Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 344 (2005) 47–68

Theoretical Computer Science

www.elsevier.com/locate/tcs

Theoretical aspects of connectivity-based multi-hop positioning[☆]

R. O'Dell[*], R. Wattenhofer

Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland

Abstract

We investigate the theoretical limits of positioning algorithms. In particular, we study scenarios where the nodes do not receