$-dimensional sphere.

The construction above and in the figure is actually not complete, since we still need to "cut out" a circle of radius 1 around the anchors which are more than one hop away. However, this has no influence on the argument above, since this still results in some interval of which we find the smallest enclosing ball (see, for instance, [52]).

3.4.1 Lessons Learned from One Dimension

The crucial insight of the 1-dimensional optimal HS algorithm was that there exist certain local structures in the unit disk graph (e.g. a skip) with which we can impose an upper or lower bound on the actual length of a hop. We will now survey some of these local structures. They can be classified into *stretchers* and *trimmers*. Stretchers and trimmers enforce a minimal and maximal length, respectively, on hops. For example, the skip was a stretcher in one dimension; enough to produce an optimal algorithm. In two dimensions, we have identified several trimmers:

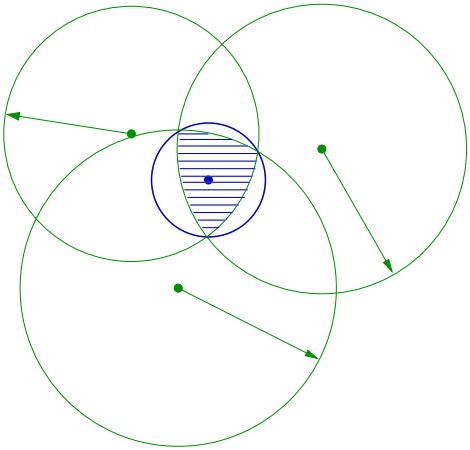

Figure 3.5: Construction of the "mid point" of a two-dimensional area. The outer circles of radius h_i (in green) represent the reach of each anchor (being h_i hops away). Their intersection is shaded in the center. The smallest circle enclosing the entire intersection area is depicted in the center (in blue). The center is the computed position.

- T_0 – a trimmer that considers hop paths of length 2. Let $P_v = uvw$ and $P_x = uxw$ be shortest paths from u to w. If $\{v, x\} \notin E$, then $\|r(u) - r(w)\| \leq \sqrt{3}$ in any realization r. See Figure 3.6.

- T_k – a generalization of T_0: Suppose that there are two shortest paths $P_v = uv_0 \dots v_k w$ and $P_x = ux_0 \dots x_k w$ connecting u and v with $\{v_0, x_0\}$ and $\{v_k, x_k\} \notin E$. For the remaining nodes, it is irrelevant whether $\{v_i, x_i\}$ is an edge for $0 < i < k$, but $\{v_i, x_j\} \notin E$ for $i \neq j$. Then $\|r(u) - r(v)\| \leq k + \sqrt{3}$ as opposed to $k + 2$ in any realization r.

- MT_{k_1,k_2} – a trimmer resulting from the merging of two paths from two different anchors. As an exemplary case, consider $MT_{1,1}$: two paths from anchors a_1 and a_2 that merge after just one hop at node u. Ignoring for the moment a constant adjustment (in the order of one unit), if the graph distance from the a_i to a node v is h, then $\|r(a_i) - r(v)\| \leq \sqrt{1 + (h-1)^2} = \sqrt{h^2 - 2(h-1)} < h$ for any realization r. The constant adjustment accounts for the orientation of m and v with respect to the a_i. An analogous computation can be made if the paths merge at u after k_1 hops from a_1 and k_2 hops from a_2.

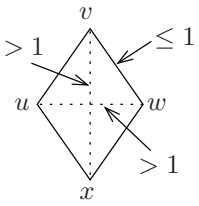

Figure 3.6: A trimmer for the path from u to w (and from x to v). The dashed lines indicate that there is *no* connection. With a simple geometric argument one can impose a maximum length on the distance of u to w.

3.4.2 The Algorithm

Based on the arguments of Section 3.4.1, we can formulate a General Hop Stretcher-Trimmer Algorithm (GHOST). The idea is that nodes examine their local neighborhoods – the details depend on which structures are considered – to extract the necessary information about existing trimmers and stretchers. When a node v receives a message with a shortest hop path from an anchor a, then it can incorporate its trimmer (stretcher) information and compute a path with maximum (minimum) actual length that is shorter (longer) than that of the received path. In some cases, other local structures might require more information such as including paths other than the shortest. In practice, one will have to make a trade-off between the efficacy of a configuration and the expense of its computation.

The affects of GHOST to time and message complexity are similar to those of HS. Let node v be h hops from anchor a. Once the nodes in N_{h-1} obtained their correct paths of length $h-1$, they send it on to nodes in N_h. In one time unit, v receives all those transmissions from neighboring nodes u in N_{h-1} and the tuples (u, P_u) will constitute (the necessary information about) all shortest paths to v. For message complexity, in the worst case a node has to receive all the information about shortest paths separately over the same link.

Observe that GHOST is actually more of a *framework* for positioning algorithms. The concrete algorithm is determined by which stretchers and trimmers are used. If structures are used which have provable bounds on the path lengths, such as T_k or MT_{k_1,k_2}, then the algorithm inherits these bounds and the maximum error is equal to or less than without them. On the other hand, if we use heuristic structures, then the resulting algorithm cannot provide worst-case guarantees anymore.

Another side effect of such a framework is that good distance bounds – obtained from physical measurements – can easily be integrated into GHOST: Instead of (or in addition to) computing the local structures resulting in

the lower and upper bounds h_l and h_u for a hop in the graph, the distance estimate can give us these values directly.

With the remarks of this section we can conclude the following.

Theorem 3.11. *In two dimensions, let f_{GHoST} be the positions computed by the* GHoST *algorithm with trimmers T_k and f_{HOP} those of the* HOP *algorithm. Then*

$$E(f_{\text{GHoST}}(v)) \leq E(f_{\text{HOP}}(v))$$

for all nodes v. Further, GHoST *has the same time complexity $O(h)$ as* HOP*, where h is the graph distance from an anchor node to the node in question.*

3.4.3 Simulation

The trimmers of Section 3.4.1 apply to any unit disk graph and therefore cannot increase the maximum error in relation to HOP. When no trimmers are present, then GHoST reduces to HOP. We want to investigate under what conditions the effect of local structures improve GHoST's accuracy.

In our simulations, we have implemented the simple HOP algorithm as described in Section 3.2.1 and the GHoST algorithm with the trimmer T_0 only. A screen-shot of the visual part of the application can be seen in Figure 3.7 on page 51. Our testing environment consists of an area of 20 by 20 units. We test random graphs for node densities (measured in the number of nodes per unit disk) ranging from 12 to 30 and anchor densities from 0.5 up to 10 percent of the nodes (creating up to almost 4000 nodes). For each combination we collect 300 position estimates along with the error for both HOP and GHoST.

Since we are interested the effect of the *improvement* of T_0 over HOP, we calculate the *average relative errors* of GHoST to HOP in Figure 3.8 (on page 52). The absolute errors of GHoST can be seen in Figure 3.9. The relative error is taken for each estimate separately instead of over the total average errors in order to gain a better understanding of how effective the trimmers are in individual situations. We see that GHoST improves the position estimate even in very low density (node and anchor) as well as in very high density situations. The most significant improvements can be seen for modest anchor densities (around 2.5%) and fairly high node densities (around 27). The improvement for low anchor densities is because then any positioning algorithm has very little information to work with and already a small amount of extra, useful information will lead to an improvement. The high node density improvement can be explained by the increased presence of trimmers which significantly reduce the upper distance bound.

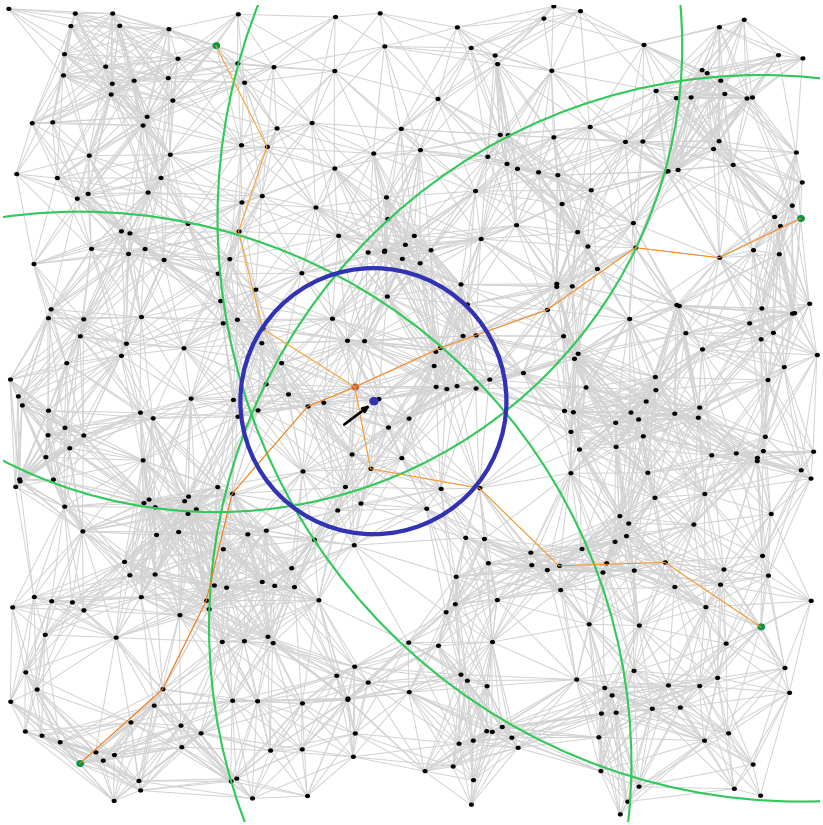

Figure 3.7: The visualization of the GHOST algorithm. The intersection of circles in the center is the area of all possible positions as calculated by the algorithm. The center of the circle (marked by the arrow) is chosen as the computed position which minimizes the maximum error.

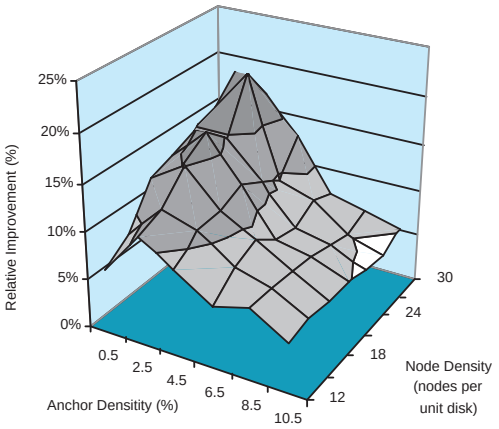

Figure 3.8: The graph shows the improvement of GHOST over HOP in the depicted anchor and node density ranges.

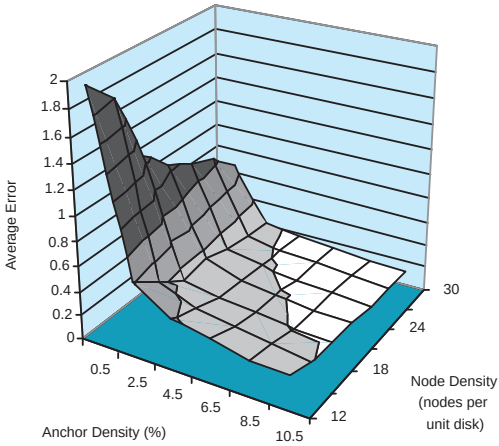

Figure 3.9: The graph shows the absolute errors of GHOST (in units of the radio range).

3.5 Discussion

Studying the UDG embedding problem from the one and two-dimensional positioning perspective has led us to see that there exist certain *local* structures which can help to bound the relative error. What is missing from this picture is a *global* view which can give bounds on the absolute error. We will turn to this aspect in the form of the virtual coordinates problem in Chapter 5.

Chapter 4

Real-World Problems

In this chapter we focus on determining the potential of using *minimal hardware requirements* for the task of positioning in a wireless sensor network. The question is how well can a node localize itself if we only have small, low-power, low-storage devices at our disposal. The answer will allow engineers of such a network to determine whether to invest in more specialized and thus bulkier and more expensive hardware, or if the position error is tolerable in their applications.

There has been some work on investigating the physical characteristics of real-world sensor networks [55, 139]. Our findings corroborate their conclusions (as in [55]) in that the link characteristics are far from the theoretical models in use, such as the unit disk graph or the quasi unit disk graph [89]. However, these works are either too general in nature in that they survey the detailed link stability but not its effect on positioning ([55] looks at flooding, [139] at routing). Or they only consider two nodes at a time, whereas our findings in Section 4.2 indicate that there are considerable differences between the experimental results if only two nodes are involved versus an entire network of nodes.

There has been a significant amount of research done on the theoretical side of positioning and virtual coordinates algorithms as previously discussed. The underlying assumption in all of these, including this dissertation, is that the network can be represented as a static graph, usually even a unit disk graph. Physical wireless links, in contrast, are prone to instabilities. Even if the nodes are not moving, the neighborhood of a node can change completely and, more importantly, usually unpredictably, over a short period of time. This can be due to the drastic effects of interference, scattering, or dampening of signals. If sensor networks are to be deployed in uncontrolled environments, then these effects simply cannot be ignored. We will further discuss these phenomena in Section 4.2 on the basis of the measurements we have taken.

Numerous of the other works also have particular hardware requirements which assume fairly accurate measurements on the part of the nodes. Examples include measuring the time of flight (or time difference of arrival (TDoA) as in [65, 117]), the angle of arrival (AoA) [107], the received signal strength (RSS) [16]. Measuring the time of flight in reasonably-sized ad hoc networks presupposes very accurate hardware which can detect differences in the nano second range: If we assume approximately the speed of light, the time it takes for a signal to cross the distance of a few meters amounts to some tens of nano seconds. In this paper, we have also opted to use the signal strength measured in terms of packet loss at different powers as a first indication of the distance between two nodes. However, instead of requiring a particular hardware component dedicated to the precise power measurement of incoming signals, we have even less stringent hardware requirements. This is described in more detail in Section 4.1.

Another line of research has been the development of hardware suited to the specific purpose of sensor localization. Papers in this area include Cricket [117], RADAR [16], or also [124]. Some of these work only indoors (Cricket, RADAR), some only outdoors (GPS). Cricket has a separate ultrasonic component and RADAR was tested on a larger scale with more available computing power (laptops). With current or foreseeable technology, a node cannot support fairly sophisticated positioning hardware *in addition to* the sensor and actuators that carry out the intended purpose, all at the smallest scale. Thus, we want to investigate the potential of very limited hardware for positioning in sensor networks. We also do not impose any indoor/outdoor restrictions.

Our contributions are on one hand results on the stability, symmetry, and distance relationship of the wireless radio links, and on the other hand we have observed that the gap in the measurements between two nodes in a lab and the nodes in a network is significant enough to render any localization attempts useless at this point. Put in another way, the distance-to-power correlation is strongly and unpredictably environment dependent.

4.1 Hardware Platform

The hardware we have used for our experiments is the ESB/2 platform from the scatterweb project, now its own company [130]. The nodes are built from standard components, consisting of a chip with a 32kHz CPU, 2kB of RAM, and a low power consumption radio transceiver, along with numerous sensors and actuators such as infrared, temperature, vibration, microphone, beep, and LED.

In the summer 2004 version available to us, the nodes can adjust their transmission power x from the application (for $1 \leq x \leq 100$ percent), but the transceiver is not able to directly measure the received power, only whether

the signal is above a threshold. The way that the power is adjusted at the sender is via a potentiometer which controls the current to the transceiver. It has been brought to our attention that it is now possible to read out the received signal strength on the ESBs directly and this modification is part of future work in this area.

What we do instead is a "software version" of RSS by measuring the packet loss while varying the transmission power at the sender. In order to determine an approximation to the received signal strength, an anchor node writes its sending signal level into a packet, and the receiving node reads out this value, takes the minimum over all received packets, and can thus determine the lowest signal level at which it can still "hear" the anchor. While this way of measuring the transmission power is certainly not the most precise way, it fulfills the natural assumption that greater perceived received signal strength means that the sender needed to use more power to reach the receiving node, thus the receiver is farther away. The exact correlation needs to be determined, but the important point we want to verify is that the same input level on the sender should reach the same distance given similar conditions.

A critical issue with these nodes is the susceptibility of the radio signal strength to various outside influences. Two nodes might be as close as a couple of centimeters, but placed near a wall or close to some underground cable, or as far as a few meters, and both times the best received signal strength will be the same. Here, we build on the preliminary measurements of the transmission range as a function of the signal strength in various settings from the project website [130].

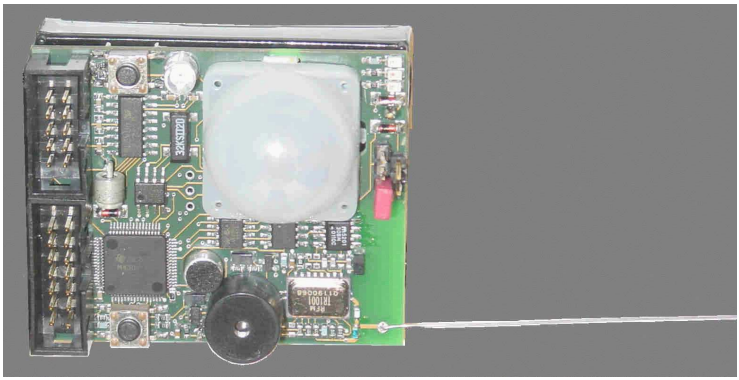

Figure 4.1: An embedded sensor board (ESB/2) from [130].

4.2 Experiments and Results

We will now discuss our measurements, their results, and the motivations that led from one experiment to another. In the following we use the terms sensor node and node interchangeably.

4.2.1 In the Lab

The goal is to implement popular positioning heuristics on real sensor nodes. Towards this end we first need to obtain some data on the correlation between the power level received and the distance of the nodes without obstacles.

Our experimental setup is the following: In the corridor of our lab, an anchor node transmits 100 packets at each power level and a receiving node placed at a specified distance measures the number of packets it receives. This experiment is repeated for inter-node distances ranging from 1cm to 120cm, as in a first step we want to explore the accuracy of the distance-to-power relation on a small scale. The minimum power level at which a packet is received at a given distance is plotted in Figure 4.2. While the data points do not lie on the theoretically assumed parabola, they are almost monotone with slight deviations of at most three levels and the curve exhibits a certain regularity.

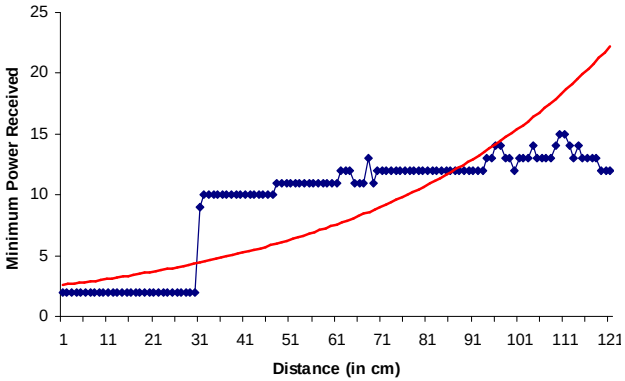

Figure 4.2: The minimum power level which was received at the given distance.

Most applications for wireless networks will, however, not be satisfied with a single packet arriving with some low probability. Therefore we furthermore

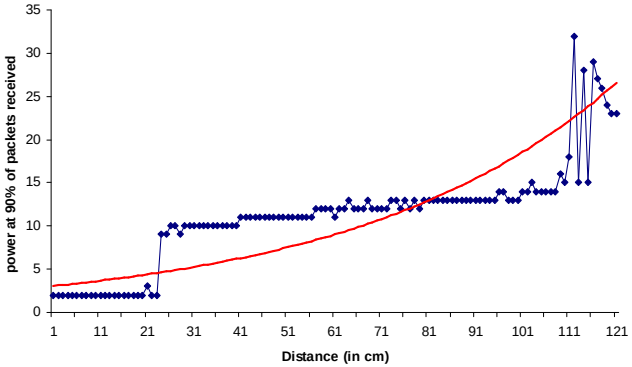

Figure 4.3: The lowest power level at which at least 90% of the packets were received.

test the link quality if we require that at least x percent of the packets arrive at the listening node. The results for $x = 90$ are shown in Figure 4.3. The graph looks similar for all other values of x (down to 50).

The deviation of the data set to the best-fit curve in both experiments is not negligible (about 10 units in the latter case), as can be seen in Figure 4.2 and 4.3. However, the data set is still well-behaved in the sense that a curve is discernible and this curve is almost monotone.

The conclusion that can be drawn from these experiments is that in a controlled environment with a clear line of sight, the distance-to-power function with a specified stability can be approximated to a certain extend by a monotonely increasing function, supporting theoretical assumptions. In such a setting, a possible localization scheme could measure the packet loss for different power levels from anchor nodes (which know their position) and then use the inverse of the distance-to-level point map in Figure 4.3 to obtain a distance estimate.

4.2.2 In a Room

Any localization algorithm in the plane needs (approximate) distance measurements from at least three non-collinear anchors. Therefore, the next step in our experiments is to expose a single node to several anchors and test the obtained measurements for their usability.

The experimental setup is similar to the one before. We place four anchor nodes on the corner of a rectangle in a room. A test node is then placed within that rectangle. An anchor sends out a packet at each power level from 1 up

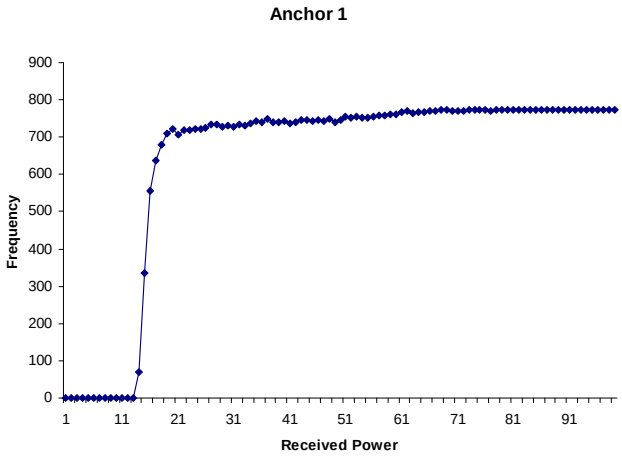

Figure 4.4: The number of times a given power level was received. Anchor 1 was 3.02 meters from the test node.

to 100, then the next anchor does the same, and so on, in a round robin fashion. Each time, the test node reports which packets it receives. Figure 4.4 shows how often a given power level is received by the test node from Anchor 1 over the course of the experiment. The results for the other anchors are similar and thus omitted.

This already goes to show that the link between the test node and the anchors is reasonably stable over time. As we already see from the earlier experiments in the lab, the *minimum* power level received, perhaps averaged over time, gives a good indication for the distance to the sending node. This is advantageous since it saves memory compared to storing all received power levels. To further strengthen this hypothesis, we also examine the size of the "holes" in the received power levels for each iteration. In other words, how big is the gap between one received power level and the next and how often does it occur. For example, if the first heard packet has level 15 and the next one heard level 18, then this results in a hole of size 3. Figure 4.5 shows that there are only small holes in most cases. Meaning that the minimum power level is a good estimate if the requirements are not too stringent.

Table 4.1 summarizes the results, comparing the average minimum received power level with the actual distance. The most notable effect is that with increasing distance, the minimum power does not steadily increase but instead fluctuates.

In a next step, we add obstacles to our room by placing various everyday

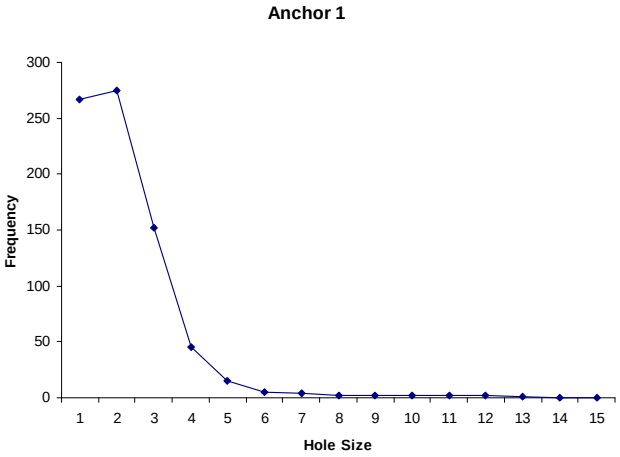

Figure 4.5: The size of the "holes" in the received power level progression and how often they occurred for Anchor 1.

anchor	distance	avg. min power
A_2	1.39	11
A_0	2.78	15
A_1	3.02	16
A_3	3.65	14

Table 4.1: Average minimum received power level from the different anchors and their true distances in meters.

objects in the area of the rectangle. The result is that the general behavior of the link quality does not appear to be affected, seen in Figure 4.6. The curve has the same "shape" as before (less data points being the reason for the height difference). Astonishingly, the peak from the experiment with obstacles is shifted to the left compared to the non-obstacles experiment. This result only hardens the conclusion of the unpredictability of real-world sensor node behavior.

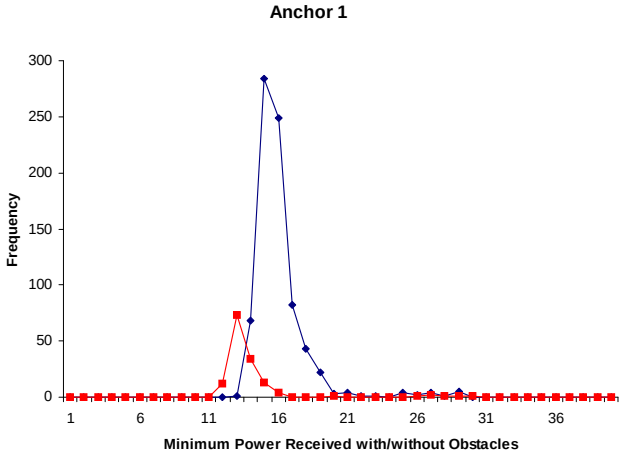

Figure 4.6: Number of times that a given power level was the minimum frequency received. The higher peak (blue) is from the setup without obstacles, the other one (red) with obstacles. The reason for the lower peak is that the experiment was run for less time due to external constraints.

On the positive side, we can say that in an environment with several stable anchors, each sending out packets one at a time to avoid collisions, the links appear to be stable over time and exhibit a sharp peak.

4.2.3 Network

The results of the previous sections suggest that distance measurements based on radio power levels can be used if the accuracy constraints are not too tight. This lead us to set up a positioning experiment with four anchors at the corners of a rectangle (a table) and several nodes on the inside of the rectangle. We use a spring-based algorithm in which each node iteratively computes its new position as a function of its neighbors' positions. This approach stems from the graph drawing context and was first adapted to

positioning in works such as [122] (see also Sections 2.3 and 2.4.2). Since the heuristics proposed in [122] are far too power and resource consuming, we implement a simplified version to suit our purposes. For the power-to-distance conversion, the data gathered in the experiments above is used. Surprisingly, between most of the computed positions and the corresponding actual positions there is seemingly no correlation. The errors are in the order of the magnitude of the sensor field. A closer examination reveals that already the powers received do not correlate with the distances at all in the sense that a node close by needed significantly more power to communicate than some nodes far apart, even without any obstacles in the room. This finding is closer examined in the next experimental setup.

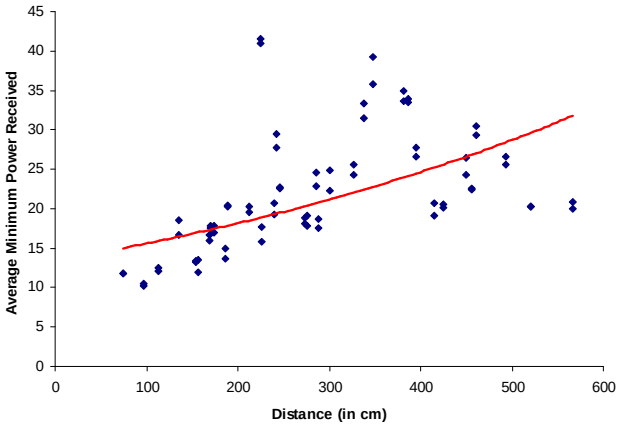

Figure 4.7: Measured power levels at various distances. The x-axis is distance (in cm), the y-axis is average minimum power level. A point is the measurement of a node to some other node.

We perform the following experiment, the result of which can be seen in Figure 4.7. We take nine nodes placed arbitrarily in a large room. Iteratively each node sends out a series of packets, starting from level 1 to 100. In each iteration all non-sending nodes record the minimum power level received from the sending node in that iteration. As can be seen, the graph looks very different from the one in the first phase of the experiment, Figure 4.2. Whereas in the first phase of the experiments a curve connecting the data points is discernible, the data points measured in the current experimental setup are much more scattered. Observe also that the scale of the y-axis in both figures is different and that the deviation of the data set to the best-fit parabola in this experiments is about 30 units.

The conclusion to be drawn here is that in a two-node setting with a constant environment, the power expectedly increases approximately quadratically with distance. In a network, however, there are several nodes, unpredictable environmental conditions such as people walking around, and different positions of walls, cables, other computers and the like. So even without the effect of node transmission interference, which we excluded in this setup, the power-to-distance correlation becomes utterly useless. The data points are scattered across the graph. While *two nodes* in the network can have a quadratically increasing (theoretical) distance-to-power function, this function between *different pairs* of nodes is not necessarily related in a predictable manner.

On the positive side, this experiment allows us to conclude on the symmetry of the links between two nodes. To that end, we compare the minimum power levels between two nodes in the network. In Figure 4.8 the number of occurrences a power level was the lowest one received for an arbitrary pair of nodes is plotted. The peaks are sharp and coincide.

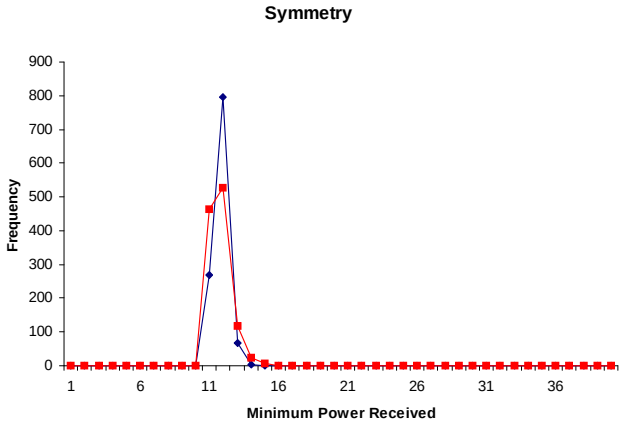

Figure 4.8: Number of times that a given power level was the minimum frequency received. The blue curve is from node 75 to node 65 and the red curve the other way around.

We furthermore compute the average minimum power level received for each node to all other nodes and take the difference between the node pairs. This value is rounded to the nearest integer and Figure 4.9 shows the number of times a difference occurs among the $\binom{9}{2} = 36$ node pairs.

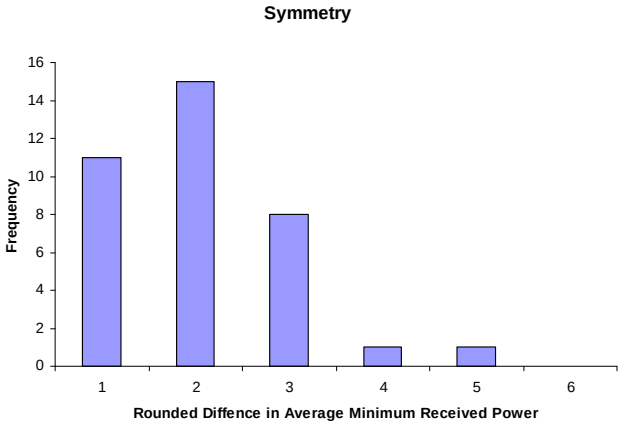

Figure 4.9: Rounded differences between average minimum received power.

4.3 Discussion and Future Work

In order to gauge the applicability of theoretical research in practical settings, we conducted an experimental study of the link quality and how it correlates with distance measurements in real-world sensor networks. If one expects the sensor network to be in place for an extended period, then we do not necessarily require an unvarying connection between two nodes, but that the link characteristics should be *steady over time*. The minimum power at which a node receives a packet from a sender node was shown to be stable, sharp and symmetric over time in all experimental setups. Whereas this power level is a good indication for the distance between nodes if there are only two nodes in a static experimental setup, this correlation does not apply to all node pairs in a larger arbitrary network. Our experiments illustrate that in larger networks the power levels measured in lab conditions are not correlated with distances in the real world, even if the real world is the lab next door. Thus drastically new models are required for sensor networks if theoretical constructs are ever to be applied with a realistic chance.

Despite the pessimistic results, the findings of this chapter open up a number of interesting directions meriting further investigation. First of all, these tests should be performed on hardware from different vendors to see whether this is a general problem. Second, with the new version of the ESBs, we can perform actual received signal strength measurements which would refine the results observed in this paper. Third, it would be interesting to investigate whether there is a qualitative difference between short and long

range measurements.

The aim of this chapter was to highlight the chasm between theoretical models and current reality of sensor networks. While the gap, as we have presented it, is glaring, we believe that the truth lies somewhere in the middle. Wireless node hardware will evolve and better models and algorithms might capture their behavior more closely. From the theoretical side, we should pursue two directions simultaneously: completing the algorithmic picture based on current knowledge while at the same time striving for more accurate models. The ultimate goal is for theory and practice to complement, not contradict each other.

Chapter 5

UDG Embedding

From the practical side, we saw that much work remains to be done in order to achieve light-weight, fine-grained wireless ad hoc network localization. From the theoretical side, we know of the $O(\sqrt{n})$ trivial approximation algorithm for the virtual coordinates problem when all n nodes are arranged arbitrarily on a grid. After some precursory investigations of the local properties of unit disk graphs in Chapter 3, we finally turn to the full-fledged embedding problem in this chapter. To the best of our knowledge, we provide the first algorithm with an approximation guarantee improving over the $\sqrt{n}$ bound. Two key ideas are necessary to achieve this. The first is that we work primarily with a maximal independent set of nodes instead of the entire graph at once. The second is to look at distances as the initial quantity of interest instead of going straight for a point-wise embedding.

5.1 Preliminaries

We will briefly gather the necessary notation and concepts before proceeding to the description of the algorithm.

5.1.1 Definitions

An *independent set* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that for all $v_i, v_j \in S$, $\{v_i, v_j\} \notin E$. An independent set is *maximal* if no additional vertex can be added to S such that the resulting set is still an independent set.

A *metric space* (or simply a *metric*) is a pair (S, ρ), where S is a set of points and $\rho : S \times S \to [0, \infty)$ is a *distance function* satisfying the properties

1. identity of indiscernibles: $\rho(p, q) = 0 \Leftrightarrow p = q$
2. symmetry: $\rho(p, q) = \rho(q, p)$

3. triangle inequality: $\rho(p,r) + \rho(r,q) \geq \rho(p,q)$

for all $p, q, r \in S$. A finite metric space can be represented as a weighted complete graph on $|S|$ vertices where the edge lengths between pairs of vertices are equal to the distance between the respective points in the metric space.

An embedding of a finite metric space (S, ρ) into an L-dimensional Euclidean space $(\mathbb{R}^L, \sigma)$ is a mapping $f : S \to \mathbb{R}^L$. In this paper we only consider *contracting* mappings, that is, for all $x, y \in S$, $\sigma(f(x), f(y)) \leq \rho(x, y)$. We say that the volume $Evol(S)$ of a set S of k vertices in $\mathbb{R}^L$ is the volume of the $(k-1)$-dimensional simplex spanned by S. Then the volume $Vol(S)$ of a finite metric space with k points is defined to be the maximum $Evol(f(S))$ over all contracting mappings f from S to $\mathbb{R}^L$, $L \geq k$. The *distortion* of an embedding f for the set S is the ratio

$$\frac{Vol(S)}{Evol(f(S))}.$$

In other words, we are not considering the pair-wise distortion of points anymore, but the distortion of an arbitrary set of points. This leads us to the central definition of this chapter. The idea of volume-respecting embeddings was first introduced by Feige [49] to solve the bandwidth problem which concerns itself with embedding a graph onto the integer line.

Definition 5.1 (Volume Respecting). *An embedding f of a metric V is* (k, D)-volume respecting *if*

$$\left(\frac{Vol(S)}{Evol(f(S))}\right)^{1/(k-1)} \leq D \tag{5.1}$$

for every subset $S \subseteq V$ of at most k vertices.

5.1.2 Terminology

In analogy to the term edge, we call a pair of vertices $\{v_i, v_j\}$ a *non-edge* if $\{v_i, v_j\} \notin E$. When referring to *high probability* we mean with probability at least $1 - O(1/n)$.

5.2 Algorithm Overview

Our virtual coordinates algorithm ViCe (Virtual Coordinates Embedder) is inspired by Vempala's work on VLSI layout [143] where he first proposed the combination of spreading constraints and volume-respecting embeddings in order to find a good grid layout. Before presenting our algorithm in full detail, we want to describe it on a higher level as well as give some intuitive ideas. Working directly on the coordinates when embedding a unit disk graph

has proven ineffective in all previous attempts at the problem. Therefore, we work on distances, that is, a metric of the graph first. Producing a Euclidean metric outright is difficult, and a large body of research on general metric embeddings bears witness to the fact that embedding metrics in low dimensions is also hard. One option is to embed into a higher space and then project down. In order to obtain a good quality of the final embedding, low edge distortion in the higher space embedding is not enough when projecting back down to two-dimensional space. This is the reason we use Feige's [49] generalized embedding where sets (as opposed to pairs) of points have low distortion.

ViCE consists of four main conceptual states. In the subsequent sections we explain the ideas behind each of the main stages separately. The important concepts are roughly the following: The main part of the computation operates on a maximal independent set M of the input UDG G. In the first stage we solve a set of linear constraints which gives indications on the distances between any two vertices in M. Those distances are used thereafter to embed the vertices in an L-dimensional space in such a way that sets of vertices have large volume. Given this high-dimensional embedding we project the points to a randomly chosen 2-dimensional plane. Finally, the points in M are placed on grid points, while the others are placed around them.

5.2.1 Linear Constraints

The main goal of the first stage of our algorithm is to compute a metric on the input graph satisfying certain additional constraints. The key difficulty is that the problem of describing the UDG conditions is inherently non-convex, in particular the constraints on the non-edges. As such, we need to replace these problematic constraints with something more manageable. We will make use of *spreading constraints* (as in Even et al. [48]), which state that

$$\sum_{v \in S} \rho(u, v) \geq c \cdot |S|^{3/2} \quad \forall S \subset V, \forall u \in V \tag{5.2}$$

for some constant c. In our case, we cannot apply this concept to all the nodes of G, but only to an independent set of nodes.

The motivation for using spreading constraints is to model the property that in any region of diameter R there are at most $O(R^2)$ points. That is, the points are forced to spread out, instead of clustering in a single spot. Now, in relation to unit disk graphs, we obviously do not want *all* points to be spread out, but rather want to model the fact that non-edges are "far apart" while edges are allowed to be "close." This becomes intuitively clear when we think of a clique of size n. Surely, a clique is a unit disk graph, but one that cannot fulfill the spreading constraints. So instead of formulating the spreading constraints on all sets of the original vertices, we observe that we want the sets of non-edges, i.e., independent sets, to be

sufficiently far apart. Serendipitously, any independent set IS of a unit disk graph G satisfies Eq. (5.2). To see this, observe that for any given realization of G, the number of independent nodes in a region of radius R is at most κR^2 (κ being a constant) since all pair-wise distances are greater than 1.

We first construct a new graph $G_M = (V_M, E_M)$ out of $G = (V, E)$ by taking the nodes of a maximal independent set (MIS) of G to be the nodes of G_M and connecting two nodes in G_M if they are connected by a path of length at most 3 in G. If one is only interested in an approximate embedding, then the MIS placement encodes all the essential properties of a unit disk graph G with n nodes. Now we are ready to formulate a set of convex constraints[1] (LP) to describe G_M. For the sake of readability, we set $x_{uv} = \rho(u, v)$, and let $x_{uu} = 0$. Also note that $m = |V_M| \in O(n)$, where $n = |V|$.

(LP)

$$x_{uv} \leq 3 \qquad \forall \{u, v\} \in E_M \tag{5.3}$$

$$x_{uv} \leq \sqrt{n} \qquad \forall u, v \in V_M \tag{5.4}$$

$$x_{uv} \geq 1 \qquad \forall u \neq v \in V_M \tag{5.5}$$

$$x_{uv} + x_{uk} \geq x_{vk} \qquad \forall u, v, k \in V_M \tag{5.6}$$

$$\sum_{v \in S} x_{uv} \geq \kappa |S|^{3/2} \qquad \forall S \subset V_M, \forall u \in V_M \tag{5.7}$$

Constraint (5.3) is a consequence of the MIS graph construction out of G. Constraint (5.4) stems from G being embeddable in a square of side length $\sqrt{n}$. Constraints (5.5) and (5.6) ensure that we have a metric which we need in order to embed the nodes into Euclidean space. Constraint (5.7) are the spreading constraints from the discussion above, with κ a constant. From the definition of unit disk graphs and the discussion of the spreading constraints above, it immediately follows that if G is a UDG and G_M its MIS graph, then there exists a feasible solution to (LP). We show in Section 5.4.5 that it can be found in polynomial time.

Observe that we are not actually computing any coordinates of the points yet, only their pairwise distances. This is perhaps the key conceptual difference between our approach and that of previous virtual coordinates algorithms, such as [42, 136, 122], where the point coordinates, which these algorithms attempt to approximate, are included in the initial equations. Unfortunately, we cannot specify linear conditions which force the metric ρ to be a true Euclidean metric, which is why the spreading constraints serve as a first approximation.

Now that we have computed a metric with the desired properties, we want to embed it into a geometric space. Preferably, we would like to embed it in the 2-dimensional plane, but in general this cannot be done directly without large edge distortion. Instead, we embed the vertices in a high-dimensional

[1] basically, a linear program without an optimization function

space where we can guarantee not only that the distortion is small but also that the embedding is *volume respecting*.

5.2.2 Volume-Respecting Embedding

Feige [49] introduced a powerful strengthening of the notion of the edge distortion of an embedding, concerning embeddings into Euclidean spaces. Whereas the usual distortion of an embedding is determined by looking at pairs of points, that is, distances, Feige's volume-respecting embedding takes into account all k-tuples for some $k \geq 2$, i.e., volumes of sets of points.

A volume-respecting embedding has many useful properties when projecting the points from the high-dimensional space to a random lower-dimensional one. Intuitively, large volumes have large projections and hence the points spread fairly well when projecting to a random lower-dimensional subspace. This leads us to the third stage of our algorithm. Once a volume-respecting embedding in $\mathbb{R}^L$ is computed, the points of the embedding are projected to a randomly chosen 2-dimensional plane.

5.2.3 Random Projection

Random projection is the technique of projecting a set of points from a high-dimensional space to a low-dimensional subspace. Lately, this technique has enjoyed great popularity in various research areas. For our application, two observations about random projections are crucial. Firstly, the length of a vector when projected from $\mathbb{R}^L$ to a random line scales by roughly $1/\sqrt{L}$ and is concentrated around this expectation. Secondly, the probability that a set of k points is projected to a small interval is inversely proportional to the volume of the points. Hence, together with the fact that our embedding was volume respecting, we get that the projected points spread quite well in the 2-dimensional plane. In fact, we prove in Section 5.4.4 that if we partition the plane into a grid with cell-width $1/\sqrt{L}$ then at most $O\big(\log^5 n \log\log n\big)$ points lie in a cell with high probability. This leaves us to show what is to be done with the points in one cell in the final operation on G_M.

5.2.4 Final Embedding

In order to guarantee that the smallest non-edge is not too short we need the points within a cell to be evenly spread out and not to cluster at a single point. A simple way to do this is to space the points in a cell onto yet another, smaller grid. The width of a cell in the refined grid is $1/\sqrt{LM}$ along each dimension, M being the maximum number of points in any cell. After rescaling the grid to have cell-width 1, the above discussion shows that this introduces at most a poly-logarithmic factor with high probability, which is needed to bound the maximum edge length.

We now have to embed all the vertices in G which are not in the MIS V_M to bound the non-edge lengths. First, we can assign each node u not in V_M to exactly one vertex in V_M. Such an assignment always exists since each vertex not in V_M has a neighbor in V_M, otherwise it would contradict the maximality of the independent set. Then we consider each $v \in V_M$ along with its reduced neighborhood $N'(v)$ consisting of all of its assigned vertices. The idea now is to compute a clique partition of $N'(v)$ so that all nodes of a clique can be put on the same point. These clique points are put onto a grid inscribed in a circle of diameter $\frac{2}{3}$ centered at v. To show that this results in a minimum distance of $\Theta\big(1/\sqrt{\log n}\big)$ between grid points, we observe that a crucial UDG property is that the neighborhood of a node can be partitioned into at most a constant number of cliques. Using an algorithm of [120], we can successively compute a maximum clique in $N'(v)$, resulting in a $\Theta(\log n)$ approximation to a minimum clique partition of $N'(v)$. Thus, there are $\Theta(\log n)$ points representing non-MIS nodes around each MIS node, bounding the minimum non-edge length.

5.3 Algorithm Details

Algorithm 5.1 gives the pseudo-code of ViCE for computing the virtual coordinates of a given unit disk graph. Algorithms 5.2-5.6 succinctly describe each of the above stages. The output of each phase serves as the input to the next one.

In the following, let us denote the position of vertex u in the volume-respecting embedding by v_u^L, the position after the random projection by r_u and the final position by p_u. The dimension of the embedding in the second step is given by Feige in [49] as $L = \Theta\big(k^2 \log^2 n \log k\big) = \Theta\big(\log^4 n \log\log n\big)$ with $k = \log n$. Recall that n denotes the original number of vertices and m the number of vertices in the MIS graph G_M.

Input: $G = (V, E)$ a unit disk graph
Output: positions $p_u \in \mathbb{R}^2$ for all $u \in V$
$V_M \leftarrow MIS(G)$
$E_M \leftarrow \{\{u, v\} \mid d_G(u, v) \leq 3 \text{ for } u, v \in V_M\}$
$G_M \leftarrow (V_M, E_M)$
$\{x_{uv}\} \leftarrow$ Solve (LP) on G_M
$\{v_u^L\} \leftarrow$ Volume-Respecting Embedding($\{x_{uv}\}$)
$\{r_u\} \leftarrow$ Random Projection($\{v_u^L\}$)
$\{p_u\} \leftarrow$ Round to Grid($\{r_u\}$)
$\{p_u\} \leftarrow$ Place Non-MIS(G, V_M, $\{p_u\}$)

Algorithm 5.1: ViCE

Input: $G_M = (V_M, E_M)$
Output: distance matrix $X = (x_{uv})$
1: solve (LP) and return set of distances x_{uv} between each pair of vertices u, v

Algorithm 5.2: Solve (LP)

Input: $m \times m$ distance matrix $X = (x_{uv})$
Output: positions $v_u^L \in \mathbb{R}^L$ for all $u \in V_M$
1: find a $(\log m, \beta \log m \sqrt{\log \log m})$-volume respecting Euclidean embedding in $\mathbb{R}^L$ using the "random subsets embedding" in [49]

Algorithm 5.3: Volume-Respecting Embedding

Input: positions $v_u^L \in \mathbb{R}^L$ for all $u \in V_M$
Output: positions $r_u \in \mathbb{R}^2$ for all $u \in V_M$
1: independently choose two random vectors $l_1, l_2 \in \mathbb{R}^L$ of unit length (lines passing through the origin)
2: for all $u \in V_M$, project v_u^L to each of the lines, that is, $r_u \leftarrow (v_u^L \cdot l_1, v_u^L \cdot l_2)$

Algorithm 5.4: Random Projection

Input: positions $r_u \in \mathbb{R}^2$ for all $u \in V_M$
Output: positions $p_u \in \mathbb{R}^2$ for all $u \in V_M$
1: enclose the projected points in a grid $C_{\sqrt{L}}$ of cell-width $1/\sqrt{L}$
2: let C be a cell in $C_{\sqrt{L}}$, then $M \leftarrow \max_C \{u \mid r_u \in C\}$
3: refine $C_{\sqrt{L}}$ by subdividing each cell into M subcells, denote the refined grid by $C_{\sqrt{LM}}$
4: **for** each cell C in $C_{\sqrt{L}}$ **do**
5: assign all points in C arbitrarily to grid-points in $C_{\sqrt{LM}}$ which lie in C
6: **end for**
7: scale $C_{\sqrt{LM}}$ by $\sqrt{LM}$ along each dimension, such that the distance between any two grid points is 1

Algorithm 5.5: Round to Grid

Input: $G = (V, E)$, V_M, positions $p_u \in \mathbb{R}^2$ for all $u \in V_M$
Output: positions $p_u \in \mathbb{R}^2$ for all $u \in V \setminus V_M$

$V' \leftarrow V \setminus V_M$
for all $v \in V_M$ and while $V' \neq \emptyset$ **do**
  $N'(v) \leftarrow N(v) \cap V'$
  $V' \leftarrow V' \setminus N'(v)$
  $x \leftarrow$ number of cliques in $N'(v)$ by successively removing maximum clique (using [120])
  create grid of cell width $1/\sqrt{x}$ inside circle of diameter $2/3$ centered at p_v
  number grid points $g_1, g_2, \ldots, g_x$
  $i \leftarrow 1$
  while $N'(v) \neq \emptyset$ **do**
    $C \leftarrow$ maximum clique in $N'(v)$ using [120]
    for all $u \in C$ **do**
      $p_u \leftarrow g_i$
    end for
    $i \leftarrow i + 1$
    $N'(v) \leftarrow N'(v) \setminus C$
  end while
end for

Algorithm 5.6: Place Non-MIS

5.4 Analysis

This entire section is devoted to proving the following main theorem.

Theorem 5.1. *The quality of the embedding p computed by the* ViCE *algorithm given in Alg. 5.1 is, with high probability,*

$$Q(p(G)) = O\Big(\log^{3.5} n\sqrt{\log\log n}\Big) \tag{5.8}$$

for input unit disk graph $G = (V, E)$ with $|V| = n$.

5.4.1 Volume-Respecting Embeddings

Once we have a set of inter-point distances, we want to ensure that the high-dimensional embedding does not distort the encoded UDG properties. The volume-respecting property of Feige's embedding ensures that independent vertices remain spread out.

Theorem 5.2 ([49] Thm. 5). *For any k and any connected graph G on n vertices, an embedding which is $(k, \beta\sqrt{\log n}\sqrt{k\log k + \log n})$-volume respecting with high probability can be found in polynomial time for some large enough constant β.*

Here we use $k = \log m$ giving a distortion of $O\big(\log m\sqrt{\log\log m}\big)$ on the volumes of $\log n$-sized sets.

Unfortunately, computing the volume $Vol(S)$ of a set S exactly is far too tedious in general. Yet, Feige showed that it can be approximated quite well by making use of the notion of the *tree volume* $Tvol(S)$ of S, which is the product of the edge lengths in a minimum spanning tree of S.

Theorem 5.3 ([49] Thm. 3). *For $S \subset V$, $|S| = k$,*

$$Vol(S) \leq \frac{Tvol(S)}{(k-1)!} \leq Vol(S)2^{(k-2)/2}.$$

5.4.2 Tree Volume

Since we want to ensure that the independent sets are neatly spread out, we need to derive bounds on the tree volume of those sets in order to use Feige's volume-respecting embedding. In particular, we want to show the following.

Lemma 5.4. *Let $G_M = (V_M, E_M)$ be the MIS graph of G with $m = |V_M|$, then*

$$\sum_{S\subset V_M,\,|S|=k} \frac{1}{Tvol(S)^2} \leq \frac{2k!}{(2\kappa)^{2k-2}} \cdot m(\log m)^{k-1}.$$

with κ from Eq. (5.7) of (LP).

Proof. To obtain an enumeration of all subsets of size k, we can first choose a $u_1 \in V_M$, then a $u_2 \in V_M \setminus \{u_1\}$, then a $u_3 \in V_M \setminus \{u_1, u_2\}$, and so on up to $u_k \in V_M \setminus \{u_1, u_2, \ldots, u_{k-1}\}$. Then each set is counted $k!$ many times. For readability, denote by I_j the set $\{u_1, \ldots, u_j\}$ for $1 \leq j < k$. From this we get that

$$\sum_{S \subset V_M,\, |S|=k} \frac{1}{Tvol(S)^2} = \frac{1}{k!} \sum_{u_1 \in V_M} \sum_{u_2 \in V_M \setminus I_1} \cdots \sum_{u_k \in V \setminus I_{k-1}} \frac{1}{Tvol(\{u_1, u_2, \ldots, u_k\})^2} \quad . \tag{5.9}$$

We will need to write $1/Tvol(S)^2$ in terms of the distances between the u_i. Feige has shown ([49] Lem. 17) that for any finite metric space $S = \{u_1, \ldots, u_k\}$,

$$\frac{2^{k-1}}{Tvol(S)} \leq \sum_{\pi \in S_k} \frac{1}{x_{u_{\pi(1)} u_{\pi(2)}} x_{u_{\pi(2)} u_{\pi(3)}} \cdots x_{u_{\pi(k-1)} u_{\pi(k)}}} \tag{5.10}$$

where the summation is taken over all permutations $\pi \in S_k$. Note that in our case the set S, by virtue of being a solution to (LP), is also a (finite) metric space. Here, we need the square of the volume since we are projecting to two dimensions instead of just one. Thus, squaring Eq. (5.10), and writing $x(\pi)$ for $x_{u_{\pi(1)} u_{\pi(2)}} \cdots x_{u_{\pi(k-1)} u_{\pi(k)}}$, we get

$$\begin{aligned} \frac{2^{2(k-1)}}{Tvol(S)^2} &\leq \sum_{\pi \pi'} \frac{1}{x(\pi) \cdot x(\pi')} \\ &= \sum_{\pi} \frac{1}{x(\pi)^2} + \sum_{\pi \neq \pi'} \frac{1}{x(\pi) \cdot x(\pi')} \\ &\leq 2k! \sum_{\pi} \frac{1}{x(\pi)^2}. \end{aligned} \tag{5.11}$$

For the last inequality, consider $\sum_{j \neq i} a_i a_j = 2 \sum_{i=1}^{l} \sum_{j=i+1}^{l} a_i a_j$ with positive a_i's in ascending order. With $a_i \leq a_j$ for $j > i$, then also $a_i a_j \leq a_j^2$ and thus

$$\sum_{j \neq i} a_i a_j \leq 2 \sum_{i=1}^{l} (i-1) a_i^2 \leq 2(l-1) \sum_{i=1}^{l} a_i^2$$

with $l = k!$ in our case.

Plugging (5.11) back into (5.9), we get that

$$\sum_{S\subset V_M,\,|S|=k} \frac{1}{Tvol(S)^2} \leq \frac{1}{2^{2k-3}} \sum_{u_1} \cdots \sum_{u_k} \sum_{\pi} \Big(\frac{1}{x_{u_{\pi(1)}u_{\pi(2)}} \cdots x_{u_{\pi(k-1)}u_{\pi(k)}}} \Big)^2$$
$$= \frac{k!}{2^{2k-3}} \sum_{u_1} \sum_{u_2} \frac{1}{x_{u_1u_2}^2} \sum_{u_3} \frac{1}{x_{u_2u_3}^2} \cdots \sum_{u_k} \frac{1}{x_{u_{k-1}u_k}^2} \tag{5.12}$$

after reordering the u_i since a permutation on the indices is equivalent to a different choice of u_i. In other words, we now only need to look at sums of the form given by the following lemma. It is a crucial property which stems from the spreading constraints.

Lemma 5.5. *Let (x_{uv}) be a feasible solution vector of (LP) with κ from Eq. (5.7). For any vertex u in V_M and any $S \subseteq V_M \setminus \{u\}$, $|V_M| = m$, it holds that*

$$\sum_{u_i \in S} \frac{1}{x_{uu_i}^2} \leq \frac{\log m}{\kappa^2}.$$

Proof. Fix u and consider any $S = \{u_1, \ldots, u_l\}$ where the u_i are ordered in increasing distance from u, that is, $x_{uu_i} \leq x_{uu_{i+1}}$ for $1 \leq i < l$. Since (x_{uv}) is a solution to (LP), $\sum_{u_i \in S} x_{uu_i} \geq \kappa l^{3/2}$ from Eq. (5.7). Therefore, at least one of the u_i has distance at least the average value of the spreading constraint, thus $x_{uu_l} \geq \kappa l^{1/2}$ since we ordered S by distance from u. Now remove u_l from the set and take $S = \{u_1, \ldots, u_{l-1}\}$, again ordered as above. By the same argument, $x_{uu_{l-1}} \geq \kappa(l-1)^{1/2}$, and in general, $x_{uu_i} \geq \kappa i^{1/2}$. Then we get

$$\sum_{i=1}^{l} \frac{1}{x_{uu_i}^2} \leq \frac{1}{\kappa^2} \sum_{i=1}^{l} \frac{1}{i} = \frac{H(l)}{\kappa^2} \leq \frac{\log m}{\kappa^2}$$

with $l \leq m-1$. □

With this, the sums from u_2 to u_k in Eq. (5.12) can be bounded in total by $(\log m/\kappa^2)^{k-1}$, there are m choices for u_1, and Lemma 5.4 now follows. □

5.4.3 Random Projection

While the volume-respecting embedding helps in keeping non-edges far apart, we also want the edges to be close together. In this section, we prove that the edge length will be bounded after the random projection step.

For starters, we will need the well-known lemma given below [144].

Lemma 5.6. *Let $v \in \mathbb{R}^L$. For a random unit vector l,*

$$\mathbb{P}\left(|v \cdot l| \leq \frac{c}{\sqrt{L}}|v|\right) \geq 1 - e^{-c^2/4}.$$
$$\mathbb{P}\left(|v \cdot l| \leq \frac{1}{c\sqrt{L}}|v|\right) = O\left(\frac{1}{c}\right).$$

The next lemma gives a connection between random projection and the volume of a set. It will become important when we count the number of nodes that fall into any given cell of the final grid.

Lemma 5.7 ([143] Lem. 5). *Let S be a set of vectors $v_1, \ldots, v_k \in \mathbb{R}^L$. For $c > 0$, consider the event that the projection of S on a random unit vector l is of length at most c. The probability of this event is bounded by*

$$\mathbb{P}\left(\max_i(v_i \cdot l) - \min_j(v_j \cdot l) \leq c\right) = O\left(\frac{c^{k-1}L^{(k-1)/2}}{(k-1)!Evol(S)}\right).$$

Lemma 5.8. *The length $|r_u - r_v|$ of an edge after the projection step is at most $2|v_u^L - v_v^L|\sqrt{\log n}/\sqrt{L}$, with high probability.*

Proof. By Lemma 5.6 with probability at least $1 - e^{-c^2/4}$ a vector v has length at most $\frac{c}{\sqrt{L}}|v|$ after the projection. Choosing $c = 2\sqrt{\log n}$, the length of a projected vector is at most

$$2|v_u^L - v_v^L|\sqrt{\log n}/\sqrt{L}$$

with probability at least $1 - 1/n$. □

5.4.4 Putting Things Together

Lemma 5.9. *The number of vertices that fall into any cell of the outer grid $C_{\sqrt{L}}$ is in $O\big(\log^5 n \log\log n\big)$ with high probability.*

Proof. Let N_C be the number of sets S of size k that fall into an arbitrary grid cell C of width $1/\sqrt{L}$. Let X_S^i be the indicator random variable which is 1 if all the vectors in S fall in C along the randomly chosen line l_i. Following the reasoning in [143], we can express the expected value of N_C in terms of

X_S^1 as follows:

$$\begin{aligned}
\mathbb{E}[N_C] &= \sum_{|S|=k} \mathbb{E}[X_S^1 X_S^2] = \sum_{|S|=k} \mathbb{E}[X_S^1]\mathbb{E}[X_S^2] = \sum_{|S|=k} \mathbb{E}[X_S^1]^2 \\
&= \sum_{|S|=k} \mathbb{P}(X_S^1 = 1)^2 \\
&\overset{\text{(Lem. 5.7)}}{\leq} \sum_{|S|=k} \left(\frac{\alpha (c\sqrt{L})^{k-1}}{(k-1)! Evol(S)} \right)^2 \\
&\overset{(c = 1/\sqrt{L})}{=} \sum_{|S|=k} \left(\frac{\alpha}{(k-1)! Evol(S)} \right)^2 \\
&\overset{\text{(Thm. 5.2)}}{\leq} \sum_{|S|=k} \left(\frac{\alpha \big(\beta \sqrt{\log m}\sqrt{k \log k + \log m}\big)^{k-1}}{(k-1)! Vol(S)} \right)^2 \\
&\overset{\text{(Thm. 5.3)}}{\leq} \sum_{|S|=k} \left(\frac{\alpha 2^{(k-2)/2} (k-1)! \big(\beta \sqrt{\log m}\sqrt{k \log k + \log m}\big)^{k-1}}{(k-1)! Tvol(S)} \right)^2 \\
&= \alpha^2 2^{k-2} \big(\beta^2 \log m (k \log k + \log m)\big)^{k-1} \cdot \sum_{|S|=k} \frac{1}{Tvol(S)^2} \\
&\overset{\text{(Lem. 5.4)}}{\leq} \alpha^2 \frac{2^{k-2}}{2^{2k-3}} \cdot k! \cdot m \cdot \big(\beta^2 \frac{1}{\kappa^2} \log^2 m (k \log k + \log m)\big)^{k-1} \\
&= \alpha^2 k! m \Big(\frac{\beta^2}{2\kappa^2} \log^2 m (k \log k + \log m)\Big)^{k-1}
\end{aligned} \tag{5.13}$$

where α is the constant hidden in the big-O notation of Lemma 5.7, and β likewise from Theorem 5.2. To bound the number of cells of $C_{\sqrt{L}}$ right after the projection, note that the longest edge has length at most $6\sqrt{\log n}/\sqrt{L}$ by Lemma 5.8 and Constraint (5.3) from (LP) and the fact that the volume-respecting embedding is non-expansive. Therefore, the longest path has length $6m\sqrt{\log n}/\sqrt{L}$, which requires $6m\sqrt{\log n}$ cells of side length $c = 1/\sqrt{L}$ to cover, with high probability.

Using Markov's inequality, we have

$$\mathbb{P}\big(N_C \geq 6mn\sqrt{\log n} \cdot \alpha' k! m (\gamma \log^2 m (k \log k + \log m))^{k-1}\big) \leq 1/(6mn\sqrt{\log n}) \tag{5.14}$$

readjusting the constants to $\alpha' = \alpha^2$ and $\gamma = \beta^2/(2\kappa^2)$. Hence, with probability at least $1 - 1/n$, the number of subsets of size k that fall in any of the $6mn\sqrt{\log n}$ cells is at most the value in (5.14). By this we can bound the maximum number of points that fall in any of the cells of the outer grid in the following way: If there are N subsets of size k, then the number of elements X is at most $kN^{1/k}$ using $N = \binom{X}{k} \geq (\frac{X}{k})^k$. In our case, we get that the

maximum number of points in an outer grid cell is with high probability at most

$$k\big(6\alpha' \cdot m^2 n\sqrt{\log n} \cdot k!(\gamma \log^2 m(k \log k + \log m))^{k-1}\big)^{1/k} = O\big(\log^5 n \log\log n\big)$$

by choosing $k = \log n$ and noting that $n^{\frac{1}{\log n}} = O(1)$, $k! \leq k^k$, and $k \log k + \log m$ is in $O(\log n \log\log n)$. □

Proposition 5.10. *The maximum edge length in G is in $O\big(\log^3 n\sqrt{\log\log n}\big)$.*

Proof. We have seen in Lemma 5.8 that an edge in the MIS graph can have length at most $2|v_u^L - v_v^L|\sqrt{\log n}/\sqrt{L}$ after projection. Hence, using the fact that the embedding is non-expansive, an edge $\{u, v\}$ spans at most $2x_{uv}\sqrt{\log n}$ cells along each dimension in $C_{\sqrt{L}}$. By Lemma 5.9, each cell of the grid $C_{\sqrt{L}}$ is divided into $O\big(\log^5 n \log\log n\big)$ cells, or $O\big(\log^{2.5} n\sqrt{\log\log n}\big)$ cells along each dimension. Keeping in mind that, by Condition (5.3) in (LP), $x_{uv} \leq 3$, the total number of cells in the final grid $C_{\sqrt{LM}}$ that are spanned by an edge of G_M along each dimension is at most $O\big(\log^3 n\sqrt{\log\log n}\big)$.

The edge lengths between nodes not both in V_M are off only by a constant factor. Suppose $u \in V_M$ and $v \notin V_M$, and v is assigned to $v' \in V_M$. Then $\{u, v'\} \in E_M$ because uvv' is a path of length 2 in G. If both $u, v \notin V_M$, then their respective assigned vertices $u', v' \in V_M$ are an edge in E_M since they are connected in G by a path of length at most 3. Thus, the final embedding has no impact on the big-O notation. □

Proposition 5.11. *The minimum distance between two non-neighbored vertices in G is in $\Omega\big(1/\sqrt{\log n}\big)$.*

Proof. For $u, v \in V_M$, $|p_u - p_v| \geq 1$ by the rescaling of the final grid. If $u \in V_M$ and $v \notin V_M$, then v was assigned to another $u' \in V_M$ and placed in the circle of radius $1/3$ there, thus $|p_u - p_v| \geq 1/3$.

For the nodes not in V_M, we need to examine the effect of the successive maximum clique assignment. First, note that the neighborhood of a node can be partitioned into a constant number of cliques. This is because a circle of radius 1 can be covered with a constant number of circles of radius $1/2$. Raghavan and Spinrad [120] give an algorithm to compute the maximum clique in a unit disk graph. Then with each successive maximum clique computed, at least a constant fraction of the remaining nodes is taken out because each graph is again a 1-hop neighborhood unit disk graph. Therefore, we are done after $O(\log |N'(v)|)$ steps, that is, the value of x in Algorithm 5.6 is bounded by $O(\log |N'(v)|)$. Non-neighboring nodes are assigned to different grid points, resulting in a minimum non-edge length of $\Theta\big(1/\sqrt{\log n}\big)$. □

Proof of Theorem 5.1. The theorem follows immediately from Definition 2.3 and Propositions 5.10 and 5.11. □

5.4.5 Polynomial Running Time

We now show that ViCE can be implemented in such a way that its running time is polynomial in n. In particular, we show how the set of constraints (LP) can be solved in polynomial time, even though there is an exponential number of spreading constraints.

To that end, we will construct a *separation oracle* [60] which will guarantee that we only need to solve polynomially many constraints polynomially often. A separation oracle for (LP) takes as input a set of lengths $\{x_{uv}\}$ and checks whether this set satisfies all constraints. If that is the case, a feasible solution is found, otherwise, the oracle returns one constraint which was violated by the input and a new solution satisfying all up to now reported constraints is computed. Constraints (5.3), (5.4), (5.5), and (5.6) can be checked explicitly in polynomial time. We cannot check the spreading constraints (5.7) explicitly since there are exponentially many subsets, but we can make use of the following observation.

The key to polynomial-time checking is that we can order the sets in such a way that if a set violates the spreading constraints, then the set of smallest edge weights also violates them. Hence, instead of checking the spreading constraints on all sets, we merely check them on those of smallest weight. For this reason, we compute the closest set of size k around vertex u for all $u \in V_M$ and check whether this set satisfies the spreading constraints. Specifically, for each vertex u, order the remaining vertices u_i in ascending order by distance from u, i.e., $x_{uu_1} \leq x_{uu_2} \leq \cdots \leq x_{uu_{m-1}}$. Then check for each $1 \leq k < m$ whether the set $I_k = \{u_1, u_2, \ldots, u_k\}$ fulfills the spreading constraint around u. If any of these $O(m^2)$ constraints are violated, then the separation oracle reports the first one found, otherwise Constraint (5.7) is fulfilled.

Theorem 5.12. *The ViCE algorithm can be solved in polynomial time.*

5.5 Discussion

One thing we can get out of the analysis of our virtual coordinates algorithm is that instead of dealing with the entire graph, it suffices to operate on a maximal independent set, giving only an extra factor of $O(\sqrt{\log n})$ to the approximation ratio. Unfortunately, this bound is tight if one uses the greedy maximum clique approach as we have done because it cannot approximate clique partition better than the $\log n$ factor. To see that, consider the following neighborhood of a node v, partially depicted in Figure 5.1. $N(v)$ consists of two rows of nodes a_i above and b_i below v, $1 \leq i \leq k$, each row on a line at distance exactly $1/2$ from v. At each spot a_i (or b_i), there are actually $|a_i| = 2^i$ number of nodes. The neighbors of a_i are b_i and a total of $\lfloor k/\sqrt{3} \rfloor$ other a_j's. The same holds for the b_i. An optimal clique partition would

group together the a_i and the b_i separately, resulting in four cliques. Now let $s_i = a_i \cup b_i$. A maximum clique in step j, on the other hand, is s_{k-j+1} because of the exponentially increasing sizes of the a_i and b_i. Therefore, the greedy clique partition will contain $k = O(\log n)$ sets instead of 4. Therefore, a constant approximation will need a completely different approach to locally approximate clique partition on a unit disk graph. One such approach was taken in [51], but the task was relaxed to finding a k-clustering: a partition into clusters of diameter at most k. While it is not explicitly stated in the paper, their solution only works for $k > 1$ and is thus not applicable in our case. ($N(v)$ is already a cluster of diameter at most 2.)

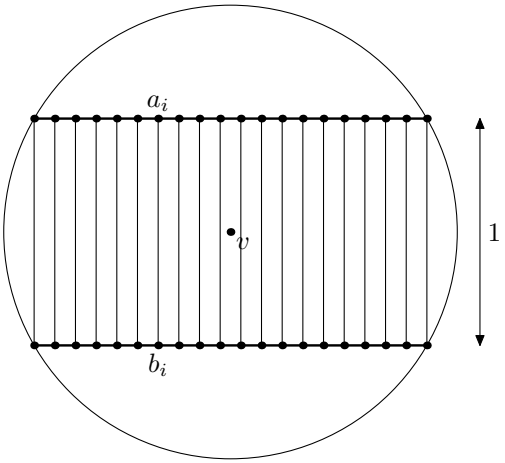

Figure 5.1: Instance where greedy maximum clique needs $\log n$ many cliques in the neighborhood of node v.

As for embedding only the MIS, this might be as hard as embedding trees. Ideally, a constant-approximation embedding of the MIS graph using a combinatorial approach is desirable, yet it is unclear whether that goal is even attainable.

Another dead-end for improvement is the use of volume-respecting embeddings for the metric. In [44], the authors show a lower bound of $\Omega\big(\sqrt{\log n}\big)$ on the volume distortion even for a path. Thus, using this approach will always give a poly-logarithmic instead of a constant approximation factor.

One issue overlooked so far is that of *robustness*. In [120], the authors actually provide a robust algorithm for the maximum clique problem in UDGs. A robust algorithm in this case takes as input any graph and outputs either a maximum clique or a certificate that the graph is not a UDG. We have two steps where the unit disk property is essential. One is in the final embedding

of the non-MIS nodes where we make use of the robust algorithm [120]. The other is the formulation of the linear program and in particular the spreading constraints. Since we cannot guarantee that the MIS of any given graph will fulfill the spreading constraints, we do not know whether a feasible solution exists in that case. However, if the MIS does fulfill the spreading constraints, a feasible solution will be found, otherwise we know that the input graph was not a UDG. Therefore, our algorithm is essentially a composition of two robust algorithms, resulting in a robust algorithm itself.

Chapter 6

Alternative Models

6.1 Network Models

It is clear that the unit disk graph is not the end of all wireless network models. As we have seen, despite its simplicity, it is NP-hard to even recognize a graph as a UDG let alone embed it. So the question begs itself: What other models are there? To what extent can we generalize and improve the unit disk concept?

One could first ponder the possibility of embedding a general graph in a UDG-like fashion. The following observations, due to [101], establishes a connection between embedding and the chromatic number. It hinges on the idea that finding a minimum clique partition is the same as coloring the complement graph, that is, it formulates the general idea of the last step of our algorithm involving clique partitions for general graphs. To define how a general graph should be embedded, we want to keep the requirement that non-edges should be farther than one unit apart. Then we ask how close can the edges be embedded. Formally, we have the following.

Definition 6.1. *For any graph $G = (V, E)$, an embedding $f : V \to \mathbb{R}^2$ is D-bounded if*

$$\{u, v\} \in E \Rightarrow \|f(u) - f(v)\| \leq D$$
$$\{u, v\} \notin E \Rightarrow \|f(u) - f(v)\| > 1$$

for all $u, v \in V$. Define $D(G)$ to be the minimum D such that there exists a D-bounded embedding of G.

For unit disk graphs, $D \leq 1$. For general graphs, we can establish upper and lower bounds as follows.

Lemma 6.1. *Let $D = D(G)$ for a graph $G = (V, E)$. Then*

$$\frac{\sqrt{\lambda}}{\sqrt{2} \cdot diam(G)} \leq D \leq \sqrt{2\lambda} \tag{6.1}$$

where $\lambda = \chi(\bar{G})$, the chromatic number of the complement graph of G (i.e., $\bar{G} = (V, \binom{V}{2} \setminus E)$.

Proof. The upper bound can be seen as follows. Consider a color in $\bar{G}$. It is an independent set in $\bar{G}$ and thus a clique in G. Then λ is equal to the size of a minimum clique partition of G. Cliques can be placed on the same point. A simple embedding places the λ cliques on a square grid of total side length $(1 + \epsilon)(\sqrt{\lambda} - 1)$ and the longest distance is between opposite corners, resulting in an extra factor of $\sqrt{2}$.

The idea for the lower bound is similar. Consider a D-bounded embedding of G with smallest possible D. Then the smallest enclosing square S which will contain all the embedded points of G will have side length at most $diam(G) \cdot D$. We can divide S into smaller squares with the property that in each new cell, all nodes form a clique. Note that the non-edge requirement implies that if two nodes have distance less than 1 in an embedding, then they are connected by an edge in G. The property then follows if the cells have width $1/\sqrt{2}$. Since each cell forms a clique in G, all nodes in it are independent in $\bar{G}$ and can thus be colored with the same color in $\bar{G}$. Therefore, the number of colors of $\bar{G}$ is at most the number of cells, which is $\leq (\sqrt{2} \cdot diam(G) \cdot D)^2$. □

Due to the link between D and the chromatic number, we can also show inapproximability in general.

Theorem 6.2. *For a general graph G, $D = D(G)$ cannot be approximated within $n^{1/2-\epsilon}$, unless $ZPP = NP$.*

Proof. Given a graph $G = (V, E)$, we can construct a graph $\tilde{G} = G \cup K_{\lceil n/2 \rceil, \lfloor n/2 \rfloor}$ where we partition V, $|V| = n$, into two (almost) equal sets, treat these as nodes of a complete bipartite graph, and add the additional edges of $E(G)$ to $E(\tilde{G})$. Then, by construction, $diam(\tilde{G}) = 2$. Consider what happens to the chromatic number of $\tilde{G}$. Adding the edges of $K_{n/2,n/2}$ to any graph will increase the number of needed colors by a factor of at most two (simply add one bit to the color of two conflicting nodes). Equivalently, deleting those same edges will diminish the chromatic number by at most one half. Hence, $\lambda/2 \leq \tilde{\lambda} \leq \lambda$. Since the chromatic number is not approximable within $n^{1-\epsilon}$, the theorem follows with Lemma 6.1. □

Moving on to more specific graphs, it is conceivable that wireless networks, modelled on the graph level, contain particular properties which set them apart from general graphs. The survey in [132] motivates and lists

several models which seem to lend themselves well to theoretical analysis. Independently, we can distill those properties of unit disk graphs which were necessary to ensure the approximation guarantees of the embedding algorithm of the previous chapter. The analysis of Chapter 5.4 shows that there are two main ingredients: that any MIS fulfills the spreading constraints and that we can embed the one-hop neighborhood of any (MIS) node without placing non-neighbors too close. The latter is straightforward in UDGs because the number of cliques in the neighborhood of a node is at most a constant and there is an algorithm which can compute a maximum clique in UDGs without the use of a realization.

One network model, suggested in [82] and termed *bounded independence graph* (BIG) in [132], generalizes the well-spread MIS idea. Let $N(v,r)$ denote the r-neighborhood of node v: the set of nodes at most r hops from v. Let M be a maximum independent set in the subgraph induced by $N(v,r)$. Then a graph G is a (polynomially) bounded independence graph if, for all nodes v and all $1 \leq r \leq diam(G)$, $|M| = O\big(r^d\big)$ for a constant d.[1] From the discussion of spreading constraints in Section 5.2.1, we immediately have that unit disk graphs are polynomially bounded independence graphs for $d = 2$. The idea is that the degree of the polynomial, d, represents the *dimension* of the graph. In that sense, the BIG model extends one of the unit disk graph properties to higher dimensions. Consequently, the spreading constraints (see Section 5.2.1) need to be reformulated to

$$\sum_{u_i \in S} x_{uu_i} \geq \kappa |S|^{1+\frac{1}{d}}$$

since the average distance between two independent points in S around a node is now in the order of $|S|^{1/d}$. Notice that Lemma 5.5 becomes

$$\sum_{u_i \in S} \frac{1}{x_{uu_i}^d} \leq \frac{\log m}{\kappa^d}$$

so the target space for embedding is $\mathbb{R}^d$. In contrast to the maximal independent set, the remaining nodes cannot be placed such that the quality measure is constant or even poly-logarithmic in general BIGs. This is a corollary to the following lemma and Lemma 6.1.

Lemma 6.3 ([80]). *There exists a graph on n nodes such that for a constant $d < 1$, a maximum independent set M and a minimum clique partition P, it holds that*

$$|M| = 1 + 2/d + o(1) \quad \textit{and} \quad |P| = \Omega\Big(n^{1-d}/\log n\Big) .$$

[1] This definition can, of course, be generalized to arbitrary functions: G is an f-BIG if $|M| \leq f(r)$ for each r and all nodes v.

Proof. The existence of such a graph can be shown via an application of the probabilistic method. Let $G_{n,p} = (V, E)$ be the graph on n nodes where each edge is included, independently, with probability p. Denote by $I(k_1)$ the random variable counting the number of independent sets of size k_1. Then $I(k_1) = \sum_{S \in \binom{V}{k_1}} I_S$ where I_S is the indicator variable for set S being independent with $\text{Prob}[I_S = 1] = (1-p)^{\binom{|S|}{2}}$. Further denote by $C(k_2)$ the random variable for the number of cliques of size k_2. Analogously, $C(k_2) = \sum_{S \in \binom{V}{k_2}} C_S$ with the probability $\text{Prob}[C_S = 1] = p^{\binom{|S|}{2}}$. Then, by the linearity of expectation, we have

$$\text{E}[I(k_1)] = \binom{n}{k_1}(1-p)^{\binom{k_1}{2}} < n^{k_1} \cdot (1-p)^{\binom{k_1}{2}}$$
$$\text{E}[C(k_2)] = \binom{n}{k_2} p^{\binom{k_2}{2}} < n^{k_2} \cdot p^{\binom{k_2}{2}}$$

which we can combine with Markov's Inequality

$$\text{Prob}[I(k_1) \geq 1] \leq \text{E}[I(k_1)]$$
$$\text{Prob}[C(k_2) \geq 1] \leq \text{E}[C(k_2)].$$

Now we can choose appropriate values for k_1, k_2, and p such that both

$$\text{E}[I(k_1)] < 0.5 \qquad \text{and} \qquad \text{E}[C(k_2)] < 0.5 \tag{6.2}$$

hold. This implies that $\text{Prob}[I(k_1) \geq 1 \text{ or } C(k_2) \geq 1] < 1$, or equivalently

$$\text{Prob}[I(k_1) < 1 \text{ and } C(k_2) < 1] > 0 \tag{6.3}$$

which states that there exists a graph where the maximum independent set has size less than k_1 and the maximum clique contains less than k_2 nodes, implying that a minimum clique partition needs at least n/k_2 cliques.

It remains to examine the possible values for k_1 and k_2. The goal is to have a constant-sized MIS and a large clique partition. Now choose $p = 1 - n^{-d}$ for a constant $d < 1$. Then the condition in (6.2) becomes

$$\text{E}[I(k_1)] < n^{k_1} \cdot \left(\frac{1}{n^d}\right)^{\binom{k_1}{2}} < 0.5$$

which holds if we choose k_1 such that

$$k_1 \geq \frac{2}{d} + 1 + \frac{\ln 2}{\ln n}$$

which can be fulfilled for a constant k_1. For the cliques, we need

$$\text{E}[C(k_2)] < n^{k_2} \cdot \left(1 - \frac{1}{n^d}\right)^{\binom{k_2}{2}} < 0.5,$$

taking the logarithm gives

$$k_2 \cdot \ln n + \binom{k_2}{2} \ln\left(1 - \frac{1}{n^d}\right) < -\ln 2$$

where we can approximate $\ln(1+x) \leq x$ so that the above condition is equivalent to

$$\ln n + \frac{\ln 2}{k_2} < (k_2 - 1)\frac{1}{2n^d}$$

which can be fulfilled for a k_2 in $O\big(n^d \log n\big)$, proving the lemma. □

Instead of increasing the dimension as we have done with the BIG, we can change the underlying metric space. For instance, we keep Definition 2.2 but generalize Definition 2.1 to require that the embedding f maps into $\mathbb{R}^2$ but, instead of the l_2 norm, there can be a different metric ρ associated with it. The target space does not even need to be $\mathbb{R}^2$. If ρ is a doubling metric, then the resulting graph has been coined a *unit ball graph* (UBG) in [85]. A metric is a *doubling metric* if every ball of radius r can be covered by a constant number of balls of radius $r/2$. The ball here refers to the points in the metric space, not the graph metric. In general, we can associate to each metric a *doubling dimension* which is the smallest d such that every ball can be covered by at most 2^d balls of half the radius. The above definition of UBG requires a metric with a constant doubling dimension, which the l_2 norm certainly is. Since a UBG is a BIG [81], the spreading constraints for any MIS apply and the first part of the algorithm remains the same. As for the placement of the non-MIS nodes, the constant doubling dimension ensures that the minimum clique partition is bounded by that constant. An approximation ratio of a to clique partition will increase the overall approximation ratio of the embedding by that factor a. In fact, the algorithm in [120] used for computing the maximum independent set (of a neighborhood) also applies to UBGs in two dimensions. Note that the above discussion implicitly assumes that the target embedding space for an algorithm is Euclidean and not the original metric space for which the UBG was defined (in particular, see Sections 5.4.1 and 5.4.3). So far, the focus of the literature has been on taking any (finite) metric and embedding it into l_2 for a particular dimension. One could contemplate the worth of computing an embedding of the nodes of the unit ball graph into the metric for which it was defined, along with what the application domain in this case would be.

The quasi unit disk graph (qUDG) model has been proposed as a generalization of unit disk graphs [20, 89]. A graph $G = (V, E)$ is a d-qUDG, for $d \in (0, 1]$, if there exists an embedding f such that $\|f(u) - f(v)\| \leq d \Rightarrow \{u, v\} \in E$ and $\|f(u) - f(v)\| > 1 \Rightarrow \{u, v\} \notin E$. The behavior between d and 1 is not specified. Again, the term embedding can be considered in the original sense or with the alterations discussed above. The range of the

resulting combinatorial graphs is wide. If $d = 1$, then we have the standard unit disk graph. If $d \to 0$, then G tends towards a general graph. If we fix d, then G is a UBG [132]. The lower bound for UDG embedding [84] can be stated in terms of quasi unit disk graphs: It is NP-hard to embed a d-qUDG as a d'-qUDG with $d' \geq 1/\sqrt{2}$ such that $d' > (\sqrt{2/3})d$. It is an open question what kind of graphs result for other ranges of d.

Going in the other direction, instead of embedding more general graphs, we could restrict the types of unit disk graphs under consideration in order to potentially simplify the problem. We could imagine that certain network characteristics (such as density or other degree bounds) can be defined such that we can give a simple algorithm with strong guarantees. We already saw from the simulation results and the discussion in Chapter 3 that when densities are high, the network is "nice" so that a single heuristic works well. Several other simulation results from prior work, as discussed in Chapter 2, bear further witness to this fact. The goal is to give these results a theoretical underpinning. A likely source for an explanation is illustrated in [90] where simulations show that the ratio of the span of a shortest hop path approaches 1, in particular, starting at node densities of around 10 nodes per unit disk on average. The span is defined as the ratio of the length of the (weighted) shortest path in the graph to the actual distance. In other words, the true Euclidean length becomes proportional to the graph distance between two nodes. As this discrepancy was one of the fundamental difficulties in the embedding of general unit disk graphs, we can ask, at least on average, how much easier the problem becomes. Further support for this approach can be found in [6] which, among other things, investigates the rigidity properties of random geometric graphs. It is shown that, if n nodes are randomly deployed in the $[0, 1]^2$ square by a Poisson process and the radius r of connectivity is chosen such that $r \in O\left(\sqrt{\log n/n}\right)$, then the resulting (unit disk) graph is globally rigid and thus uniquely realizable with high probability and such a realization can be found efficiently. The expected time until all nodes have computed a position depends on the density of the anchors nodes, with the longest time in $O\left(\sqrt{n/\log n}\right)$ for the case of three anchors within each other's transmission radius. It remains to be seen how to take this analysis further to other degree and diameter combinations as well as how to extend this to approximation algorithms instead of unique realizability.

In a completely different point of view, fundamental results in the theory of wireless networking suggest to move away from the graph and on to a physical model [61, 103]. The reason for this view is that a graph model is inherently static and binary with respect to the existence of a link between nodes. Since the network is assumed to have a purpose, the ongoing communication, stemming from the application, changes the physical characteristics of the wireless "connection" between two nodes. A premium model that describes this effect is the signal-to-interference-plus-noise (SINR) model [123]

which describes the physical characteristics necessary for the reception of a signal. It states under what conditions communication is possible. The main motivation is that communication of other nodes causes interference which should be modelled as a continues variable affecting all other nodes. The key observation is that the "links" change even if all the nodes are static. Since the problem of embedding a network is only concerned with the static (or at least current) geometric layout of the nodes, this additional consideration of communication effects is irrelevant. In fact, it is only the distance between sender and receiver which is of main concern. The definition of the task then depends on the input. If the distance between (all or some) nodes are known then this is exactly in the domain of metric embeddings and is trivial if all pairwise Euclidean distances are given. If the input comes in any binary form describing the physical effects on the links, such as a graph (for each time step, for example), then it is not clear whether any reasonable embedding can be computed at all. Should a different embedding be computed for every time step? Or one embedding after a certain amount of time? What if different communication request patterns produce completely different embeddings for the same static layout of the nodes? The conclusion here is that physical models are suitable for studying the communication characteristics of wireless networks, but not static properties such as their geometry or layout.

6.2 Approximation Quality

One could imagine alternatives to our definition of the approximation quality of a virtual coordinates algorithm (cf. Definition 2.3). For instance, we could consider the following simple measure. Suppose we have computed an embedding f of $G = (V, E)$, a unit disk graph. Then we count the number of node pairs which are embedded incorrectly:

$$w = |\{\{u, v\} \notin E \mid \|f(u) - f(v)\| \leq 1\}| + |\{\{u, v\} \in E \mid \|f(u) - f(v)\| > 1\}|$$

and define the quality of f as

$$\tilde{Q}(f(G)) := \frac{w}{\binom{n}{2}} \tag{6.4}$$

because there are $\binom{n}{2}$ node pairs with $n = |V|$. In other words, we do not care by *how much* a distance is distorted, only *how many* of them. This type of definition makes sense in other areas of computational complexity. For instance, consider the NP-hardness proof [29] and its reduction to 3SAT. The MAX3SAT problem variant, while also NP-hard, considers the task of maximizing the number of true clauses. For the virtual coordinates problem, we can give an embedding f such that $\tilde{Q}(f(G)) = 2$ for any UDG G. Let $m_1 = |E|$ and $m_2 = \binom{|V|}{2} - |E|$. If $m_1 \geq m_2$, then there are more edges than

non-edges and we embed all nodes as a clique, that is, all nodes are placed arbitrarily inside a circle of unit diameter. Then $w = m_2 \leq (m_1 + m_2)/2$. Otherwise, $m_2 > m_1$, and we embed the nodes on a line at distance, say, 2, that is, $f(v_i) = (2i, 0)$. Then $w = m_1 < (m_1 + m_2)/2$. In either case, $\tilde{Q}(f(G)) = 2$.

The difference with the quality definitions is most apparent when we consider what the above discussion implies for the complexity of the UDG embedding problem. With the definition of $\tilde{Q}$, the virtual coordinates problem lies in APX. On the other hand, using our original definition, it is not clear whether there is such a polynomial-time constant approximation algorithm. We are tempted to conjecture that no such algorithm exists. Any other alternative embedding quality definition must somehow have a "sense of scale." It must necessarily involve the actual distances in the embedding since both the definition of UDG as well as the problem involve distances in $\mathbb{R}^2$.

An example of an approach which combines measuring distance distortion with the number of distorted distances is given by Kleinberg et al in [35, 75]. They introduce the notion of *slack* which is the percentage of all distances which are not embedded correctly. However, motivated by Internet latency measurements, the input to their problem is not a unit disk graph, but rather a subset of inter-node distances [75] or other general metrics [35], in particular those of low doubling dimension. In the field of study of metric embeddings, they are able to show that any finite metric can be embedded, with constant slack and constant distortion, into constant-dimensional Euclidean space. It might be worthwhile to consider a similar notion of approximation in the unit disk graph case where we ask how much slack is necessary for a constant quality embedding.

Given that geometric routing is a distinguished motivation for studying virtual coordinates, we can define the quality of an embedding in terms of its effect on routing. In fact, already [122] considered constructing coordinates for the expressed purpose of georouting. Recently, evidence has been mounting [133, 134] that already small errors in the location of nodes leads to a significant degradation in the performance of geometric routing as far as successfully reaching the destination is concerned. A good embedding algorithm in light of georouting can be defined in several ways. One goal would be to compute virtual coordinates such that optimal georouting is still possible. In the quasi unit disk graph model, it is known that optimal georouting is possible for $d \geq 1/\sqrt{2}$ if G is a d-qUDG [89]. On the other hand, the lower bound [84] states that embedding G as a d-qUDG for $d > \sqrt{2/3}$ is NP-hard. Thus we have a narrow gap of embedding G to ensure optimal routing performance.

Instead of looking for an optimal geometric path, one could request that the coordinates represent the network sufficiently such that a greedy forward-

ing strategy will always succeed if the graph is connected. In other words, the virtual coordinates do not necessarily need to reflect the actual coordinates (if the input graph was a UDG), but they should reflect the *routing structure* of the graph. Two existing approaches go in that direction. In [145], the idea is to determine a (small) set of anchors from which the other nodes compute their "positions" as a vector of hop distances to all anchors. The goal is to be able to use the greedy routing paradigm in order to route on these positions. More generally, the aim of *compact routing* schemes (for instance, see [140]) is to encode the graph or specifically the routing structure as compactly as possible at each node. This can be done either in form of a label chosen for the node, or, in the so-called name-independent version, as local storage. All of the above described mechanisms have nothing to do with the input graph being a unit disk graph. The objective of these algorithms is geared specifically towards optimizing the routing process instead of obtaining a general "picture" of the graph where routing is only one of the applications. Since routing on networks is in itself already a complex problem, in particular in the face of mobility, we will now turn our attention to it in the second part of the dissertation.

Part II

Mobility

Chapter 7

Mobile Networks

Why are wireless ad hoc networks such a hot research topic? The two main difficulties, decentralization and dynamics, are already present in the Internet. Routers and cables are largely provided by autonomous systems with their individual business goals; the hardware infrastructure, both on a global scale as well as in local area networks, is subject to failure and user reconfiguration. The aspect which wireless ad hoc networking brings into the picture is that it greatly magnifies these challenges. Every node can be seen as its own, perhaps even selfish and malicious, autonomous agent. Initially, the nodes might not even be aware of each other, with no bootstrapper to contact. Nodes are allowed to move around in an unpredictable fashion. Wireless links are more unreliable than cables. All these issues promote network dynamics in every sense from a side concern to a major research task.

If we consider networks to be mobile (or, more generally speaking: dynamic), then "mobility" is an integral part of all network tasks. It is not an isolated issue which we can consider solely on its own. The point of view we adopt in this dissertation is that mobility should be seen in terms of its *effect* on network operations. After pessimistic results about the limited capacity of wireless networks [61], follow-up work has shown that mobility can in fact increase the capacity [58]. Another positive effect is the use of mobile robots to enable computations which might otherwise prove intractable [119]. On the other hand, if mobility is completely outside of our control, then its effect is often negative. A node might have to perform an expensive recomputation of its routes if some nodes move out of its communication range. In this part of the dissertation, will focus on this adverse aspect of mobility: the additional costs it places on the nodes.

The fact that mobility is a common ingredient of all wireless ad hoc network problems makes it a key research issue but at the same time difficult to capture. Thus, one strategy to overcome this difficulty is to narrow down the scope. We will focus on one of the quintessential tasks in communication

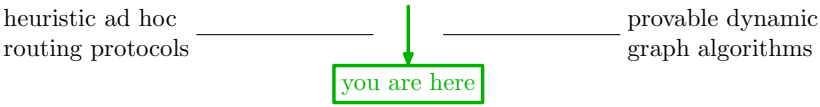

Figure 7.1: Orientation map for Part II.

networks: routing. The central theme of this part of the dissertation can now be refined to the question of how mobility changes the way we need to look at routing. We want to approach the topic from a theoretical angle because, up to this point, there is no consensus on which practical application scenarios will dominate the use of mobile ad hoc networks (MANET). Even the IETF MANET working group currently specifies both a proactive and a reactive routing protocol [34] in order to be prepared for any type of mobility. Our mind set is best described by a picture as in Figure 7.1. On the left, we have a virtual jungle of protocol suggestions for MANETs, all of which fail to make any general performance and cost guarantees. On the right, we have the set of proposed algorithms for various tasks in dynamic graphs along with their complexity proofs. Unfortunately, the latter often impose unrealistic or too harsh restrictions and provide impractical guarantees. Ultimately, what we strive for are provable statements which are specific enough to sort out the key routing concepts and general enough to learn the fundamentals of the mobile routing process.

7.1 Definitions of Mobility

Perhaps the biggest challenge facing researchers studying wireless networks today is the task of capturing their dynamics. Connections are unstable due to the inherent properties of wireless links; users are mobile and can move in and out of each other's transmission ranges. So how exactly does one define mobility?

On a first level, we must distinguish scenarios by how much mobility is within the network administrator's control. In some cases, we can introduce a mobile entity, such as a roaming robot, into a static network to simplify or even make possible certain tasks [119]. In other cases, all the nodes may be mobile and they await instructions from the network in order to control their paths, as in the task of motion coordination [98]. In contrast, we look at the case where mobility is completely outside of our control. The mobility of the wireless network is given in combination with the dynamic link properties. Since agreement on real-world application scenarios and their implications for network characteristics is still lacking, we must attribute a large degree of unpredictability to the way that a dynamic ad hoc network changes over time.

7.1.1 Mobility Models

One option to define mobility is by specifying a typically randomized algorithm which describes the path of nodes, usually moving about in the Euclidean plane. These so-called *mobility models* are the literature standard for evaluating ad hoc routing algorithms in simulation. We will briefly survey the most common ones and otherwise refer to overview articles such as [33, 128] for more details. It is important to note that these models are specific to wireless ad hoc networks. In the case of cellular networks which involve mobile users and fixed base stations, there has been significantly more progress [146], probably due to the availability of concrete use cases. In the descriptions below, it is usually assumed that the simulation area is bounded and nodes move with fixed velocities. Both cases can be extended. There has been work on boundless simulation area models [62] as well as on including an acceleration process [22].

One of the earliest models to describe chaotic motion is the *random walk* (RW) model, also known as Brownian motion. Each node picks a velocity (within given bounds) uniformly at random and follows that for some time t. The choice is independent of the node's history. There are numerous refinements to the concept, such as the probabilistic version in [36]. The *random direction* (RD) model [125] is basically the same as the random walk model above, except that a node follows its path to the boundary of the simulation area, pauses there for some time (chosen randomly within given bounds), and then chooses its new velocity. In the *Gauss-Markov* (GM) model [141], the focus is more on movement changes. At fixed time intervals, for each node, a new velocity is chosen based on the current value and a tuning parameter given by the input. This allows for a variety of movement patterns which produce smoother node traces than the above two models.

The most common mobility model currently in use for simulation and analysis is the *random waypoint* (RWP) model. Its description is simple: Each node chooses a random location in the simulation area, goes to that point in a straight line with random speed, and then pauses for some random time. The speed and pause times are usually given in terms of an upper and lower bound, and both are chosen uniformly at random within that interval. A major drawback for using models such as RWP in simulation is that the configuration of nodes evolves; after some time, the nodes cluster in the center and are thus not spread out uniformly over the simulation area. This so-called *steady state* is reached only after a significant number of simulation steps. In [95], the authors give an analytical framework for deriving steady state properties, shown in particular for the RWP model. They highlight the main deficiencies in standard simulations, speed decay and a long time to reach steady state. The steady or "stationary" state is also discussed in [94]. Further refinements to the above models and RWP in particular have been proposed, most notably the introduction of *obstacles* [68].

We can identify another set of models which contains what we term a *social component*. The most prevalent concept is that of *group mobility* [2, 66, 92]. Instead of all nodes moving individually and independently, we can think of clustering the nodes into groups. These can be fixed for the entire time in question or nodes might be able to migrate from one to another under certain conditions. All the nodes within the group may travel with the same velocity, or they can randomly move around a group leader within certain bounds. Another socially motivated concept is to restrict node movement to certain lines representing city streets or highways [40, 99].

The above discussion shows that the mobility models widely used today (in simulation and separate analysis) have one or more of three main ingredients: randomness, geometry, and social components. The primary goal of Part II of this thesis is to move away from all of those assumptions and to look at mobile routing from as general a view as possible. We are interested in the behavior of algorithms in a worst-case sense, since we do not believe that real-world applications generally exhibit nice uniform randomness properties. We also do not believe that geometry alone captures the complexity of the physics of wireless links, regardless of the interference which arises due to communication.

"Worst case" mobility is considered in [131], but the nodes are allowed to adapt their transmission ranges such that they are significantly greater than the speed or acceleration in order to ensure stable links over time. Thus, the effect of mobility on the actual communication graph is diminished. In [43] and other follow-up work, the authors model a set of stationary nodes, called virtual nodes, in terms of the actual moving nodes. Unfortunately, no analytical bounds for non-average cases are given and the model needs to be developed further to show its use for routing and other applications.

7.1.2 Mobility Metrics

The above mobility models give a description of how to produce a particular set of traces or paths in time for the nodes. Different patterns can also be produced with different input parameter values. What they lack is a common denominator to describe "how much" mobility a particular instance has. It should also enable a standardized comparison between different models. What we need is a *mobility metric* in order to describe the amount of dynamics an application has to deal with. It should convey the relative difficulty of a scenario.

Initial work in this direction [30] suggests to use *pause time*, the length of time that nodes remain stationary between movements, as a characterization of mobility. Later, [33] evaluates the DSR [70] routing protocol under different mobility models where the measured quality values are a function of the node speed. This suggests that *node speed* is the key parameter governing the quality of the mobility models. Unfortunately, both approaches are too

simplistic as shown in later work (see, for instance, [128]) and are by far not general enough to cover all of the aforementioned models.

A first step towards a mobility metric is taken in [93]: They propose to measure the *change in average distance* A_v from one node v to all other nodes between successive simulation time steps (Δt). This is then averaged over the whole simulation time T and all nodes. In follow-up work [69], they suggest to use *relative node speed*: Determine relative velocity between two nodes at a given time, take the length of that vector (speed) and average this over the whole simulation time and all nodes. A small experimental evaluation shows a linear correlation between the mobility (in their measure) and the average number of link changes. Later, Kwak et al [91] find that this does not hold generally, and they take these ideas further to define an abstract mobility measure which greatly correlates with the number of link changes. In other words, it is ultimately the number of link changes which the mobility metric describes.

The study in [142] shows that the expected route lifetime is the important mobility metric. The obvious difficulty lies in calculating as well as measuring it in large-scale simulation. They claim that it can be estimated with the minimum hop lifetime of the network; this was shown not to be the case in [128].

Finally, the focus has shifted on identifying the *effect* of mobility on routing protocols. [17] highlights different geometric characteristics of mobility models and studies the effects of mobility on the connectivity graph and thus on routing. For the connectivity graph metrics, the authors find the following: 1) average link duration is higher for group mobility models; 2) the same holds for the average shortest path duration; 3) average number of link changes does not differentiate between the models; 4) average path availability (i.e., whether nodes are connected) is high in most cases (otherwise routing is not interesting anyway). In summary, (1) and (2) imply that the metrics "average link duration" and "average shortest path duration" create different dynamic graphs and can thus be used to differentiate between mobility models on the connectivity level. Interestingly enough, despite identifying the above graph properties, the routing protocol evaluation is still based on the maximum speed concept. The results are as expected: No single protocol has highest throughput and lowest overhead, the proactive (DSDV [115]) and reactive (AODV [116] or DSR [70]) intersect at different speed values. In other words, again there is no global mobility measure, absolute speed values specific to the scenarios are used instead.

In [18], a follow-up on [17], the authors give more detailed statistics of link and path duration. Also, with simple analytical models, they show that the overhead (and throughput) of DSR is inversely proportional to the average path duration. This time, path duration is defined properly over all paths and not just the shortest path. However, in their simulations, they

take (equally-spaced) snapshots of the graph and consider the (currently) shortest path instead of all paths.

The master's thesis [128] also aims at studying the effects of mobility on reactive routing. The main conclusion, as we also know from earlier work, is that the *path lifetime* is a critical mobility parameter, yet hard to measure and/or estimate. It proposes a heuristic method to estimate the route life time (RLT) distribution of the shortest path between two random nodes. This is done by estimating the link life time as well as the distance distribution of the nodes. The median of the distribution is then used as a measure of mobility hardness. From the simulation, the thesis concludes that the RLT distribution (and in particular his measurement of it) alone is not enough to measure the effects of mobility on routing, but that the concept of route life time is nonetheless a key ingredient.

These last mentioned papers are the most recent and also most relevant to our ideas, as we will deal with mobility in terms of its *effect* and not its *generation*. However, all the above works tend to focus on a single mobility parameter, instead of focusing on the coupling with routing, despite the fact that [128] points at that the problems stem from unknown actual users movements *as well as* communication patterns, i.e., route requests. That is why we propose to study the *mobility ratio* in Chapter 9, putting the effects of mobility in relation to route requests.

7.2 Ad hoc Routing Protocols

As our central topic is mobile routing, we move our discussion of related work from mobility models to routing protocols. In the Internet, there is a hierarchy which leads to highly organized routing. Ad hoc networks, by their very nature, are flat, and we are free to impose whatever control structures on the network we choose. We will survey some of the main routing proposals for homogeneous dynamic networks in this section.

7.2.1 Mobile Routing Heuristics

We will first give a brief overview over the common concepts and some protocols found in the literature on mobile ad hoc routing protocols. We refer to [64] for a comprehensive fairly recent survey, and, to a lesser extent, to [30, 121] for older works. Note that the emphasis in this subsection is on heuristics; the next subsection will briefly mention some results on routing algorithms for which performance guarantees have been given. We will mention crossreferences between the heuristics and the algorithms where applicable.

The first proposals for mobile routing stem from adaptations of fixed infrastructure routing algorithms to handle the increased dynamics of ad hoc networks. Thus it is not surprising to find the concept of *proactive* protocols:

Nodes respond to changes in the network as soon as they are detected in order to keep their routing tables up-to-date. The most prominent protocol is DSDV (Destination-Sequenced Distance Vector) [115], which we can describe as "distance vector goes mobile." It focuses on how to modify standard Bellmann-Ford type routing mechanisms to avoid loops. Basically, this is what early routing focused on: loop freedom in the face of high dynamics. The link reversal algorithms, described in Section 7.2.2, technically also fall into this category, since they are basically proactive and loop freedom was the main property (until costs were investigated much later).

As the mobility under consideration increased, researchers turned more towards *reactive* or on-demand protocols. Instead of keeping data structures current, the nodes only deal with finding and updating their information as route requests come in. Thus the focus shifted from responding to mobility to responding to route requests. One of earliest proposed reactive ad hoc routing protocol is DSR (Dynamic Source Routing) [70, 71], where the source keeps a list of cached paths to each destination and floods if old routes do not work. See also [33] for more simulation results for various mobility models. The link reversal concept (discussed below in Sec. 7.2.2) has been put into practice by TORA (Temporally-Ordered Routing Algorithm) [114]. The main proclaimed advantage of link reversal is that it avoids flooding. The heuristics in TORA are necessary for dealing with partitioning and link joining problems. Perhaps the most popular protocol, right next to DSR, is AODV (Ad hoc On-demand Distance Vector), first described in [116]. It is the reactive counter-part to DSDV in that it adapts distance vector to deal with mobility, but in a reactive instead of proactive fashion. Its main conceptual difference to DSR is that the routing is done point-to-point, instead of writing the entire route into the packet header as with DSR.

In between proactive and reactive routing there is also the notion of *hybrid* protocols such as ZRP (Zone Routing Protocol) [63] and IZR (Independent Zone Routing) [126]. The idea is to define clusters (zones) in which to route proactively and outside of which to route reactively. There has also been a discussion of hierarchical-type ad hoc routing, which we can essentially describe by clustering approaches. For example, routing can be done via a backbone that is formed by a dominating set [87]. The biggest problem with this approach is how to maintain the clusters in the face of mobility. Also, the compact routing paradigm can broadly be classified as hierarchical routing. See for instance [1] as one of the recent papers on compact routing for general graphs. Briefly, the basic idea is that we want information similar to that provided by distance vector, but with considerably less memory overhead. Since routing tables are kept short by storing a only limited information about distant nodes, usually called landmarks, we can see this as forming a suitable routing hierarchy. These landmarks are then responsible for the local routing. In static graphs, a good selection of landmarks can be

guaranteed. The problem is updating and maintaining these fairly complex data structures. In dynamic graphs, it is not clear how much of this information can be retained if mobility is more than a few local changes. So, in a way, the hybrid protocols are the heuristic mobile counter-part to (dynamic) compact routing efforts, discussed further in Section 7.2.2.

For the case of highly mobile networks, the recent work of [4] looks at different routing strategies for what they term "extremely mobile networks" where the network may never be one connected component, but there exists a "communication path" in time. They propose to route via an information aging strategy (FBG = Forward to Best Gateway), the effectiveness of which in random instances is shown by simulation.

Another approach which falls somewhere between this and the next section is what has been called dynamic addressing [47]. In the spirit of virtual coordinates, nodes have a permanent ID as well as a transient routing address, indicating its location in the network. The routing scheme proposed in [47] is proactive. The idea is similar to the P2P approach [3] of a hierarchical organization with binary addressing, as well as dynamic distance labeling [77]. Virtual coordinates solely for the purpose of routing can also be found in [145], but only for special network topologies. While the last two mentioned papers are of theoretical nature, trying to tie down the model and give analytical bounds, [47] describes a heuristical approach and evaluates it for randomly and mostly uniform networks.

7.2.2 Dynamic Routing Algorithms

The protocols cited so far have been heuristic proposals, some dynamic routing algorithms with provable guarantees of one sort or another have also been proposed. We will briefly take a look at some of the currently known results.

The above mentioned idea of compact routing, or its companion problem of distance labeling, has seen its first adaptations to dynamic graphs. In a first step, [78] looks at dynamic labeling in trees. The main drawback is that dynamics are restricted so that the tree can only grow and shrink at the leaves. Dynamic routing in general graphs has been looked at only very recently [77]. Again, the price we pay for having provable guarantees is a weak model: Edges are allowed to change their weight by one unit (but not be deleted or added), and the graph has to remain stable for a while. The second part is similar in spirit to distinguishing the routing and the mobility phases as we do in Chapter 9. The first part goes in the opposite direction of ours, since we are interested in deletions and additions but do not yet consider weight changes. Interestingly enough, message complexity for the update procedure is dominated by D, the local density, which also plays a key role in our analysis of Chapter 9. The basic idea of [77] is to keep information spread throughout the graph at exponentially growing distances, and the goal is to have a routing scheme with low storage overhead as well

as low stretch.

Also recently, [5] considers routing on specific topologies (hypercube, mesh) when edge faults occur at random, that is, mobility is modelled as an edge fault probability.

One direction to deal with dynamic routing is to assume that nodes will want to send large streams of data and thus the goal is to optimize throughput. It started as flow considerations [12, 9] where the source keeps sending packets to the destination. The main idea is that of local balancing: If a neighbor has a shorter queue than me, then send the packet to him. We can also view this as a potential function. It can be shown that if a feasible flow exists, then such an approach will find it, despite adversarially changing edge capacities (because a path with a potential drop exists given that the destination is constantly absorbing packets). These ideas culminate in [10], which is basically optimal with respect to worst-case throughput. Intuitively, competitiveness is achieved because if the adversary wishes to harm the algorithm, it has to harm a lot of paths, in which case also the optimal path is long. Note that the competitive ratio is with respect to one fixed (best) path. It also builds upon [11] which looks at continuously finding the shortest path by getting feedback from the network, also effective against an adaptive adversary.

The heuristic idea of information aging discussed above ([4]) forms the basis for Last Encounter Routing. It has been shown analytically to perform well for the random walk [59] and random waypoint [127] mobility models.

The link reversal approach to dynamic routing dates back to 1981. Instead of maintaining complex routing tables, the edges are turned into directed links which ultimately produce a directed acyclic graph where the (single) destination is the unique sink in a steady state. The implementation can be done via a height or potential function. (Thus, we can see this as the single-packet counter-part to the local balancing approach above.) Initially, this approach was shown to stabilize in a finite number of steps [54]. More than twenty years later, [32] analyzes the time and message complexities of the total and partial link reversal algorithms for the recomputation of heights to produce another correctly oriented graph.

7.2.3 Dynamic Graph Algorithms

Networks are typically modelled as graphs, where links represent the ability to communicate between nodes. Dynamic graph algorithms have been around long before the fairly recent trend of ad hoc networking research. Consequently, a variety of standard graph problems have been studied in which the graph may change over time (see, for instance, [27, 46, 50, 56]). A related area to ours is that of peer-to-peer (P2P) networks. There, the dynamics stem mainly from the frequent and abrupt leaving of old peers and joining of new peers. Maintaining efficient data structures in the face of high

churn has also been looked at, where [86] provides worst-case guarantees, but only for a bounded number of nodes which can join or leave in each phase.

7.3 Roadmap

Before providing an outline for the remainder of this part, we should mention a few words on the motivation for the abstractions we will use. First of all, we must establish what the goal of a routing algorithm for a wireless ad hoc network is. Should it be energy, delay, or even bandwidth? The latter is unlikely, especially in comparison with the current standards which customers of fixed networks are accustomed to. In order to achieve high throughput, working on more stable wireless technology seems to be a key first step. If speed, or message delay, is the main criteria, then a solution such as flooding every time might be best. However, a large energy drain currently seems to be one of the biggest concerns of wireless networks; and sending a message over a wireless link is costly. Therefore, the message complexity of a routing algorithm is a good candidate to give an indication of the protocol's performance. In order to isolate the issue, subsequently we look at routing individual packets, not entire streams. The focus is on the overall energy spent by the network in order to service the route requests.

Second, the above statement that flooding is the method of choice if we want fastest delivery of messages from their source to their destination needs to be put into proper context. The statement is only definitely true if we isolate the routing layer. But what if we add collisions and interference to our network model? In order to truly optimize and understand the routing process, we need to consider the entire picture of wireless networks, in particular, we must include the link layer. The interaction between routing and MAC protocols has been studied via simulation in [19], with the conclusion that no single protocol from a layer was predominantly better; nor was any combination best over all parameters governing the mobility models and route requests. Again, as a first step, our focus in this dissertation rests on the network layer, since the lower layers, most prominently the MAC layer, are entire research areas in their own right. We fully realize that, ideally, the two need to be looked at together. We leave this open as a challenge for future research.

We tackle two questions in this part on mobility. The first asks what is feasible in extremely mobile networks. In this sense, Chapter 8 asks when and how flooding is possible under most severe conditions possibly caused by mobility. The results are partially based on the preliminary work found in [112] with major extensions and minor improvements. As a reaction to [112], the authors in [138] further distill the properties necessary to successfully flood the entire mobile network. The next question concerns the relative advantages of proactive versus reactive routing and is answered, to an extent,

in Chapter 9, the contents of which have not been published elsewhere. An indispensable element in the analysis is a careful definition of how to measure the amount of mobility and the efficiency of a routing algorithm. Only then is it possible to compare these two at first glance disparate mobile routing paradigms.

Chapter 8

High Mobility

At the start of our journey into the domain of mobile ad hoc networks, we ask what is possible at all if the network changes haphazardly and does so at a high rate. We see mobility on the graph level, that is, in terms of its effect on the links, and we initially abstract away any geometric (or social) realization. Thus, the first crucial step is to define our model properly. If mobility and communication links are completely arbitrary, then we cannot guarantee anything. We need to keep the model as general as possible to allow for maximum applicability and at the same time leave room for rigorous analysis. We proceed by incrementally adding those assumptions which are necessary in order for any algorithm to a priori have a chance to reliably propagate information from one part of the network to another.

Initially, we will consider flooding (a.k.a. broadcast) as a basic application involving the entire network. Flooding is a key back-up ingredient of virtually all mobile routing algorithms; and if flooding does not do the job, then what else would? We focus on a single message to be delivered instead of looking at flow or throughput considerations. Our goal is to have provably correct algorithms which at the same time require only finite amounts of energy. We will give several algorithms which fulfill these criteria given that the nodes have some minimal amount of knowledge about the network, such as its size.

8.1 Model

A *static network* is a graph $G = (V, E)$. A *mobile network* or *dynamic graph* is a sequence of static graphs: $G = G_{t_0}, G_{t_1}, G_{t_2}, \ldots$, with $t_{i+1} > t_i \geq 0$ and where $G_t = (V, E_t)$ is a graph. In particular, the set of nodes V remains fixed while the set of undirected links E_t can change arbitrarily. The only requirement is that G_t be connected for all t. Unless explicitly noted otherwise, we assume that the nodes have no knowledge of their current neighborhood. In fact, nodes are allowed to store only a poly-logarithmic

number of bits (that is, a constant number of identifiers) *in addition to* the actual message being flooded. Note that if we would allow nodes to learn the IDs of their neighbors, then we would also need to allow $\Omega(|V| \log |V|)$ storage overhead per node.

A network algorithm should have two basic properties: correctness and termination. A flooding algorithm is *correct* if the message from the starting node s reaches all other nodes in V for all dynamic graph instances and scheduling executions. *Termination* means that, for all dynamic graph instances and scheduling executions, there is a time such that no more message is sent. In accordance with this concept, we do not look at any kind of "periodic beaconing" algorithms where idle nodes send request messages to their neighbors. This would go against any idea of terminating since nodes would constantly send messages with new neighborhoods. We will scrutinize the definition of termination in Section 8.2.2.

We will consider both synchronous and asynchronous message passing models. First of all, note that we live in the wireless *local broadcast* world, that is, when a node sends a message, all of its (current) neighbors will hear it. Extending the *synchronous model* to mobile networks is straightforward. Communication proceeds in rounds. In one round, each node can send a message, then receive and process any potential messages from its current neighbors, and subsequently it may "move" to a new position. For ease of discussion, we will consider the rounds equivalent with time slots, that is, round i takes place at time $t = i$. In other words, the dynamic graph is the sequence $G = G_0, G_1, G_2, \ldots$ and if two nodes are neighbors in G_i, then they are guaranteed to be able to send each other one message during round i. The time complexity for a synchronous algorithm starting at time 0 is the number of rounds until completion.

Defining the asynchronous model is more intricate. We want a model with as few restrictions as necessary and as much freedom as possible. In accordance with that thought, the dynamics of the network must be such that they are still amenable to algorithmic analysis. Intuitively, we have to capture the idea that the graph cannot change indefinitely faster than it takes for messages to be delivered. A first restriction is that when a node moves into a new neighborhood, the new neighbors have to be able to exchange at least a single message. Otherwise, a node can continuously move from one neighborhood to the next and will never receive a message. To see that this needs to be strengthened further, consider the following execution in the neighborhood around a node v. There is a message on its way from v to a neighbor w which takes bounded but arbitrarily long time. Meanwhile, other nodes move into v's neighborhood, quickly exchange a message, and move out again. So v will possibly send arbitrarily many messages until w ever hears from v. Thus our asynchronous model should reflect the fact that message delivery time and graph connectivity duration must be correlated.

Formally, we specify our *asynchronous message passing model* the following way. Recall that in the standard distributed systems definition, the nodes do not have a sense of global time nor of message delivery order or speed. Local processing time is considered instantaneous. Algorithms are event driven (such as message receipt). A maximum time T on the delay of any message is assumed (though unbeknownst to the algorithm). The time complexity is the worst-case execution time over all possible graphs and message delivery schedules in units of T. For the dynamic case, motivated by the above discussion, we need to tighten the bound on the message delivery time, namely, by the frequency at which the graph changes. Let t_{uv} be the minimum time that nodes u and v are connected, and set $t_{uv} = \infty$ otherwise. The maximum time T that a message can be delayed is bounded by

$$T \leq \min_{u,v \in V} t_{uv} \tag{8.1}$$

which quantifies the concept of messages travelling at least as fast as the graph changes. Time complexity is again measured in terms of units of T, over all possible dynamic graph instances and scheduling executions in the worst case. Also, a node must be able to detect when a new node joins its neighborhood. Specifically, node v receives a "ΔN" event at time t if there exists a node w such that $w \in N_t(v)$ but $w \notin N_{t'}(v)$ for all t' in $t_0 \leq t' < t$ for some $t_0 < t$.

Indeed, message delivery and the graph connectivity requirements are the only constraints we put on the way that the network is allowed to evolve. In some sense, we can interpret this as allowing seemingly arbitrary local changes as long as the network retains some global structural properties. Furthermore, these two conditions can be generalized. We can parameterize Equation (8.1) to $T \leq \delta \cdot \min_{u,v \in V} t_{uv}$ in which case $\delta \leq 1$ must be known to the nodes. Similarly, instead of requiring that G_t be connected for all t, we can allow disconnection for a time period of at most $\varepsilon \geq 0$, in which case ε must also be known to the nodes. As this does not change the principal concepts of this chapter and would only serve to clutter the presentation, we will only present the case of $\varepsilon = 0$ and $\delta = 1$. An appropriate inclusion of δ and ε into the forthcoming algorithms will give the general case.

Notation and Terminology All of the algorithms are considered to be initiated by a starting node s at time $t = 0$. For ease of presentation, we will often refer to nodes as *idle*, denoted by $I_t \subset V$ and depending on the time t, if they have not yet received the message at time t. The rest of the nodes $F_t = V \setminus I_t$ are the *flooding* nodes. Note that $s \in F_t$ for all $t \geq 0$. We will also need the concept of *passive* nodes which have already received the message but consider themselves "done" with the protocol unless they receive new information. The remaining flooding nodes are *active*.

8.2 Knowledge of $|V|$

In this section we assume that the nodes know a polynomial upper bound $n \geq |V|$ on the size of the mobile network. This fairly powerful assumption will provide an easy case to allow us to become acquainted with the model. It will further open up the road to more complex algorithms when n is not known.

It is straightforward to build a terminating and correct *synchronous* algorithm in this case. Upon initial receipt of the message at a node v, it broadcasts the message for n rounds. Correctness is guaranteed by the fact that, due to graph connectivity, a new idle node is reached in each round. Therefore, the last idle node receives the message after at most $|V|-1 \leq n-1$ rounds. After another at most n rounds, all nodes stop sending messages. A slight improvement in complexity can be achieved if the nodes send along a counter h, initialized by s to $h = n - 1$ and decremented each round, and only broadcast the message for h instead of n times.

8.2.1 Algorithm

In the asynchronous case, we need to do some more work. Pseudo-code for nFLOOD is given in Algorithm 8.1. The basic idea is that a node v, after receipt of the message to be flooded, re-broadcasts the message to at most $2n^2$ "new neighbors." This leaves the message enough time to travel throughout the rest of the network.

```
1: receive msg
2: broadcast msg
3: k ← 0
4: afterwards, for each ΔN event:
5: if (k < 2n^2) then
6:    broadcast msg
7:    k ← k + 1
8: end if
```

Algorithm 8.1: nFLOOD

We now show the two essential properties of the asynchronous algorithm.

Lemma 8.1. *Algorithm nFLOOD terminates.*

Proof. The nodes in I_t do not send messages for all $t \geq 0$. Let t_f be the time after which F_{t_f} does not change anymore. Then for all nodes in F_{t_f}, it holds that either the dynamic graph keeps changing such that, eventually, $k = 2n^2$ at all $v \in F_{t_f}$, or the neighborhood of a node does not change after some time $t > t_f$, in which case that node does not send any more messages. □

Lemma 8.2. *Algorithm* nFLOOD *is correct.*

Proof. Let T be the maximum message delivery time of Equation (8.1). In a period of T time, any node v has less than $|V|\ \Delta N$ events, because every node can (re-)enter the neighborhood of v at most once during time T. Thus in a total time of $2T|V|$, v has less than $2|V|^2 \leq 2n^2$ events. Now fix a time t. We argue that at least one new idle node $u \in I_t$ is reached by time $t + 2T$. It is most instructive and intuitive to proceed by a straightforward case analysis.

Consider all pairs of the form $\{u_i, v_i\} \in E_t$ with $u_i \in I_t$ and $v_i \in F_t$. We know that $i \geq 1$ because G_t is connected; at least one idle node is bordering a flooding node at all times while there remain idle nodes. If all v_i have a local broadcast message out on the way at time t, then either a u_i receives it at the latest by $t + T$, in which case $u_i \in F_{t+T}$, or else all the u_i move away from their respective neighbors v_i. Since this happens to all such pairs, there will be a new pair $\{u', v'\} \in E_{t'}$ at time $t' < t + T$ such that a ΔN event is triggered at $v' \in F_t$, thus $u' \in F_{t''}$ for $t'' \leq t' + T < t + 2T$.

On the other hand, say that there is at least one pair $\{u, v\} \in E_t$, $v \in F_t$ and $u \in I_t$, which does not have a message traveling at time t because no relevant event happened at v. Let $t_1 < t$ be the time that first $v \in F_{t_1}$ and let t_2 be the time that u entered the neighborhood of v the last time before t. Since there is no event at v, both $t_1 < t$ and $t_2 < t$. If $t_1 \leq t_2$, then there was a ΔN event at node v at time t_2 and u is guaranteed to receive the message at time at most $t_2 + T < t + T$. If $t_1 > t_2$, then there is a message receipt event at node v at time t_1 and u receives the message at time at most $t_1 + T < t + T$ or else we are in the above case. Note that in both cases, we actually have a contradiction to either the assumption that there is no message on its way from v to u at time t or the assumption that $u \in I_t$.

Therefore, after 2 time units, a new idle node is reached and thus after a total of less than $2|V|$ time units, all nodes are flooded and the counter at each node is less than $2n^2$. □

To be very precise, we could set the termination condition in Line 5 of Algorithm 8.1 to $k \leq 2n(n-1)$ after closer inspection of the proof. As this would not significantly affect the performance and in order to not distract from the main idea, we have left it as $2n^2$ in the algorithm description.

The importance of this type of counter flooding algorithm is that even in asynchronous settings it guarantees the propagation of the message to at least n nodes for input value n in time proportional to n. This is a key property used to construct algorithms which do not have an accurate knowledge of $|V|$. With that in mind, we restate this formally.

Corollary 8.3. *For input values* $n \geq |V|$*, Algorithm* nFLOOD *reaches all nodes in* V *in time at most* $2|V|$*, that is,* $F_t = V$ *for* $t \geq 2|V|$.

8.2.2 Explicit Termination

Note that the above algorithm terminates in a very weak sense: There is a point in time such that no more messages will be sent. What if we want to know this point explicitly? If we consider synchronous settings, then explicit termination is straightforward. The starting node, for example, could just count the number of time steps; after $n \geq |V|$ rounds, all nodes have been reached. In this section, we will consider how to achieve a stronger notion of termination even in asynchronous models.

We propose three notions of a *termination requirement*, in the order of increasing strength:

1. There is a time T_m such that no more messages are sent.

2. There is a time T_c such that all nodes know correctness is achieved *and* there is a time $T_m \geq T_c$ such that no more messages are sent.

3. There is a time T_f such that all nodes know that they will never send another message.

The nFLOOD algorithm above gives an example of an algorithm terminating in the sense of Requirement 1. It is the standard definition we have adopted throughout this chapter. The motivation behind Requirement 2 is that the nodes can differentiate between a message propagation state and a state where storage of the message is no longer necessary.

We will show below that termination in the sense of Termination Requirement 3 is impossible. In other words, requiring that the nodes *know* when they will never send a message again (i.e., that all the other nodes are finished) can never be done, even in a stronger system model. Interestingly enough, if we knew that G would continue to change, then nFLOOD of Section 8.2 would indeed terminate in this strong sense since, eventually, the counters of all the nodes will reach $2n^2$. Likewise, if G would not change at all, then we have the static case in which the standard flooding algorithm of send once does the job. However, since the mobility we consider is unpredictable and the mobility can be somewhere "in between" static and constant motion, this problem becomes intractable.

Lemma 8.4. *In a correct asynchronous flooding algorithm, the nodes cannot reach a state in the sense of Termination Requirement 3 for all mobile networks and all message schedules.*

Proof. Assume that there exists an algorithm $\mathcal{A}$ which is correct and terminates in the sense of Term. Req. 3. Then there is a time $T_v \leq T_f$ and a node v such that v finishes its execution of $\mathcal{A}$, that is, it determines that it will never send a message again. Now a node's decision cannot depend solely on the number of neighborhood changes since an adversary can just

present $\mathcal{A}$ with a static graph. Therefore, a node's state must also depend on messages it receives and sends. In an asynchronous algorithm, messages can be delayed such that the nodes arranged in a line are all in different states of their respective execution of the algorithm. Therefore, at time T_v there exists another node w which has yet to receive several necessary messages or neighborhood changes causing it to terminate. Now set G_t such that $N_t(w) = \{v\}$ for all $t \geq T_v$. In order to terminate, w will wait indefinitely for another message which v never sends. This contradicts the assumption that w terminates in the sense of Term. Req. 3 at time T_f.

Observe that the above reasoning holds even if we allow the nodes to know their neighborhoods at all times and have unlimited storage capacity. □

A consequence is that synchronous systems are indeed more powerful than asynchronous ones in the message-passing model if the graph is allowed arbitrary dynamics. This is different from the static case, where there is a cost for synchronizers in time or message complexity [8], but it does not influence feasibility. Indeed, the assumption of a synchronous model implies that the nodes have some notion and measure of *time*. And timing is precisely what is critical in dynamic systems, hence it makes perfect sense that we can achieve more in a system model which has some timing capabilities, even if they are implicit.

Now we describe how to obtain an asynchronous algorithm which terminates in the slightly weaker sense of Termination Requirement 2. The idea is to run both the *dynamic* and the *static* parts "in parallel." Specifically, the dynamic part refers to the idea of executing nFLOOD and counting neighborhood changes; the static part refers to copying the simplest synchronizer algorithm for static graphs: If all (current) neighbors of node v are in round at least r, then v increments its round to $r + 1$. If either the counter of the dynamic part reaches $2n^2$ or the counter of the static part reaches $2n^3$, then the node has reached the passive state. Both static and dynamic parts are necessary for unpredictable graph mobility. This also implies the necessity of a stronger system model since only knowing neighborhood changes does not tell a node enough about its neighborhood if it remains static. Thus we assume that nodes know their neighborhood at all times.

The pseudo-code for such an explicitly terminating asynchronous algorithm TERMnFLOOD with knowledge of $n \geq |V|$ is given in Algorithm 8.2. Each node starts its round counter r with the round counter in the first message it receives, the starting node s with $r \leftarrow 0$, and the round information r is appended to the message.

Lemma 8.5. *Given a polynomial input $n \geq |V|$ and let $N_t(v)$ be known for all $v \in V$ and all $t \geq 0$, Algorithm* TERMnFLOOD *is correct and terminates in the sense of Termination Requirement 2 with T_c in $O(n^3)$.*

Thread main:

1: `receive` (msg,r')
2: $r \leftarrow r' + 1$
3: start nFLOOD (msg,r)
4: start synch thread
5: start check thread

Thread synch:

1: **while** $r \neq$ END **do**
2: **if** received r_i from all $v_i \in N_t(v)$ for some time t **then**
3: **if** $r \leq \min_i r_i$ **then**
4: $r \leftarrow \min_i r_i + 1$
5: **end if**
6: `send` (msg,r)
7: clear received from $N_t(v)$ status
8: **end if**
9: **end while**

Thread check:

1: for each event (message or ΔN) check:
2: **if** $k = 2n^2$ **then** $r \leftarrow$ END **end if**
3: **if** $r = 2n^3$ **then** $r \leftarrow$ END **end if**
4: **if** `receive` (END) **then** $r \leftarrow$ END **end if**
5: **if** $r =$ END **then**
6: delete msg
7: stop threads
8: nFLOOD (END)
9: **end if**

Algorithm 8.2: TERMnFLOOD

Proof. To prove termination, we need to show that at least one node v sets $r_v =$ END by time T_v in $O(T_c)$, since then the rest of the nodes will set $r_w =$ END by time $O(T_v + |V|)$. Note that if v ends because of Line 2 in the check thread, then correctness and termination easily follow from the previous discussion. If no node reaches $k = 2n^2$, then the round counter r will steadily be incremented until it reaches its stopping value $2n^3$. It remains to show that $r(t) < 2n^3$ at all nodes while $I_t \neq \emptyset$ in order to prove correctness.

In the time it takes to inform a new node of the message, we will bound the amount by which $r_{\max}$, the maximum counter in G_t at time t, grows. To that end, we make a few observations. First, border nodes ($B_t = \{v \in F_t \mid N_t(v) \cap I_t \neq \emptyset\}$) cannot increase their round counter since they will not receive information from all neighbors given that at least one is idle. Second, in a static graph G' (if we take a snapshot of G_t at time t) the discrepancy

for the static synchronizer is at most $|G'|$ for round decisions based on the information of the neighbors. Third, in time at most T, each node has at most $|V|$ ΔN events, as before.

Putting these together, we note that the discrepancy $r_{\max} - r_{\min} \leq |F_t|$ at time t since only nodes in F_t have a valid round counter. With each $\Delta N(v)$ (at most $2|V|$ in time T), $r_{\max}$ can grow by at most 1 (at most with $|F_t|$ nodes). Therefore, at the time $t' \leq t + 2T$ when a new idle node is reached, we have

$$r_{\max}(t') \leq (r_{\max}(t) + |F_t|) + 2 \cdot |V| \cdot |F_t| = r_{\max}(t) + (1 + 2|V|)|F_t|.$$

Thus, when all nodes are reached after at most $2|V|$ time units, we have

$$r_{\max} \leq (1 + 2|V|) \sum_{i=1}^{|V|-1} i \leq (1 + 2|V|)\frac{|V|^2}{2} < 2n^3$$

proving the lemma. □

Observe that the last part of TERMnFLOOD can be done differently which will not affect message or time complexity asymptotically but can save messages for the passive nodes in some cases. Such an approach would be to have the nodes, once in the END state, only send END messages *if requested by a neighbor*. In other words, upon node v newly entering the neighborhood of node w, w (who notices the change) sends a "round request" packet if w is still active. If v moves again before it replies to w, then w still has had a ΔN event, thus increasing k in nFLOOD. Therefore, w will also eventually set $r \leftarrow$ END. If w is already passive, then both v and w save a message since they do not automatically send a message to the new neighbor.

For the remainder of this chapter, we will return to our original definition of termination, since the subsequent algorithms do not have direct information about $|V|$. They do not know whether their current estimate of $|V|$ will need to be updated in the future, so that they risk initiating the termination phase too soon based on a too low value of n.

8.3 The Power of Identifiers

We have seen some of the possibilities of flooding algorithms in highly mobile networks if the nodes have a good estimate of the size of the network. Frequently, knowledge of the number of nodes might be an unrealistic assumption, in particular, if the network grows beyond any bounds imagined by the system engineer at the time of the original design. In this section, we want to take a look at what remains possible in mobile networks if we remove the $n \geq |V|$ assumption and replace it with other conditions.

8.3.1 IDs as a Substitute for $|V|$

A system requirement similar in power, but less global, is the assumption that nodes have (unique) identifiers with a logarithmic number of bits (i.e., the IDs are polynomial in $|V|$).

The key idea is to have the nodes learn and propagate an estimate of $|V|$. Once they have a suitable upper bound, we know that we can correctly reach all nodes. We will see below that it is sufficient to store the maximum seen identifier as this estimate. Initially, $\hat{n} \leftarrow v$, that is, the estimate for the number of nodes at node v is node v's own ID. When it receives a message with a higher estimate, it uses that new value. The details in pseudo-code for such an asynchronous algorithm are given in Algorithm 8.3 where $f(n) = cn$ for some value c. The function $f(n)$ influences how fast the flooding reaches all nodes. Below we will show that for $f(n) = 2n$, this will be optimal in time $O(|V|)$.

```
1: init: n̂ ← v
2: receive (msg,w)
3: if w > n̂ then
4:     n̂ ← w
5:     (re-)start nFlood (msg,n̂) with n = f(n̂)
6: end if
```

Algorithm 8.3: Asynchronous algorithm maxFlood at node v as a response to a message receipt event. The neighborhood change events are handled in the subroutine of nFlood.

Lemma 8.6. *Given that nodes have unique small identifiers, Algorithm* maxFlood *terminates and reaches all nodes in time $O(|V|)$ for $f(n) \geq 2n$.*

Proof. To see correctness, by the property of nFlood (Corollary 8.3), observe that the current maximum value $n_{\max}$ will reach $cn_{\max}$ many nodes: because lower-ID ones are reactivated if necessary, and higher-ID ones will be newly awakened (otherwise we did not consider the maximum ID). Therefore, we are guaranteed to reach a node with ID at least $cn_{\max}$, if it exists. Otherwise we have reached a node with ID at least the number of nodes and this will ensure the flooding of the entire graph. In other words, for $c \geq 2$, the maximum currently active ID $n_{\max}$ doubles in time $O(n_{\max})$. Thus we wait at most $\sum_{i=1}^{k-1} O(2^i) = O(2^k)$ time until $n_{\max} \geq |V|$ with $k = \log_2 |V|$. After that, it takes another at most $O(|V|)$ time units for nFlood to reach all nodes.

Algorithm maxFlood terminates because in time $O(|V|)$ the node with the overall maximum identifier $N_{\max}$ is reached, this information is propagated throughout the network by nFlood, and termination follows from termination of nFlood with $n = N_{\max}$. □

Note that message complexity is in $O(N_{\max}^2)$ per node, which can be considerably greater than $O(|V|^2)$ messages for the entire network. Below we will discuss a variant which has message complexity in $O(|V|^2)$, but requires greater message and storage size.

8.3.2 List Gathering

Apart from the nodes having unique identifiers, we can further relax the constraint that nodes are only allowed a small storage overhead. This could drastically decrease message complexity (if we disregard the size of a single message) as well as provide the opportunity to exactly enumerate and count the number of nodes in the network.

Another motivation for dropping the storage overhead is to consider another primitive of ad hoc and in particular sensor networks, namely, *information gathering*, the reverse of information dissemination, i.e., flooding. We can imagine that a starting node s wants to obtain the current data stored at all nodes. A naive approach is for s to flood a request message and each node responds with a separate message flooding its data, which will ensure that s will eventually (in time $O(n)$, $n = |V|$) receive the data from all the nodes. However, this uncoordinated second step can potentially burden the network with as many as $O(n \cdot \hat{n}^2)$ messages where $\hat{n} \geq |V|$ is the estimate the nodes have, such as the maximum identifier as in the previous section. The aim of this subsection will be to formulate and analyze a more efficient procedure to obtain the information stored at all nodes with only $O(n^2)$ messages. We further do not need the assumption that the nodes know the number of nodes, n, a priori.

Seen from this angle, technically, we are not violating the storage overhead constraint introduced in Section 8.1 if we consider the data list to be part of the message.

Our algorithm listFlood for list gathering works as follows. A node v needs to keep track of two lists, one for the data, d, and one for the IDs seen so far, l. It initializes $l \leftarrow \{v\}$ and sets $\hat{n} \leftarrow 1$. Upon receiving a message, it checks whether it needs to update its estimate of $\hat{n}$. A function of the current estimate $\hat{n}$ is used as the input to nFlood. Pseudo-code is given in Algorithm 8.4. Again, $f(n) = cn$ as in the previous subsection.

We restrict our reasoning to the development of l, the list of IDs, since this implies that all data from the nodes in l has been gathered. Let l_v denote the list at v. Consider $l_{\max} = \max_{v \in V} l_v$ and the following claim.

Lemma 8.7. *The currently largest list, l_{max}, doubles in size in time $O(|l_{max}|)$ for $f(n) \geq 4n$.*

Proof. Let T_1 be the time that the maximum list grows to $l := l_{\max}$ and let $n := \hat{n}_{max} = |l|$. Recall that by Corollary 8.3, the list l will be known to $f(n)$ nodes in time $O(f(n))$ by the property of nFlood with input $f(n)$.

```
1: init: d ← {v : data}, l ← {v}, n̂ ← 1
2: receive (d′,l′)
3: d ← d′ ∪ d
4: l ← l′ ∪ l
5: if |l| > n̂ then
6:    n̂ ← |l|
7:    (re-)start nFlood (d,l) with n = f(n̂)
8: end if
```

Algorithm 8.4: Asynchronous algorithm listFlood at node v as a response to a message receipt event. The neighborhood change events are handled in the subroutine of nFlood.

At most $2(n-1)$ time units are "wasted" on informing the other nodes $v \in l$, such that the list does not grow. Assuming $f(n) = cn$, there remain $m \geq f(n) - n + 1 \geq (c-1)n + 1$ other nodes $u_1, \ldots, u_m$ ($u_i \notin l$ for $1 \leq i \leq m$) which will receive a message containing $l' \supseteq l$ before any of the $v \in l$ become passive due to nFlood stopping. Because $u_i \notin l$, such a message will (re-)activate u_i so that each of the u_i's will itself spread the information to at least $f(n+1)$ nodes. Let the u_i be numbered in the order in which they received the (first) message containing (a superset of) l. Then a list containing at least $l \cup \{u_i\}$ will spread to at least $x_i \geq m - i + 1$ other nodes (because u_i is active for enough time to spread its information to at least another $m - i + 1$ others before the first node in l becomes passive).

To summarize, after $2f(n)$ time units, when the first node in l might potentially become passive, there is a set L of $n + m$ nodes such that $l_v \supseteq l$ for $v \in L$. Among those there are x_i nodes w such that $l_w \supseteq l \cup \{u_i\}$.

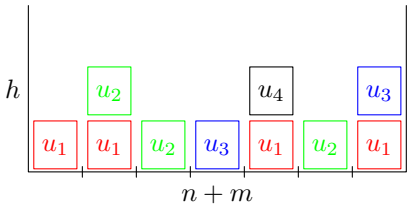

Figure 8.1: Arrangement of the information for the proof of Lemma 8.7.

In order to find the minimum growth of $l_{\max}$, consider the arrangement of the information (i.e., the list) of each of those $n + m$ nodes. To that end, we can consider a container of width $n + m$ and height h, see Figure 8.1 for an illustration. A unit of width represents a node in L and the height represents the elements in the list at each node *without* those in l. Now we must fill this container with all x_i, $1 \leq i \leq m$, unit elements which represent the

new information of the u_i's. Note that because of mobility, these blocks do not need to be contiguous, all we know is that there are x_i non-overlapping pieces of each u_i. The maximum height will represent the growth of $l_{\max}$ after $2f(n)$ time units. Now the *minimal maximum height* is given when the container is filled evenly, so that

$$\begin{aligned} h &\geq \frac{\sum_{i=1}^{m} x_i}{n+m} \geq \frac{m(m+1)/2}{n+m} \\ &\geq \frac{m(m+1)/2}{m \cdot c/(c-1)} \\ &= \frac{1}{2}\frac{c-1}{c}(m+1) \end{aligned} \tag{8.2}$$

where the first line follows from $x_i \geq m - i + 1$, and the second line from $m \geq (c-1)n + 1$, that is,

$$n + m \leq \frac{m-1}{c-1} + m = m\frac{c}{c-1} - \frac{1}{c-1} \leq \frac{mc}{c-1}.$$

Observe that for $c \geq 4$ (or, more precisely, for $c \geq 2 + \sqrt{3}$), Equation (8.2) becomes $h \geq n$ again using that $m \geq (c-1)n + 1$. In other words, there is a node in L which has h new elements in addition to $l_{\max}$. Thus, at time $T_2 \leq T_1 + 2Tf(|l_{\max}|)$, the maximum list has at least twice as many elements as the maximum list at time T_1. □

The remainder of the correctness and termination proof is the same as that for maxFlood in Lemma 8.6.

Corollary 8.8. *Given that nodes have unique small identifiers and that nodes are allowed storage overhead of $\Omega(|V| \log |V|)$, Algorithm* listFlood *terminates and reaches all nodes in time $O(|V|)$ for $f(n) \geq 4n$.*

Proof. Each time the maximum list doubles from n to $2n$, we pay at most $2Tcn$ time for $f(n) = cn$. Thus, we wait at most $2Tc\sum_{i=1}^{k-1} 2^i \leq 2Tc \cdot 2^k$ until $l_{\max} = |V|$. Here, $k = \log_2 n$, thus $l_{\max}$ is V after $2Tc|V|$ time. Once a single node has an estimate $\hat{n} \geq |V|$, then it takes at most another $2T|V|$ time for nFlood to reach all nodes. □

As discussed above, this approach can also be utilized for fast flooding instead of the maximum-ID-based scheme from Section 8.3.1, if enough storage space is available. The advantage of this scheme is that, in the end, $\hat{n} = |V|$ instead of the possibly much higher maximum ID. This saves in messages and also makes it possible to count the nodes.

8.4 Impossibility

It is time we turn our attention to the limitations which mobility imposes. We have shown that flooding is possible with some assumptions, but what if we drop those? How crucial are they? In this section, we will look at how mobility can make it impossible to successfully flood a network in a scenario where it is almost trivial to do so were the graph static. In fact, this holds even when the network is seen as nodes in the plane, moving arbitrarily with some maximum velocity. Specifically, we will provide evidence that, without further knowledge, flooding a light-weight ad hoc mobile network is impossible for node speeds greater than a quarter of the communication radius per time step.

8.4.1 Model and Notation

While the forthcoming statements hold for arbitrary graphs, we will look at unit disk graphs in particular, giving stronger lower bounds. As we have seen in the first part, UDGs might not be the perfect model of reality, but they can give us a more geometric picture when talking about mobile networks. Specifically, we want to quantify the speeds with which nodes are allowed to move, acting as a guide in choosing the appropriate parameters for any kind of simulation or testing of routing or other algorithms. The rest of the model is the same as before, described in Section 8.1: The dynamic graph $G_t = (V, E_t)$ is connected and now G_t is unit disk graph for all times $t \geq 0$. Let $r : V \times \mathbb{R}^+ \to \mathbb{R}^2$ be the realizations of G over time: $r(v, t)$ is the position of node v in the Euclidean plane at time t. For convenience of notation, we write $d_t(u, v) = \|r(u, t) - r(v, t)\|$ for two nodes $u, v \in V$ at time t. Further, denote by ν the maximum speed with which nodes move such that

$$\frac{\|r(v,t) - r(v,t')\|}{|t - t'|} \leq \nu$$

holds for all $v \in V$ and all $t \neq t'$, $t, t' \geq 0$. The nodes are not aware of their neighborhoods and are only allowed poly-logarithmic storage overhead. We stress that $|V|$ is not known to the nodes nor are they allowed the use of node identifiers, since these cases were covered in the previous sections.

To keep things simple and in order to understand where the main problems arise, we will only consider a synchronous model in this section. As before, this means that there are communication rounds, each of which consists of the sending of one message, then the receipt and processing of any potential messages from neighbors and subsequently the nodes may move to a new position. Additionally, the starting time (that is, starting node s "receives" the message at time 0 and starts sending at time $t = 1$) of the algorithm is known to all nodes. This is merely a conceptual simplification

and can be implemented via a counter sent along with the message. This counter would be incremented by each flooding node in each round.

8.4.2 A First Approach

In order to explore the limits of mobility, we want to know how fast nodes may move in order for the flooding information to propagate faster than the nodes. Recall that we seek an algorithm which is correct and terminates. With that in mind, we will first consider a simple terminating algorithm $\mathcal{A}_1$, with pseudo-code given in Algorithm 8.5 on page 123, and explore its correctness properties. Later on, we will generalize it and argue why no algorithm can both terminate and be correct above a certain node speed.

The idea of $\mathcal{A}_1$ is that if a node v first receives a message in round t, then there are at least that many nodes plus v itself. Thus, time (or, in a sense, the number of hops) gives the nodes an indication on the size of the network. This estimate $\hat{n}$ is now used similarly to the nFLOOD algorithm. A node stops sending after $f(\hat{n})$ rounds with its latest update on $\hat{n}$. For termination, it is crucial that $\hat{n}$ is not incremented arbitrarily (seen over the entire network), and for correctness we want $f(\hat{n})$ to be large enough such that the nodes are active until the remaining idle nodes are reached.

```
Thread main:
1: receive (msg,n')  at time t
2: n̂ ← t + 1
3: start update thread
4: for i = 1 to f(n̂) do
5:    send (msg,n̂), once per round
6: end for
Thread update:
1: receive (msg,n')
2: if n' > n̂ then
3:    n̂ ← n'
4:    i ← 1 in main
5: end if
```

Algorithm 8.5: $\mathcal{A}_1$

Lemma 8.9. *Algorithm $\mathcal{A}_1$ will terminate.*

Proof. Since $V_t = V$ at all times t, there will be a node $v \in V$ which is the last to receive the message for the first time at some time t. In other words, the set of flooding nodes is $F_{t'} = F_t$ for all times $t' \geq t$ because either the algorithm is correct and then $F_t = V$, or there will be some nodes which are never reached by the algorithm. Then $\hat{n}_v = t + 1 = \max_{v \in V} \hat{n}$. Since $\hat{n}$ is

never increased beyond an n' which is received in the update thread and v was the last to set its $\hat{n}_v$ according to the time, $\hat{n}_u \leq t+1 \ \forall u \in F_t$ and for all times $t' \geq t$. After the information of $\hat{n}_v = t+1$ propagates to at most all nodes in F_t in finite time, the update thread will become ineffective. Then each of the nodes $u \in F_t$ executes the for loop in the main thread for another at most $f(\hat{n}_u)$ rounds. □

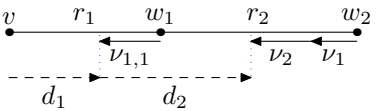

Figure 8.2: Message progress after two rounds for one possible dynamic unit disk graph instance.

We now turn our attention to correctness. To gain some intuition, consider a quantity δ which is the average distance the *message* (or: information) can move in the plane per step. To that end, consider the progress of a message at a node v over a path $p = vw_1w_2$ of two hops, such as depicted in Figure 8.2. Let r be the realization, set $r_1 = \|r(v,t) - r(w_1,t)\|$ and $r_2 = \|r(w_1,t) - r(w_2,t)\|$. Then $r_1 + r_2 > 1$, otherwise the message could go from v to w_2 directly. After one round at time $t+1$, the information has travelled at least $d_1 = r_1 - \nu_{1,1}$ distance along the path from v to w_1. In the next round, $d_2 = r_2 + (\nu_{1,1} - \nu_1) - \nu_2$. Thus the total minimum distance $2\delta = d_1 + d_2$ covered by the message away from v in two rounds is

$$2\delta = r_1 + r_2 - (\nu_1 + \nu_2) > 1 - 2\nu, \tag{8.3}$$

making the average progress per step $\delta > 1/2 - \nu$. Note that if $\nu > 1/2$, then $\delta \leq 0$ is possible.

Lemma 8.10. *If $\nu > 1/2$ per time unit, then Algorithm $\mathcal{A}_1$ will* not *reach all nodes for any choice of f.*

Proof. We will construct a counter example such that, after some time, no new information will reach the starting node s in which case it stops sending messages. Intuitively, the setup is as follows. There are two phases. In the first phase, any node updating the value of $\hat{n}$ will immediately move away from s, keeping the new information from reaching s. In the second phase, there must be a large enough buffer between s and the nodes still currently active, until the node with the highest information on $\hat{n}$ becomes passive and intercepts the information progress towards s.

Set $x = \min\{1, \nu\} > 1/2$. The nodes of the graph consist of s, nodes u_i, $1 \leq i \leq m$, and nodes v_j, $1 \leq j \leq n - m - 1$. We will determine the value for m later.

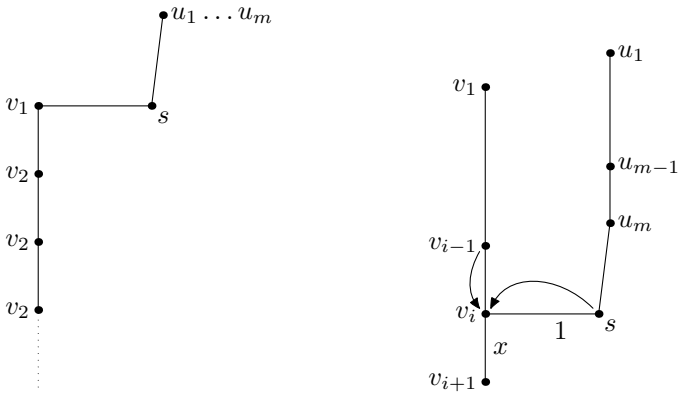

Figure 8.3: Snapshots of the embeddings of the unit disk graph in the counter example in the proof of Lemma 8.10. On the left, at time $t = 1$, on the right, at time $t = i$ (both at the beginning of their respective rounds).

When s starts sending messages at time $t = 1$, let the embedding r be given by

$$\begin{aligned} r(s,1) &= (0,0) \\ r(u_i,1) &= (\varepsilon, 1-\varepsilon) && 1 \le i \le m \\ r(v_j,1) &= (-1,(1-j)\cdot x) && 1 \le j \le n-m-1 \end{aligned}$$

for some small $\varepsilon > 0$. The resulting graph is a clique of m points u_j clustered at one point, joined to a path $sv_1v_2 \dots v_{n-m-1}$ at node s, so G_0 is connected. The realization $r(\cdot, 1)$ is shown in Figure 8.3 on the left. When s broadcasts the first message, all the u_i as well as v_1 will receive it in round $t = 1$.

For the next rounds until $t = t_1$ (we specify t_1 below), all nodes v_j move upward by x and all nodes u_i, $i \le t_1$, move upward by a fraction of x:

$$\begin{aligned} r(u_i, t+1) &= r(u_i,t) + (0, \tfrac{t_1-i}{t_1-1}x) \\ r(v_j, t+1) &= r(v_j,t) + (0,x) = (-1,(t-j)x). \end{aligned}$$

The realization in the beginning of round i is depicted in Figure 8.3 on the right. Since $x > 1/2$, v_{i+1} will not receive a message in round i, only v_i hears from s and v_{i-1}. After t_1 rounds, $d_{t_1}(u_j, u_{j+1}) = x$ for $1 \le j \le t_1$. In other words, we have a path of the u_i on the right, and a path of the v_j on the left, in both of which the nodes are spread x apart. The horizontal distance between the two paths is $1 + \varepsilon > 1$. The setup is such that $v_i \in I_t$ for all $1 \le t < i$, and $v_i \in F_i$, thus v_i sets $\hat{n}_{v_i} = i + 1$ initially.

The key to this first phase of the counter example is that s never receives any information back from the v_j because they always move just out of range.

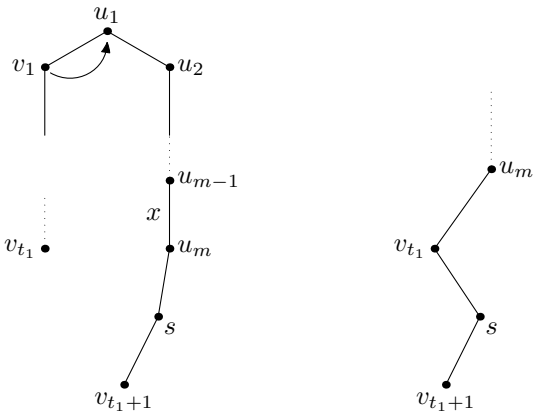

Figure 8.4: On the left, $t = t_1$ (at the end of the round), and on the right, $t = t_2 + 1$ of the counter example.

Therefore, only the u_i propagate their estimates to s, thus $\hat{n}_s = 2$ for times $t \geq 2$. In the above construction, we let t_1 be the time such that s sends its last message in round t_1, specifically, $t_1 = f(2) + 2$ (as s sends two messages before it receives $\hat{n}_s = 2$). The left side of Figure 8.4 depicts the network at the end of round t_1 when the nodes have moved. At this point, v_{t_1} severs its connection with the remaining v_j, $j > t_1$, and a new connection between u_1 and v_1 is established at the top. Now s will not send any more messages to v_{t_1+1}, and the closest message (in terms of hops) to s with $\hat{n} > 2$ is located at the top with u_1. Note that $d_{t_1+1}(v_1, u_2) > 1$.

Let the message get to u_2 at time $t_1 + 2$. From there on, the chain of the u_i's keeps moving upward by $x > 1/2$:

$$r(u_i, t + 1) = r(u_i, t) + (0, x)$$

for $t \geq t_1 + 2$ and $2 \leq i \leq t - 4$. Then the message progress on the path from $r(u_2, t_1)$ to $r(s, t_1)$ is $\delta \leq 0$. In other words, we can wait an arbitrary amount of time and s will not receive the message with $\hat{n} > 2$ with enough buffer nodes u_i.

Since v_{t_1} is the last node which has received the message up to this point, no other node will have newer information about $\hat{n}$ which would cause v_{t_1} to reset its loop counter to 1. Let t_2 be the round after which node v_{t_1} stops, that is, $t_2 = t_1 + f(t_1 + 1)$. At the end of round t_2, the remaining nodes u_j which were connected to s move slightly upwards while v_{t_1} moves to the right serving as the only link between s and the u_j for times $t > t_2$. Now s will never receive a message with $\hat{n} > 2$ (since v_{t_1} is the only non-idle node connected to s and it has stopped sending and will not be woken up

again). Therefore, node v_{t_1+1}, being only connected to s, will never receive the message if we set $m \geq t_2$. □

Lemma 8.11. *If $\nu > 1/4$ per time unit, then Algorithm $\mathcal{A}_1$ will* not *reach all nodes for any choice of f.*

Proof. We will extend the above proof of Lemma 8.10 to include node speeds between one half and one quarter. To that end, we must reconstruct the two phases, first for $1 \leq t \leq t_1$ until s becomes passive and second for $t_1 < t \leq t_2$ until v_{t_1} becomes passive.

The idea for the first phase is that if both s and v_j move in opposite directions with speed $\nu > 1/4$, then the chain of v_j's such that $d_t(v_j, v_{j+2}) > 1$ can be constructed as before. Node s and the lower half of the u_i's now move downward (s by ν, the other by the appropriate fraction) and the upper half of the u_i's move upward, resulting in $d_{t_1}(u_i, u_{i+1}) = 1/2 + \varepsilon$. This is also the key for the second phase, since for $\nu > 1/4$, $\delta < \nu$ in the above chain of u_i's when each u_i moves opposite the message flow after receiving it. In other words, the new information cannot catch s if it is also moving away and we apply the same idea as before by letting v_{t_1} intercept the message at s at time t_2 since $t_2 = f(t_1+1)+t_1$ and we can let the message be forwarded to an arbitrary number of u_i nodes given that $\delta > \nu$. □

Theorem 8.12. *Algorithm $\mathcal{A}_1$ terminates but is not correct if $\nu > 1/4$ for any choice of f.*

In fact, we can reconstruct phase two for any speed $\varepsilon > 0$ by having multiple strands of the u_i, where one replaces the other as the message moves down. It might be necessary to extend the length of the initial chain of u_i's far enough such that one node can cover the distance greater than 1 in order to replace the layer. This comes at a cost of getting s to know more information in order to increase t_1, but is irrelevant since the layer replacement can be done indefinitely for large enough m. It remains as an open problem to determine the exact speed bound such that phase one can be reconstructed.

8.4.3 Generalizations

Algorithm $\mathcal{A}_1$ was a naive first approach to finding a correct and terminating flooding algorithm. We have seen that for node speeds above a certain threshold, the algorithm will terminate but not be correct. As simple as this algorithm is, we will now argue why it might be impossible to do any better.

First we discuss a generalization of $\mathcal{A}_1$. In the hopes of fixing the problem from the counter example in Lemma 8.10, instead of one estimate $\hat{n}$ depending on time, we have k such cascading variables, giving Algorithm $\mathcal{A}_k$. Specifically, instead of $\hat{n}$ now the variables $t_1, \ldots, t_k$ are appended to

the message. A node v sets t_1 the first time it hears the message. If it receives a message with $t'_1 > t_1$, then it updates t_1 and sets t_2 to the time it received this update. Otherwise, if $t'_2 > t_2$, it updates t_2 and sets t_3 and so on. See Algorithm 8.6 for the detailed pseudo-code. However, by choosing the number of "buffer nodes" from the proof above appropriately, this generalization of Algorithm $\mathcal{A}_1$ will not succeed either. We can simply repeat the construction for the k nodes with the greatest t_i. Above, at time t_1 the graph around v_{t_1} is constructed analogously to s, and the pattern is repeated for each v_{t_i} with respective stopping time t_i, $1 \leq i \leq k$. Since k has to be determined a priori, it is independent of the number of nodes and for m large enough, $\mathcal{A}_k$ will terminate but will not be correct.

Thread main:

1: `receive` (msg, $t'_1, t'_2, \ldots, t'_k$) at time t
2: $t_i \leftarrow t + 1$ for each $i = 1 \ldots k$
3: start update thread
4: **for** $i = 1$ to $f(t_k)$ **do**
5: `send` (msg, $t_1, t_2, \ldots, t_k$), once per round
6: **end for**

Thread update:

1: `receive` (msg, $t'_1, t'_2, \ldots, t'_k$) at time t
2: **if** $t'_1 > t_1$ **then**
3: $t_1 \leftarrow t'_1$
4: $t_i \leftarrow t$ for each $i = 2 \ldots k$
5: $i \leftarrow 1$ in main
6: **else if** $t'_2 > t_2$ **then**
7: $t_2 \leftarrow t'_2$
8: $t_i \leftarrow t$ for each $i = 3 \ldots k$
9: $i \leftarrow 1$ in main
10: ...
11: **else if** $t'_k > t_k$ **then**
12: $t_k \leftarrow t'_k$
13: $i \leftarrow 1$ in main
14: **end if**

Algorithm 8.6: $\mathcal{A}_k$

To further enhance $\mathcal{A}_1$, consider the information at a node's disposal. The available information is (a) the (number of) messages received or not received in a round and (b) the time (synchronous) or hops (asynchronous). The problem with (a) is the termination requirement because a node cannot differentiate between an idle neighbor and a neighbor which has become passive. Thus, if it assumes that a non-sending new neighbor has not heard the message, then we can create a mobile instance such that the algorithm does

not terminate. We will discuss the counting of message below. Concerning (b), this case is covered by $\mathcal{A}_1$ since we can choose the stopping function f freely, yet it can only depend on t, the time the message has been active; so the problem here is the correctness requirement as we saw above.

The remaining open question concerns the counting of messages. A node cannot measure its current degree since it is not aware of its neighborhood, but it can estimate the degree by counting the number of messages received in one round. The degree can help establish a bound on the number of nodes in the network. In the counter example of the proof of Lemma 8.10, node u_m will hear a message from all m of its neighbors in round $t = 2$, so it could set $\hat{n} \geq m + 1$. This would invalidate the argument in the proof since now s could remain awake sufficiently long until it knows that there are more nodes in the network. To counter such an algorithm, we need to refine our example. Instead of having all the u_i clustered at a single point initially, they can be spread out such that the subgraph of the u_i has diameter D and maximum degree Δ. Since we have a unit disk graph, this implies that the graph is stretched out over an area at most D by D wide and, dividing it into cells of width $1/\sqrt{2}$, we immediately get that $\Delta \geq 2m/D^2$. Or, in other words, if all nodes keep sending, then the highest degree measured (by counting messages per round) along with the longest hop path give an upper bound on the number of nodes m in the subgraph. It remains to be seen whether an instance of a dynamic unit disk graph can be constructed such that not all nodes are sending at all times and the longest diameter cannot be approximated well either. Perhaps this will open another chasm between synchronous and asynchronous mobile algorithms.

It is interesting to note that if we do not consider UDGs, then we can find a graph such that $\Delta = O(1)$ and $D = \Theta(\log m)$, such as a complete binary tree rooted at s. In this case, a correct estimate of the number of nodes is exponential in D, which violates the storage requirement in other graph instances where $D = \Theta(n)$. Unfortunately, in the general graph case, the intuitive notion of node speed is inapplicable.

To summarize, we end up with the following picture. If we consider unit disk graphs with allowed node speeds greater than one quarter unit per time step, then it appears that a correct and terminating flooding algorithm is hard to come by. An essential ingredient will be determining the maximum degree by counting of messages received in one round, in combination with estimating the diameter of the network via time (hops). While there is slight hope that this might be possible in synchronous settings, we would wager that this is not the case for the asynchronous model with nodes unaware of identifiers and neighborhoods. Moving away from the geometric unit disk model, the above arguments, albeit without rigorous proof, strongly suggest that the nodes cannot collect enough information in order for any terminating algorithm to also be correct.

If the above is true, an implication of the previous discussion is that we can apply this in a broader sense: Mobile algorithms' performance (and feasibility) will depend on the speed of mobility. In other words, we should look at mobile algorithms and tune their parameters as a function of the maximum or at least average node speed.

8.5 Routing

Since flooding without identifiers appears difficult if not impossible, one can imagine that a related problem, that of routing, is solvable within our constraints. That is, as long as one is sure that the destination exists and will respond, otherwise the problem degenerates to flooding. We will use the destination ID synonymously with a node being able to check locally whether it is the destination or not.

In this section, we will take a closer look at the routing problem of delivering a message from a source node s to a designated destination t. Note that correctness now means that we only need to reach t, not all nodes. As in previous algorithms, the idea is to estimate some finite upper bound $\hat{n} \geq |V|$ and then use the ideas from nFLOOD to guarantee correctness and termination. As we have seen, the difficulty lies in estimating the number of nodes without the use of IDs. In the routing case, this hurdle is overcome by the fact that the destination node t can acknowledge the receipt of the message and initiate a termination phase which will make use of a message counter.

The algorithm works as follows. Every node v stores a counter $\hat{n}$. Initially, every node is in INIT mode, that is, idle. Once a node has seen the message, it can be in one of two states: Either it assumes that the destination has not been reached and the message needs to be propagated at every opportunity, or else it *knows* that the destination has been reached and it needs to be careful about letting the other nodes know that they can stop as well but so as to not endanger the termination requirement. The counter $\hat{n}$ is incremented in the first state (FLOOD mode) and then used as an upper bound on $|V|$ in the second state (TERM mode). Once $\hat{n} \geq |V|$, the nFLOOD algorithm will guarantee us termination and correctness.

The details are given in Algorithm 8.7. The source s starts the algorithm by broadcasting FLOOD (message, 1).

Lemma 8.13. *Algorithm 8.7 is a correct routing algorithm.*

Proof. Correctness is easily seen from the algorithm as nodes will continue sending (Line 6 in INIT and Lines 1-3 in FLOOD mode) until the destination has received the message (Line 3 in INIT) and will announce this with the acknowledgement TERM (Line 2 in TERM). □

Input: none
Output: receipt of message, $\Delta N \neq \emptyset$

mode = INIT

1: `receive` msg with n'
2: $\hat{n} \leftarrow n' + 1$
3: **if** (msg = FLOOD $\wedge$ v = dest) $\vee$ (msg = TERM) **then**
4: mode $\leftarrow$ TERM
5: **else**
6: `broadcast` FLOOD (message, $\hat{n}$)
7: mode $\leftarrow$ FLOOD
8: **end if**

---

mode = FLOOD

1: for each ΔN event:
2: $\hat{n} \leftarrow \hat{n} + 1$
3: `broadcast` FLOOD (message, $\hat{n}$)
4: for each `receive` msg with n' event:
5: **if** msg = FLOOD $\wedge$ $n' > \hat{n}$ **then**
6: $\hat{n} \leftarrow n'$
7: `broadcast` FLOOD (message, $\hat{n}$)
8: **else if** msg = TERM **then**
9: $\hat{n} \leftarrow \max(\hat{n}, n')$
10: mode $\leftarrow$ TERM
11: **end if**

---

mode = TERM

1: process/delete message locally
2: nFLOOD ($\hat{n}$), msg = TERM ($\hat{n}$)
3: `receive` msg with n'
4: **if** $n' > \hat{n}$ **then**
5: $\hat{n} \leftarrow n'$
6: nFLOOD ($\hat{n}$), msg = TERM ($\hat{n}$)
7: **end if**

Algorithm 8.7: Asynchronous Mobile Routing Algorithm at node v.

By inspection and the discussion of proof of Lemma 8.2, we also know that the routing algorithm is efficient in terms of reaching the destination quickly.

Corollary 8.14. *Algorithm 8.7 achieves correctness after at most* $2n$ *time units.*

We now turn to termination.

Lemma 8.15. *If* G *keeps changing such that* $\Delta N(F) \neq \emptyset$ *for the set of nodes* $F \subset V$ *with mode(v) =* FLOOD $\forall v \in F$*, then any* $x \in F$ *will eventually enter* TERM *mode.*

Proof. Consider the terminated and flooding nodes $T_t = \{v \in V \mid \text{mode}_t(v) = \text{TERM}\}$ and $F_t = \{v \in V \mid \text{mode}_t(v) = \text{FLOOD}\}$, respectively, depending on the time t. Initially, T_0 will contain the destination node (set $t = 0$ when the message has reached its destination).

If $\Delta N(v) \neq \emptyset$ for a node $v \in F_t$, this will increment $\hat{n}(v)$. If $\hat{n}(v) > \hat{n}(w)$ for $w \in N(v)$, then $\hat{n}(w) \leftarrow \hat{n}(v)$ after at most 2 time units. Thus, the highest estimate $\hat{n}$ will propagate throughout a connected component of flooding nodes.

Now assume there is a time t where all nodes $x \in T_t$ have stopped sending upon $\Delta N(x) \neq \emptyset$ (i.e., the call to nFLOOD in Lines 2 or 6 of TERM mode has finished) and $F_{t+2T} \neq \emptyset$. Then $\max_{x \in T_t} \hat{n}(x) < |V|$, otherwise the call to nFLOOD with the TERM message would spread to all nodes in V. The eventual graph changes will now continually cause an increase of $\hat{n}(v)$ at some $v \in F_t$. Then at some time $\tilde{t} > t$, there will be $n' := \hat{n}_{\tilde{t}}(v) \geq |V|$ at some $v \in F_{\tilde{t}}$. It will take at most $|F_{\tilde{t}}| < |V|$ hops for the message containing n' to reach a node $x \in T_{\tilde{t}}$ at time less than $\tilde{t} + 2T|V|$. Then $n' > \hat{n}(x)$, prompting a call to nFLOOD of message TERM (n') at node x with parameter at least $|V|$. Therefore, before time $\tilde{t} + 4T|V|$, all nodes will have received a TERM message. □

Lemma 8.16. *Algorithm 8.7 terminates.*

Proof. We only have to prove that if the graph changes, the algorithm will stop sending messages eventually. Otherwise, if there are no more ΔN events after a certain time, termination of the routing algorithm follows from the termination of nFLOOD. Therefore, assume subsequently that G keeps changing. If the edge changes only involve the neighborhoods of nodes which have already TERM-inated, then Algorithm 8.7 will eventually stop sending messages because of a finite maximum $\hat{n}$ among the terminated nodes. Otherwise, by Lemma 8.15, we know that every node which has entered the FLOOD mode must eventually receive a TERM message. At that point, the highest estimate $\hat{N}$ is fixed since only FLOOD-ing nodes cause an increment of the counter (see Line 2 of FLOOD mode). Then $\hat{N}$ will be some finite value and all other

nodes will update their $\hat{n} \leftarrow \hat{N}$ and the TERM messages will be propagated at most that many times to already terminated nodes (if the graph continues to be mobile). In other words, once all the nodes are TERM-inated, then they execute the nFLOOD algorithm which we know will end for a finite $\hat{N}$. □

The final item to be shown concerns the storage requirement of mobile algorithms.

Lemma 8.17. *Algorithm 8.7 needs only $O(\log |V|)$ bits of storage and header size.*

Proof. In order to ensure small local storage and header size, we need to bound the largest estimate $\hat{n}$ by a polynomial in $|V|$. To that end, we examine the counter values of the above termination argument in more detail. We will keep track of the largest counter $\hat{n}$ in N. Set $n := |V|$.

Since the destination is reached after at most $2Tn$ time units, $N \leq 2n^2$ when the destination enters the TERM mode by the previously observed fact that a node can receive at most n ΔN events in time T.

Thereafter, we have seen in Lemma 8.15 that once any node has an estimate $\hat{n} = n$ due to continual increases in its neighborhood, the rest of the algorithm will take its course to ensure that the nodes terminate. In the worst case, this information needs to travel less than n hops to a TERM node, so $N < n + 2 \cdot n \cdot n$ in the meantime. In any case, once the call to nFLOOD in Line 6 (TERM mode) is issued, it takes less than another $2n$ time units for all nodes to TERM-inate, which implies less than $2n^2$ more neighborhood changes at a node. In the end, $N = O(n^2)$ by the time that no more nodes are in FLOOD mode and $\hat{n}$ will not be incremented anymore. □

8.6 Discussion

This chapter served as a preliminary investigation into the feasibility of an analytical treatment of routing in highly mobile networks. After presenting a hodgepodge of algorithms, the most important result is that flooding is indeed possible in a strong asynchronous model when the number of nodes or unique identifiers are given. Routing to an existing destination is possible even without this additional information. Conversely, finding a correct and terminating flooding algorithm which does not rely on node IDs in any way and which observes polynomial storage constraints seems to be an exercise in futility. If, however, we restrict the nodes' movement to the Euclidean two-dimensional plane with specified maximum speed, then it remains to be seen whether indeed even a synchronous flooding algorithm exists.

Chapter 9

Moderate Mobility

We pushed the amount of mobility to an extreme in the last chapter. If instead the state of the network changes at moderate speeds, then the focus of attention shifts from feasibility to efficiency. Common knowledge says that reactive algorithms are better when mobility is "high" and proactive ones when it is "low." In this chapter, we want to *quantify* the efficiency of different mobile routing paradigms and to state precisely, at least for some scenarios, what exactly high and low mean. An important point to keep in mind in this discussion is that considering merely a mobile network is pointless without an application. It just means that the network is changing, but what is of interest is the *effect* of those changes. We must consider mobility and routing together to make any meaningful statements about either. This is central when defining what an efficient algorithm is. It is also what sets this chapter apart from previous works in the area of comparing different routing algorithms under varying mobility conditions as outlined in Section 7.1.2.

9.1 Model of Mobility

In the preceding chapter we looked at mobile events (link changes) as interleaved with ongoing route or flooding requests. In this chapter, we want to slow down the dynamics of the network in order to examine the aspect of routing efficiency rather than feasibility. Therefore, to ease analysis, we adopt a model where graph changes and route requests are separated in time. We distinguish between two phases: either the occurrence of a *mobile event* where the graph can change and the nodes may react to it, or the servicing of a *route request* in its entirety. We will discuss the different types of possible mobile events below; for now we can think of it as a single link changing its state. One route request means that a single source initiates the routing of a data packet to its destination; the route request is finished when no more

packets pertaining to that route request are travelling in the network. Since proactive routing algorithms respond to mobile events and the nodes a priori do not know how many such graph changes will occur, we must also allow for an algorithm to pass along messages after each mobile event; these messages must be completed before the next mobile event occurs. In other words, a mobile event is coupled with the response (or update) messages from a proactive algorithm. Of course, the reality will look quite different, where network change events are interleaved with update messages and route requests. For the sake of analytical study, the above model introduces two sharp boundaries. One boundary separates mobile issues from route requests in order to ensure that routing always succeeds. The next boundary separates routing messages from update messages. The reason for letting the updates to complete before the next network change instead of gathering all updates after several mobile events is that we want to count the effect of each link change since the nodes do not know how many there will be. Otherwise, if the nodes know when the network has reached a quiescent state, then it could perform a single network update flooding and all subsequent route requests are handled efficiently. See also Figure 9.1.

We obtain the following picture: In one *session*, there are m mobile events, after each of which the nodes are allowed to send messages to recompute their routes, and subsequently r route requests are processed in their entirety. The dynamics of the network is measured in terms of the *mobility ratio* $m : r$ of mobile events to route requests. Since we are interested in *moderately mobile* networks, as opposed to highly mobile in the last chapter, we note that m is in $O(r)$. Also note that $m \geq 1$.

The key idea to note is that the ratio of graph changes to route requests is important, not their absolute values. Even if the graph changes in its entirety such that there are $O(|V|^2)$ link changes, this can be mitigated by a large number of subsequent route requests which, in a fully proactive algorithm, would be serviced optimally.

Although we only considered link changes in the previous chapter, a mobile event can be specified in any number of ways. We can consider exclusively link changes or exclusively node additions and deletions, or, more naturally, a combination of both. Furthermore, nodes can react to link (or node) joins differently than to failures. One instance of such an approach can be found in the family of link reversal algorithms [54]. Link failures are treated proactively and new paths are computed, while not much effort is spent on a new link coming up. In this case, we can imagine splitting up the mobile events into the number of link joins versus the number of link leaves per routing session. Initially, when talking about a mobile event, we only consider the case of an edge joining or leaving, thus the number of nodes in the network, n, again remains constant.

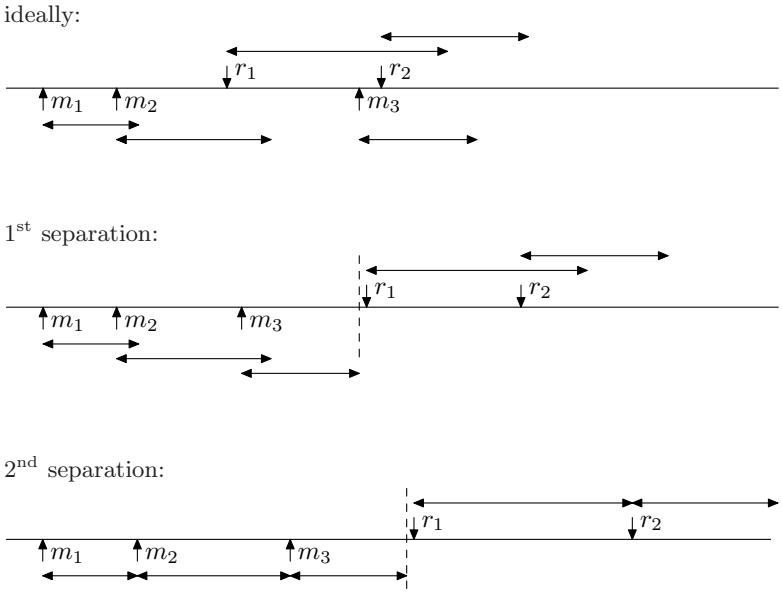

Figure 9.1: The three figures schematically show the timeline of mobility and route request events and how long the corresponding update and routing messages are in the network for three different levels of abstraction. Ideally, we would like to look at route requests and mobility-caused update messages completely interleaved as in the top drawing. Since we do not look at the success rate of routing but rather at the efficiency, in a first step, we separate the mobility events from the route events, as shown in the middle figure. Lastly, we want to count, in the worst case, the effect of each single event (thus ignoring any "cumulation bonus" from processing several mobile events at once), so all the messages are completely separated in time, as indicated by the schematics on the bottom. This lowest drawing is what we have in mind when we refer to a session of $m:r = 3:2$.

9.2 Measures of Efficiency

The focus of this chapter being routing efficiency, it is crucial to define what costs are associated with a route request. In (mostly) static networks, a primary aim of routing schemes is to minimize the *stretch*, the ratio of the chosen route over the shortest path. In other words, it minimizes the *time* of delivery. If we were to apply the same standard to mobile networks, then using flooding[1] to search for the destination every single time would find the optimal path for each route request. However, few would agree that this qualifies as "efficient" in the case of energy-constrained wireless ad hoc networks which exhibit only moderate dynamics. This shows that we need to include all the messages pertaining to a route request, not just the messages on the path from the source to the destination. On the other extreme, a protocol such as distance vector tries to use the shortest path, but it does so at the cost of recomputing all paths after every relevant link change. So the messages for updating and maintaining a route should also count towards the cost of a route request. Equivalently, for mobile ad hoc networks, it is the message complexity of a route request which matters, not so much the time complexity.

Formally, we measure the efficiency of a routing algorithm in terms of its amortized *message complexity per route request*. Consider two consecutive route requests q_1 and q_2. The number of messages credited towards q_2 is all the messages after the completion of q_1 until the completion of q_2. If the graph changes between q_1 and q_2, then any potential messages are counted towards the cost of q_2 since they occur after the completion of q_1. In other words, a proactive algorithm, responding to m link changes, will add these costs to the next route request. As with the idea of the stretch, we need to compare the cost of the algorithm to the optimal cost, which is the sum of the shortest paths of the route requests. Here, instead of comparing the routes individually, we amortize over an entire session of route requests. We consider a wireless broadcast medium, that is, a node locally broadcasting a message to all its neighbors is counted as one message.

Definition 9.1. *Consider a session $m:r$ of m mobility events and r route requests. An algorithm sends c_m messages in response to the m network changes and needs $c_r(i)$ messages to service the ith route request. Then the* competitive ratio *ρ per route request of the algorithm is*

$$\rho := \frac{c_m + \sum_{i=1}^{r} c_r(i)}{\sum_{i=1}^{r} \text{OPT}(i)} \tag{9.1}$$

where $\text{OPT}(i)$ is the shortest path between the source and destination of the ith route request.

[1] recall that we ignore any lower layer issues such as collisions

Our primary aim in this chapter is to obtain a rough idea of how the mobility ratio influences the choice of proactive versus reactive types of algorithms. For ease of analysis, we need to make a few simplifying assumptions. We will only consider the case where all nodes route to the same destination. That is, while the source is not known before each route request, the destination always is. Likewise, the size of the messages as well as the storage space at each node is unlimited, focusing only on short route constructions and not on how and where they are stored. For instance, the difference between AODV and DSR is that one is point-to-point, the other stores the entire route in the packet; this is an implementation detail (albeit an important one) which we wish to disregard in our analysis. We also exclude the entire line of research on compact routing, as little has been done to understand the costs of updating the relatively complex data structures in the face of dynamic graphs. We only consider topology-based algorithms, that is, where only the graph structure $G = (V, E)$ is known, excluding all types of algorithms knowing the positions of the nodes in (two or three-dimensional) Euclidean space.

An alternative between completely flooding the network (either pro- or reactively) and not sending any messages at all is a *local flooding* up to k hops. Recall that $N(v, k)$ denotes the neighborhood of up to k hops around node v. Then we define the *local density* D of a graph $G = (V, E)$ to be

$$D = \max_{v \in V, 1 \leq k \leq \mathrm{diam}} \frac{|N(v, k)|}{k}. \tag{9.2}$$

The local density, albeit with a scaling factor of 2, has been widely used in the context of the graph bandwidth problem (see for instance [49]). In our case, it gives the worst-case perspective on the cost of flooding with a limited time-to-live parameter: Starting at a node v, flooding up to distance d reaches $|N(v, d)| \leq 2Dd$ nodes. Observe that $1 \leq D \leq n$ (considering only connected graphs).

9.3 Proactive versus Reactive

The terms *reactive* and *proactive* describe when an algorithm becomes active. A proactive algorithm responds to mobility events and computes new routes at this point, usually computing an optimal route which it can subsequently use for a route request. Since the nodes do *not know* the mobility ratio of the dynamic network, a proactive algorithm will generally respond to every mobility event since it might be the only one for a while. A reactive algorithm ignores mobility and only responds to a route request, at which point it can either use an old route or decide to search for a new one. Thus, in a reactive algorithm, we necessarily have $c_m = 0$. What has been studied in the literature as *hybrid* protocols we will term *partially proactive* and examine this case further in Section 9.3.2.

9.3.1 Classic Approaches

Classic Reactive: Flooding A classic example of a reactive routing algorithm is flooding. The idea is to search for a new route with each and every route request. A simple protocol of adaptive flooding searches the area around the source with exponentially increasing radius until it finds the destination. The exponential growth ensures that the cost is dominated by the distance to the source. Since the number of nodes also grows with distance, we have the total cost for each such search i

$$c_r(i) = \Theta(D \cdot \text{OPT}(i))$$

giving $\rho = \Theta(D)$ per route request independent of the $m : r$ session since $c_m = 0$. The issue of when $\rho = \Theta(D)$ or $o(D)$ hinges on the answers to the questions at the end of Section 9.4, where we will discuss the graph-theoretic details. All further mention of the term "flooding" will refer to the above described adaptive protocol.

Classic Proactive A classic proactive routing protocol would inform the network about each change, essentially flooding the information "link e has changed" to all nodes, incurring costs of $c_m \leq m \cdot n$, $n = |V|$, for the m mobility events. The details of $c_m = m \cdot \Theta(n)$ is the subject of the discussion in Section 9.4. Any subsequent route request can now be serviced optimally with cost $\text{OPT}(i)$ per route request. Thus, in an $m : r$ session, the total cost for the r route requests in such a proactive protocol is $c \leq m \cdot n + \sum_i \text{OPT}(i)$, or

$$\rho \leq \frac{m \cdot n + \sum_{i=1}^{r} \text{OPT}(i)}{\sum_{i=1}^{r} \text{OPT}(i)} \leq \frac{m \cdot n}{r} + 1$$

which gives $\rho = O\big(\frac{m}{r} \cdot n\big)$ since $\text{OPT}(i) \geq 1$ and thus $\sum_{i=1}^{r} \text{OPT}(i) \geq r$.

Comparison The analysis of the above two classic routing paradigms puts us in a position to compare them and determine how much mobility is allowed in order for such a fully proactive algorithm to be better than a flooding approach. In other words, *roughly* comparing the two competitive ratios above, we want

$$D \geq \frac{m \cdot n}{r} \tag{9.3}$$

that is, the number of route requests should be at least $r = \Omega\big(m \cdot \frac{n}{D}\big)$, or, alternatively, the mobile events should be at most $m = O\big(r \cdot \frac{D}{n}\big)$. In other words, for linearly growth-bounded graphs where $D = O(1)$, the number of route requests (on average) before the graph changes again should be in the order of the number of nodes such that proactive measures are effective. Only in very dense graphs with $D = \Theta(n)$ is the overhead of the proactive and flooding algorithms comparable when the network changes about as frequently as route requests are issued.

What we learn from the above is that the question of when proactive routing (such as done in the Internet, for instance) is better than adaptive flooding depends mainly on the ratio $m : r$, but also on the local density of the graph as compared to the number of nodes, that is, D/n.

9.3.2 Partially Proactive

A proactive algorithm might not necessarily want to inform all relevant nodes. For instance, the hybrid Zone Routing Protocol (ZRP) [63] is proactive on a small scale and reactive with respect to the rest of the network. Abstractly, we define a k-proactive algorithm as follows: Whenever a link changes, inform affected nodes up to (hop) distance k. When servicing a route request, a node chooses a (potentially outdated) cached route of length d_{old}. If the path fails, the node can perform adaptive flooding to find a new path, subsequently using this new path of optimal length. (A node might want to store several old paths; if any of them eventually work, we are in the above case where it uses an old route of length d_{old}, plus the additional costs of searching the failed paths. Thus, storing multiple paths does not help in the worst case.)

The proactive costs will be $c_m = m \cdot O(Dk)$, or more specifically, $c_m = m \cdot O(\min(Dk, n))$. The routing cost for each request i is either optimal ($c_r(i) = \text{OPT}(i)$) because the node learned of the change in the network from the proactive case or the route was unaffected, or the optimal distance is greater than k ($\text{OPT}(i) > k$). Conversely, if $\text{OPT}(i) > k$, then either the old route still exists and $c_{\text{succ}}(i) = d_{\text{old}}(i) = O(n)$; or the old route is broken, then $c_{\text{fail}}(i) = d_{\text{old}}(i) + O(D \cdot \text{OPT}(i))$. Let us consider the favorable case (for the algorithm) that all r route requests are issued by the same source (though this is not known a priori). Then only $c_{\text{fail}}(1) = d_{\text{old}} + O(D \cdot \text{OPT})$ and $c_{\text{fail}}(i) = \text{OPT}$ for $1 < i \leq r$ where OPT is the shortest path length.

We obtain the following competitive ratios for the case that all routes concern the same sender-destination pair in an $m : r$ mobile network. For $\text{OPT} \leq k$, the costs are dominated by c_m since the routing costs are optimal, so

$$\rho_{\text{opt}} = \frac{m \cdot O(Dk)}{r \cdot \text{OPT}} + 1 = O\Big(\frac{m}{r} Dk\Big).$$

For $\text{OPT} > k$, $c_m/(r \cdot \text{OPT}) < c_m/(rk)$, so

$$\rho_{\text{succ}} = O\Big(\frac{c_m}{rk} + \frac{rn}{rk}\Big) = O\Big(\frac{m}{r} D + \frac{n}{k}\Big)$$

$$\rho_{\text{fail}} = O\left(\frac{c_m}{rk} + \frac{n + D \cdot \text{OPT} + (r-1)\text{OPT}}{r\text{OPT}}\right) = O\left(\frac{m+1}{r} D + \frac{1}{r}\frac{n}{k}\right).$$

This gives a worst-case overall competitive ratio of

$$\rho = O(\rho_{\text{opt}} + \rho_{\text{succ}} + \rho_{\text{fail}}) = O\Big(\frac{m}{r} \cdot Dk + \frac{n}{k}\Big). \tag{9.4}$$

Now for a fixed ratio $m:r$ we can solve for the optimum k, giving

$$k = \sqrt{\frac{n}{D} \cdot \frac{r}{m}}$$

which ensures $Dk \leq n$ if we consider mobility ratios $m/r \geq D/n$. (Below that we already know that fully proactive routing outperforms flooding.) Plugging this value of k into Eq. (9.4), we get

$$\rho = O\left(\sqrt{\frac{m}{r} \cdot Dn}\right).$$

To compare with reactive algorithms, we ask again, when is

$$D \geq \sqrt{\frac{m}{r} \cdot Dn}, \tag{9.5}$$

or, for which mobility ratios does this become better than flooding with a competitive ratio of roughly D. Notice that this is essentially Eq. (9.3). In other words, a k-proactive algorithm does not outperform flooding unless the mobility-to-routing ratio is as low as it is for a fully proactive algorithm, even in the favorable case of the same source issuing subsequent route requests.

9.3.3 Other Reactive Approaches

Investment of Path Costs The definition of reactive also allows for a little more room to play with. It is not necessary for a reactive algorithm to always search for a new route. For instance, a reactive algorithm could proceed as follows: If the sender node s has cached a route p to the destination t, then it tries that route. Further it invests the cost of the used route to find a new one; it does so for each route request since the algorithm does not know the mobility ratio. If s fails to reach t, then it proceeds by incremental flooding as above. As before, let us assume that, in a session of $m:r$ mobile events and subsequent route requests, the sender is always s. This way the investment has a chance to pay off.

Let $d = |p|$ be the length of the stored route and OPT the optimum route length. If p is broken, then s fails with costs less than d and the search for the new route costs $O(D \cdot \text{OPT})$, giving a total of $c_{\text{fail}}(1) = O(d + D \cdot \text{OPT})$. If s reaches t via p, then the actual route costs d and the search for a new route stops at distance either d (in which case d was optimal) or proportional to OPT (in which case p was outdated and too long). Thus, $c_{\text{succ}}(1) = d + O(D \cdot \text{OPT})$. All other requests are dominated by the search for a new path up to radius OPT, thus $c_r(i) = O(D \cdot \text{OPT})$ for $1 < i \leq r$. Therefore, $c_r = O(d + r \cdot D \cdot \text{OPT})$, giving

$$\rho = O\left(\frac{d}{r \cdot \text{OPT}} + D\right) = O\left(\frac{n}{r} + D\right)$$

using that OPT ≥ 1 and $d \leq n$ (assuming it had a valid loop-free path). Note that the above expression is at least D regardless of the mobility, which means that it is not better than flooding which does not use any old routes, even in the case of the same source sending to the same destination.

Reuse of Old Routes This leads us to question the efficiency of an algorithm which uses an old path whenever possible, without the searching above. First, note that since we have a reactive algorithm, it ignores all network change events. This means that it cannot know when to search for a new path and when not. This implies that a node's decision to search given that it has a valid old path is independent of the existence of a new path. So a reactive algorithm which always uses an old path (assuming it exists) will have the ratio of the old versus new path as its stretch, which can be as bad as in the order of nodes (i.e., $\rho = \Theta(n)$) which is worse than the classic proactive algorithm for any $m \leq r$.

Periodic Searching There can also be hybrid approaches where a node searches every I steps for a new path, in the other cases assuming it had a valid old path. In a best-possible scenario, consider again always the same source (or: the same source after searching for a new path) and $r > I$, otherwise at least every other session of route requests has overhead of $\Theta(n)$ in the worst case. The average number of route requests after a mobility event and before the next search (counting I route requests, regardless of mobility) is in the order of $\Theta(I)$. Then the costs for a routing session (i.e., r consecutive route requests) is

$$c_r(I) = \Theta(I \cdot d) + \lceil r/I \rceil O(D \cdot \text{OPT}) + (r - I)\text{OPT}$$

where the first term stems from using an old route of length d after some mobility events, the second term are the search costs, and the remaining route requests are serviced optimally with costs OPT. This gives a competitive ratio of

$$\rho(I) = O\left(\frac{I \cdot n}{r} + \frac{\lceil r/I \rceil D}{r}\right) = O\left(n\frac{I}{r} + \frac{D}{I}\right)$$

again with $d \leq n$ being the old path and OPT ≥ 1. Solving for the optimum $\rho(I)$ gives $I^* = \sqrt{r \cdot D/n}$, thus

$$\rho(I^*) = \Theta\left(\sqrt{D \cdot \frac{n}{r}}\right).$$

Note that the condition $I < r$ is satisfied since $D < n$. In order for this to be at least as good as classic adaptive flooding, we need $r = \Omega\left(\frac{n}{D}\right)$ (compare to the extra m factor in the proactive case). For sparse graphs with low D, this implies a large number of consecutive route requests for such an algorithm

to be superior to flooding, even in the favorable case of the same source issuing the route requests after a path discovery. Further observe that I^* depends on r, which actually invalidates the above discussion. If the nodes know the "period of stability" and the pattern of route requests and mobility events occur regularly (as we have assumed in the above discussion), then the nodes can determine the timing of the mobile events and in fact design an optimal algorithm. Without knowledge of r, we can set $I' = \sqrt{n/D}$ to obtain $\rho_{I'} = O\Big(\sqrt{Dn}(1 + 1/r)\Big) = O\Big(\sqrt{Dn}\Big)$, yielding no advantage over flooding at all.

9.3.4 Perspective

A priori, we know that the more mobile a network, the more reactive a routing algorithm should be since the costs of proactively responding to mobile events will eventually outweigh the benefit of storing short routes. There is a natural tradeoff between how much messages are invested to keep information up to date and the quality of the used routes when they are needed. In Section 9.3.2 we have seen that, asymptotically and in the worst case, we have nothing to gain from using a partially proactive algorithm over a fully, classical proactive approach. The same holds for the other reactive approaches as discussed in Section 9.3.3. Thus, a posteriori, it seems that we can abandon any hopes of finding a family of algorithms which has a parameter to adjust smoothly for the mobility of the network. Indeed, the picture looks binary, as shown schematically in Figure 9.2: If the mobility ratio is greater than about D/n, then only a fully reactive algorithm such as adaptive flooding makes sense from the worst-case, asymptotic perspective. Otherwise, use a fully proactive algorithm.

9.4 The Fine Print: Discussion and Future Work

Aside from shedding some light on the pro- versus reactive routing algorithms debate, this chapter, more than anything, has uncovered numerous further important questions. First of all, on the positive side, we have shown, as in the case for highly mobile networks, also for moderately mobile networks an analytic treatment of the costs of routing algorithms is possible. The first step in this direction was to properly define the costs of a routing algorithm in the face of mobility. After that, what we have given is a bird's eye view of the number of messages per route request for several routing paradigms. This asymptotic and worst-case analysis has opened up the path for a more detailed theory of routing algorithms. It is in the worst-case view that we see the most room for future work.

The two major assumptions of the previous analysis requiring scrutiny are the worst-case overheads in both the pro- and reactive case. We can

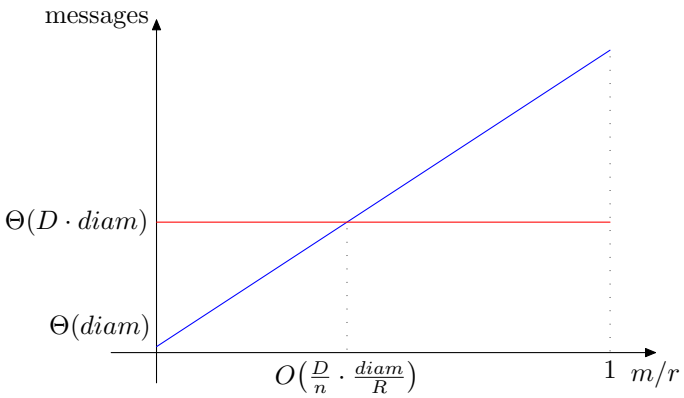

Figure 9.2: Comparison of the messages per route request for classic pro- and reactive algorithms in terms of m/r, increasing to the right. We assume that the average path length is in the order of the diameter *diam* of the network and we run as many sessions (with the same $m : r$ ratio) such that a total of R route requests are serviced. The (red) horizontal line is the adaptive flooding algorithm. The other (blue) is the classic proactive algorithm, (potentially) informing all n nodes of a link change, thus the equation for the line is in the order of $\Theta(n \cdot (m/r) \cdot R + diam)$.

succinctly state the problem as one of determining how *paths* are affected by mobility. It is clear that after a large number of link changes, almost all paths in the graph can change. But if the mobility events in a session are few, then a central combinatorial question in this case is to determine how many nodes' paths are significantly affected by only a certain number of link changes in the graph.

The assumption affecting the cost of reactive algorithms is whether flooding up to hop distance d can really always cost as much as $\Theta(D \cdot d)$? For instance, a star topology has $D = n$, but the center node only needs to send out one message to reach all nodes. If we extend the leaves of the star by short paths, then indeed $\Theta(n)$ messages are necessary to flood. If we imagine that the destination is connected by a single link to only one outermost star node in each session of two mobile events, then a reactive algorithm must indeed go through half of all paths (on average) on the star to determine where the new path is, see also Figure 9.3. On the contrary, a partially proactive algorithm which sends a message from the new link to the center of the star would eliminate this problem of a reactive algorithm. By the same token, we can ask ourselves if it is really fair to consider graphs where only in a few places the local density is as high as D while the rest of the nodes are

connected sparsely. One option might be to look at uniform-density graphs.

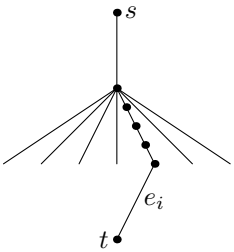

Figure 9.3: Topology for fully reactive routing algorithms such that each route request costs $\Theta(n)$ messages. Edge e_i changes its location with every mobile event.

In the proactive case, we assumed that each single link change can affect a large part of the network, that is, in the order of n nodes. In a simple view, where nodes want to remain up-to-date with the exact distance to the destination, this is the case as shown by the example of a ring network with a different link missing each mobile event (strictly speaking, every other mobile event). However, a node which previously had a short path to the destination of length $d_1 < n/2$ can still "try" the old route first, then use the new route of length $d_2 \geq n/2$, incurring costs of less than $2d_1 + d_2 < 3d_2$, that is, the stretch of the new route is at most 3 without sending any proactive messages. We can extend this example by allowing a set of logarithmically many links (with exponentially increasing many nodes in between) to create "shortcuts" to the destination such that no constant stretch of the old over the new route is possible without proactively informing many nodes. What this example also shows is that we need to treat link deletions and additions separately. If links only go away (in a session), then trying a previously shortest route first will not hurt, since any new route will be longer. The stretch is affected by new routes. This is where splitting up the mobile events into link joins and leaves will prove helpful.

A further extension to the mobility model is to include weighted edge lengths. The primary questions then is how do we define adaptive searches instance? Further, we need to include a weight change as a mobile event.

The above discussion, in particular Section 9.3.3, shows that the quality of being reactive does not alone determine how well an algorithm performs with respect to mobility when compared to a proactive approach. This motivates us to identify another algorithm parameter which we term *investment*. The intent is to model how much effort an algorithm puts into searching for new paths. In this sense, both of the classical protocols are the same because flooding searches for a new path each route request and because the proactive

protocol, by reacting to each change in the network, already has every possible new path ready for any route request. Thus, they both invest the same amount into finding short paths, the difference is only when: proactive upon mobility, and reactive upon routing. The crucial question here is whether this could lead to a qualitatively new way of classifying routing protocols.

Chapter 10

Conclusion

Geometry and mobility, two seemingly disparate concepts which are united by the important role they play in wireless ad hoc networks. In the first part of this dissertation we have demonstrated that it is indeed possible to give algorithms with provable guarantees for the intricate problem of unit disk graph embedding. Together with other results in the area, this promotes research on network localization from purely heuristic methods to theoretical analysis. Two main further goals loom ahead: One ideal is to have a simple, combinatorial, and most importantly *local* algorithm which still provides guarantees on the quality of the computed embedding. The other and perhaps equally idealistic goal is to find a constant approximation algorithm.

As for mobility models, some prior work exists measuring and analyzing the effects of a mobility model on the network graph. In particular, recent emphasis has been on link and path lifetimes. We have complemented this geometric and random point of view with a general, worst-case perspective. This is only the beginning. What remains is to obtain more extensive simulation results as well as to continue the analytical study, in particular the investigation of path stability. It seems to us that there is considerable potential in studying the path lifetime and its associated change on the lengths of shortest paths between nodes from a combinatorial point of view as well as by means of more statistical analysis. The latter in particular implies that more information about use cases of mobile ad hoc networks is vitally important. Moreover, the focus should not rest on "measuring the mobility" alone to describe and classify mobility models; we have shown the importance of looking at network changes *together* with route requests as a meaningful way to express the complexity and overhead of mobile ad hoc routing protocols.

We emphasize that the second part of the dissertation is by far more preliminary than the preceding discussion of unit disk graph embeddings. This is due largely to the great contrast in the objectives of the two parts. For UDG embedding, the problem is clear cut, all that is left is to find a

good algorithm. For mobile routing, theory is still in its infant stage, and we need to explore the space spanned by the different options with which we can model mobility. Despite these differences, or perhaps even because of them, we hope to have shed at least a tiny sparkle of light into the vast darkness which still clouds wireless ad hoc networking research.

Bibliography

[1] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *Proc. of the* 16^{th} *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004.

[2] S. Agarwal, S. Krishnamurthy, R. Katz, and S. Dao. Distributed power control in ad-hoc wireless networks. In *Proc. of the* 12^{th} *IEEE Intl. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2001.

[3] K. Albrecht, R. Arnold, M. Gähwiler, and R. Wattenhofer. Aggregating information in peer-to-peer system for improved join and leave. In *Proc. of the* 4^{th} *IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2004.

[4] G. Allard, P. Jacquet, and B. Mans. Routing in extremely mobile networks. In *Proc. of the* 4^{th} *Mediterranean Ad Hoc Networking Workshop (MedHocNet)*, 2005.

[5] O. Angel, I. Benjamini, E. Ofek, and U. Wieder. Routing complexity of faulty networks. In *Proc. of the* 24^{th} *ACM Symposium on the Principles of Distributed Computing (PODC)*, 2005.

[6] J. Aspnes, T. Eren, D. Goldenberg, A. S. Morse, W. Whiteley, Y. R. Yang, B. Anderson, and P. Belhumeur. A theory of network localization. *IEEE Transactions on Mobile Computing*, to appear, 2006.

[7] J. Aspnes, D. Goldenberg, and Y. R. Yang. On the computational complexity of sensor network localization. In *Proc. of the* 1^{st} *Intl. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, 2004.

[8] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4), 1985.

[9] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *Proc. of the* 42^{nd} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

[10] B. Awerbuch, D. Holmer, R. Kleinberg, and H. Rubens. Provably competitive adaptive routing. In *Proc. of the* 24^{th} *IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[11] B. Awerbuch and R. D. Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proc. of the* 36^{th} *ACM Symposium on Theory of Computing (STOC)*, 2004.

[12] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. of the* 26^{th} *ACM Symposium on Theory of Computing (STOC)*, 1994.

[13] M. Bădoiu, J. Chuzhoy, P. Indyk, and A. Sidiropoulos. Low-distortion embeddings of general metrics into the line. In *Proc. of the* 37^{th} *Symposium on Theory of Computing (STOC)*, 2005.

[14] M. Bădoiu, E. Demaine, M. T. Hajiaghayi, and P. Indyk. Low-dimensional embedding with extra information. In *Proc. of the* 20^{th} *Symposium on Computational Geometry (SoCG)*, 2004.

[15] M. Bădoiu, K. Dhamdhere, A. Gupta, Y. Rabinovich, H. Räcke, R. Ravi, and A. Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *Proc. of the* 16^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[16] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proc. of the* 19^{th} *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2000.

[17] F. Bai, N. Sadagopan, and A. Helmy. The IMPORTANT framework for analyzing the Impact of Mobility on Performance Of RouTing protocols for Adhoc NeTworks. *Ad Hoc Networks*, 1(4):383–403, 2003.

[18] F. Bai, N. Sadagopan, B. Krishnamachari, and A. Helmy. Modeling path duration distributions in MANETs and their impact on reactive routing protocols. *IEEE Journal on Selected Areas in Communications*, 22(7):1357–1373, 2004.

[19] C. Barrett, M. Drozda, A. Marathe, and M. V. Marathe. Characterizing the interaction between routing and mac protocols in ad-hoc networks. In *Proc. of the* 3^{rd} *ACM Intl. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2002.

[20] L. Barriére, P. Fraigniaud, and L. Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *Proc. of the* 5^{th} *Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 2001.

[21] A. Basu, J. Gao, J. Mitchell, and G. Sabhnani. Distributed localization by noisy distance and angle information. In *Proc. of the* 7^{th} *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2006.

[22] C. Bettstetter. Smooth is better than sharp: A random mobility model for simulation of wireless networks. In *Proc. of the* 4^{th} *ACM Intl. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2001.

[23] J. Beutel. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Location Management in Wireless Sensor Networks. CRC Press, 2004.

[24] R. Bischoff and R. Wattenhofer. Analyzing connectivity-based, multi-hop ad-hoc positioning. In *Proc. of the* 2^{nd} *IEEE Intl. Conference on Pervasive Computing and Communications (PerCom)*, 2004.

[25] P. Biswas, T.-C. Liang, K.-C. Toh, and Y. Ye. An SDP based approach for anchor-free 3d graph realization. Preprint, 2005.

[26] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proc. of the* 3^{rd} *Intl. Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.

[27] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[28] J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[29] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry: Theory and Applications*, 9(1-2):3–24, 1998.

[30] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of the* 4^{th} *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 1998.

[31] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *Proc. of the* 6^{th} *ACM Intl. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2005.

[32] C. Busch, S. Surapaneni, and S. Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proc. of the* 15^{th} *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2003.

[33] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[34] I. D. Chakeres and J. P. Macker. Mobile ad hoc networking and the IETF. *ACM SIGMOBILE Mobile Computing and Communications Review*, 10(1):58–60, 2006.

[35] H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, and A. Slivkins. Metric embeddings with relaxed guarantees. In *Proc. of the* 46^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

[36] C.-C. Chiang. *Wireless Network Multicasting*. PhD thesis, The University of California at Los Angeles (UCLA), 1998.

[37] J. D. Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 4(3):197–229, 1997.

[38] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of the* 10^{th} *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2004.

[39] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996.

[40] V. A. Davies. Evaluating mobility models within an ad hoc network. Master's thesis, Colorado School of Mines, 2000.

[41] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[42] L. Doherty, K. Pister, and L. E. Ghaoui. Convex position estimation in wireless sensor networks. In *Proc. of the* 20^{th} *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2001.

[43] S. Dolev, L. Lahiani, S. Gilbert, N. Lynch, and T. Nolte. Brief announcement: Virtual stationary automata for mobile networks. In *Proc. of the* 24^{th} *ACM Symposium on Principles of Distributed Computing (PODC)*, 2005.

[44] J. Dunagan and S. Vempala. On euclidean embeddings and bandwidth minimization. In *Proc. of the* 5^{th} *Workshop on Randomization and Approximation (RANDOM-APPROX)*, 2001.

[45] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[46] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

[47] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable ad hoc routing: The case for dynamic addressing. In *Proc. of the* 23^{rd} *Conference of the IEEE Communicatons Society (INFOCOM)*, 2004.

[48] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. In *Proc. of the* 36^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995.

[49] U. Feige. Approximating the bandwidth via volume respecting embeddings. *J. of Computer and System Sciences*, 60(3):510–539, 2000.

[50] J. Feigenbaum and S. Kannan. Dynamic graph algorithms. In K. H. Rosen, editor, *Handbook of Discrete and Combinatorial Mathematics*, chapter 17.1, pages 1142–1148. CRC Press, 2000.

[51] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *Proc. of the* 2^{nd} *ACM Intl. Workshop on Principles of Mobile Computing (POMC)*, 2002.

[52] K. Fischer, B. Gärtner, and M. Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *Proc. of the* 11^{th} *European Symposium on Algorithms (ESA)*, 2003.

[53] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.

[54] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communication*, 2(1):11–18, 1981.

[55] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical Report UCLA/CSD-TR 02-0013, UCLA Computer Science, 2002.

[56] F. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical Report IC/2003/38, Swiss Federal Institute of Technology (EPFL), 2003.

[57] C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. *Journal of Graph Algorithms and Applications*, 9(3):327–349, 2005.

[58] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *Proc. of the* 20^{th} *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2001.

[59] M. Grossglauser and M. Vetterli. Locating nodes with EASE: Last encounter routing in ad hoc networks through mobility diffusion. In *Proc. of the* 22^{nd} *IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[60] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1998.

[61] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[62] Z. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proc. of the* 6^{th} *IEEE Intl. Conference on Universal Personal Communications (ICUPC)*, 1997.

[63] Z. J. Haas and M. R. Pearlman. ZRP: A hybrid framework for routing in ad hoc networks. In *Ad Hoc Networking*, pages 221–253. Addison-Wesley, 2001.

[64] M. Heissenbüttel. *Routing and Broadcasting in Ad-Hoc Networks*. PhD thesis, Universität Bern, 2005.

[65] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning Systems: Theory and Practice*. Springer, 5th edition, 2001.

[66] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *Proc. of the* 2^{nd} *ACM Intl. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 1999.

[67] P. Indyk and J. Matoušek. *Handbook of Discrete and Computational Geometry*, chapter Discrete metric spaces. CRC Press, second edition, 2004.

[68] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. In *Proc. of the* 9^{th} *Intl. Conference on Mobile Computing and Networking (MobiCom)*, 2003.

[69] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proc. of the* 5^{th} *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 1999.

[70] D. B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. of the* 1^{st} *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[71] D. B. Johnson and D. A. Maltz. *Mobile Computing*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks. Kluwer Academic Publishers, 1996.

[72] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[73] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[74] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Lecture Notes in Computer Science Tutorial 2025, Springer, 2001.

[75] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proc. of the* 45^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

[76] L. Kleinrock and J. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. In *Proc. of the IEEE National Telecommunications Conference*, 1978.

[77] A. Korman and D. Peleg. Dynamic routing schemes for general graphs. In *Proc. of the* 33^{rd} *Intl. Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[78] A. Korman, D. Peleg, and Y. Rodeh. Labeling schemes for dynamic tree networks. In *Proc. of the* 19^{th} *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2002.

[79] J. Kruskal and J. Seery. Designing network diagrams. In *Proc. of the* 1^{st} *GeneralConference on Social Graphics, U. S. Department of the Census*, 1980.

[80] F. Kuhn. personal communication, May 2006.

[81] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In *Proc. of the* 19^{th} *International Symposium on Distributed Computing (DISC)*, 2005.

[82] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Local approximation schemes for ad hoc and sensor networks. In *Proc. of the* 3^{rd} *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2005.

[83] F. Kuhn, T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. *Algorithmica*, 2006. to appear.

[84] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit disk graph approximation. In *Proc. of the* 2^{nd} *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2004.

[85] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *Proc. of the* 24^{th} *ACM Symposium on the Principles of Distributed Computing (PODC)*, 2005.

[86] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proc. of the Fourth Int'l Workshop on Peer-To-Peer Systems (IPTPS)*, 2005.

[87] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proc. of the* 22^{nd} *ACM Symposium on the Principles of Distributed Computing (PODC)*, 2003.

[88] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proc. of the* 22^{nd} *ACM Symposium on Principles of Distributed Computing (PODC)*, 2003.

[89] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proc. of the* 1^{st} *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2003.

[90] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proc. of the* 4^{th} *ACM Intl. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

[91] B.-J. Kwak, N.-O. Song, and L. E. Miller. A mobility measure for mobile ad hoc networks. *IEEE Communication Letters*, 7(8):379–381, 2003.

[92] T. J. Kwon and M. Gerla. Clustering with power control. In *Proc. of IEEE Military Communications Conference (MilCOM)*, 1999.

[93] T. Larsson and N. Hedman. Routing protocols in wireless ad-hoc networks. Master's thesis, Luleå Tekniska Universitet, 1998.

[94] J.-Y. Le Boudec and M. Vojnovic̀. Perfect simulation and stationarity of a class of mobility models. In *Proc. of the* 24^{th} *IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[95] G. Lin, G. Noubir, and R. Rajaraman. Mobility models for ad hoc network simulation. In *Proc. of the* 23^{rd} *IEEE Conference on Computer Communications (INFOCOM)*, 2004.

[96] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[97] Z. Lotker, M. M. de Albeniz, and S. Pérénnes. Range-free ranking in sensors networks and its applications to localization. In *Proc. of Ad-Hoc, Mobile, and Wireless Networks:* 3^{rd} *Intl. Conference (ADHOC-NOW)*, 2004.

[98] N. Lynch, S. Mitra, and T. Nolte. Motion coordination using virtual nodes. In *Proc. of the* 44^{th} *IEEE Conference on Decision and Control and European Control Conference*, 2005.

[99] J. G. Markoulidakis, G. L. Lyberopoulos, D. F. Tsirkas, and E. D. Sykas. Mobility modeling in third generation mobile telecommunication systems. *IEEE Personal Communications*, 4(4):41–56, 1997.

[100] J. Matoušek. *Lectures on Discrete Geometry, Graduate Texts in Mathematics (GTM) 202*, chapter Embedding Finite Metric Spaces into Normed Spaces. Springer, 2002.

[101] J. Matoušek and U. Wagner. personal communication, July 2004.

[102] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proc. of the* 2^{nd} *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2004.

[103] T. Moscibroda and R. Wattenhofer. The complexity of connectivity in wireless networks. In *Proc. of the* 25^{th} *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2006.

[104] T. Munzner and P. Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *Proc. of the* 1^{st} *Symposium on Virtual Reality Modeling Language (VRML)*, 1995.

[105] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *Proc. of the* 2^{nd} *International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.

[106] E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. of the* 21^{st} *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[107] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proc. of the* 44^{th} *IEEE Global Communications Conference (GLOBECOM)*, 2001.

[108] D. Niculescu and B. Nath. Ad hoc positioning system (APS) using A0A. In *Proc. of the* 22^{nd} *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[109] D. Niculescu and B. Nath. DV based positioning in ad hoc networks. *Journal of Telecommunication Systems*, 22(1-4):267–280, 2003.

[110] D. Niculescu and B. Nath. Error characteristics of ad hoc positioning systems (APS). In *Proc. of the* 5^{th} *ACM Intl. Symposium on Mobile Ad hoc Networking and Computing (MobiHoc)*, 2004.

[111] M. O'Dell, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Lost in space or positioning in sensor networks. In *Proc. of the* 1^{st} *Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.

[112] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proc. of the* 3^{rd} *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2005.

[113] R. O'Dell and R. Wattenhofer. Theoretical aspects of connectivity-based multi-hop positioning. *Theoretical Computer Science*, 344(1):47–68, 2005.

[114] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. of the* 16^{th} *IEEE Conference on Computer Communications (INFOCOM)*, 1997.

[115] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, 1994.

[116] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the* 2^{nd} *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 1999.

[117] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. of the* 6^{th} *ACM Intl. Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[118] N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Poster abstract: Anchor-free distributed localization in sensor networks. In *Proc. of the* 1^{st} *Intl. Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[119] N. B. Priyantha, H. Balakrishnan, E. D. Demaine, and S. Teller. Mobile-assisted localization in wireless sensor networks. In *Proc. of the* 24^{th} *Conference of the IEEE Communicatons Society (INFOCOM)*, 2005.

[120] V. Raghavan and J. Spinrad. Robust algorithms for restricted domains. *Journal of Algorithms*, 48(1):160–172, 2003.

[121] R. Rajaraman. Topology control and routing in ad hoc networks: A survey. *ACM SIGACT News*, 33(2):60–73, 2002.

[122] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Localization from mere connectivity. In *Proc. of the* 9^{th} *ACM Intl. Conference on Mobile Computing and Networking (MobiCom)*, 2003.

[123] T. Rappaport. *Wireless Communications: Principles & Practice*. Prentice Hall, 1996.

[124] K. Römer. The lighthouse location system for smart dust. In *Proc. of the* 1^{st} *ACM/USENIX Intl. Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2003.

[125] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *Proc. of the* 36^{th} *IEEE Intl. Conference on Communications (ICC)*, 2001.

[126] P. Samar, M. R. Pearlman, and Z. J. Haas. Independent zone routing: An adaptive hybrid routing framework for ad hoc wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 12(4):595–608, 2004.

[127] N. Sarafijanovic-Djukic and M. Grossglauser. Last encounter routing under random waypoint mobility. In *Proc. of the* 3^{rd} *IFIP-TC6 Networking Conference*, 2004.

[128] M. Särelä. Measuring the effects of mobility on reactive ad hoc routing protocols. Technical Report HUT-TCS-A91, Helsinki University of Technology, 2004.

[129] C. Savarese, J. Rabaey, and K. Langendoen. Robust positioning algorithms for distributed ad-hoc wireless sensor networks. In *Proc. of the USENIX Technical Conference*, 2002.

[130] ScatterWeb GmbH. `http://www.scatterweb.net`, 2004.

[131] C. Schindelhauer, T. Lukovszki, S. Rührup, and K. Volbert. Worst case mobility in ad hoc networks. In *Proc. of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2003.

[132] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Proc. of the* 14^{th} *Intl. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2006.

[133] K. Seada, A. Helmy, and R. Govindan. On the effect of localization errors on geographic face routing in sensor networkse. In *Proc. of* 3^{rd} *Intl. Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.

[134] R. Shah, A. Wolisz, and J. Rabaey. On the performance of geographical routing in the presence of localization errors. In *Proc. of the* 40^{th} *IEEE Intl. Conference on Communications (ICC)*, 2005.

[135] Y. Shang and W. Ruml. Improved MDS-based localization. In *Proc. of the* 23^{rd} *Conference of the IEEE Communicatons Society (INFOCOM)*, 2004.

[136] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. Localization from mere connectivity. In *Proc. of the* 4^{th} *ACM Intl. Symposium on Mobile Ad hoc Networking and Computing (MobiHoc)*, 2003.

[137] A. M.-C. So and Y. Ye. Theory of semidefinite programming for sensor network localization. In *Proc. of the* 16^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[138] E. Soedarmadji and R. McEliece. A dynamic graph algorithm for the highly dynamic network problem. In *Proc. of the* 1^{st} *IEEE Intl. Workshop on Foundation and Algorithms for Wireless Networking (FAWN)*, 2006.

[139] D. Son, B. Krishnamachari, and J. Heidemann. Experimental study of the effects of transmission power control and blacklisting in wireless sensor networks. In *Proc. of the* 1^{st} *IEEE Conference on Sensor and Adhoc Communication and Networks*, 2004.

[140] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. of the* 13^{th} *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001.

[141] V. Tolety. Load reduction in ad hoc networks using mobile servers. Master's thesis, Colorado School of Mines, 1999.

[142] D. Turgut, S. K. Das, and M. Chatterjee. Longevity of routes in mobile ad hoc networks. In *Proc. of the IEEE Vehicular Technology Conference (VTC 2001 Spring)*, 2001.

[143] S. Vempala. Random projection: A new approach to vlsi layout. In *Proc. of the* 39^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1998.

[144] S. Vempala. *The Random Projection Method*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 2004.

[145] M. Wattenhofer, R. Wattenhofer, and P. Widmayer. Geometric routing without geometry. In *Proc. of the* 12^{th} *Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2005.

[146] M. M. Zonoozi and P. Dassanayake. User mobility modeling and characterization of mobility patterns. *IEEE Journal of Selected Areas in Communications*, 15(7):1239–1252, 1997.

Acknowledgements

I owe my greatest debt of gratitude to my family—my foundation in life. Mamulya and Vati, none of this would have been possible without your unwavering love and support throughout my life. I also feel very fortunate to have parents-in-law who welcomed me from the very beginning and who have been there for us with so much warmth ever since. I want to thank Mike for the continued exceptional fulfillment of all his "husbandly duties" of partner, father, and friend. And, of course, the brightest stars in the sky, Damian, for being incontestable proof of how wonderful children can be, and the little one, for behaving so well and staying inside until the completion of this dissertation.

Any words must pale in comparison to how I feel about all of you.

No PhD is possible without the dedicated guidance and support for which I would like to thank my advisor, Roger Wattenhofer. I very much enjoyed the keen technical discussions we had, the variety of insights you gave into all aspects of academia, and the challenging working environment you provided.

My thanks also extend towards my coexaminers, Rajmohan Rajaraman and Dorothea Wagner, for investing their time into reading this dissertation and providing valuable comments and questions.

I am particularly grateful to my coworkers at the Distributed Computing Group for the great working environment they have created. In particular, I want to thank Dr. Aaron Zollinger, fellow vegetarian, for patiently sharing with me his thoughts on all matters and passion about languages, and for showing us all the way of saintly reverence; Dr. Fabian Kuhn, fellow topologist, for effortlessly planting a sharp intellect on firm moral ground; Dr. Keno Albrecht, fellow Siedler, for discussing the cooking of fine food, monkey business, and all things in between with me and generally offering his unsolicited but delightful opinion on anything and everything; Dr. Thomas Moscibroda, fellow fervent interlocutor, for his entertaining and cheerfully heated daily debates; Pascal von Rickenbach, fellow temporarily incapacitated sportsman, and his evil twin Patrick for providing the proper counter weight to my alleged feminism; Nicolas Burri, fellow big-bellies-rule! advocator, for challenging every theory of mine with practical arguments; and Andi Wetzel, fellow linuxer, for patiently tackling any of my obscure computer questions and his amicable sense of humor.

I also thank the new generation of DCG, Stefan Schmid, Roland Flury, Michael Kuhn, Olga Goussevskaia, Yves Weber, Thomas "Anglo-Saxon virtuoso" Locher, and Yvonne Anne Oswald, for upholding the torch.

And last but certainly not least, due reverence goes to my office mate and honorary DCG member, Mirjam Wattenhofer, fellow Fr. Dr. Mama, for the invaluable discussions of greatly varying technical degree and listening to all my strange theories about life as we know it. Our twin careers at the ETH have come to an end, leaving me to wonder where next our paths will cross.

You've been a great bunch to work with! So long, and thanks for all the cake ...

Curriculum Vitae

Apr. 26, 1980	Born in Kazan, Russia
1986–1997	primary, secondary, and high schools in Germany, Russia, and USA
1998–2003	Studies in computer science, ETH Zurich, Switzerland
Apr. 2003	Diploma in computer science, ETH Zurich, Switzerland
2003–2006	Ph.D. student, research and teaching assistant, Distributed Computing Group, Prof. Roger Wattenhofer, ETH Zurich, Switzerland
Sept. 2006	Ph.D. degree, Distributed Computing Group, ETH Zurich, Switzerland Advisor: Prof. Roger Wattenhofer Co-examiners: Prof. Rajmohan Rajaraman Northeastern University, Boston, USA Prof. Dorothea Wagner Universität Karlsruhe, Germany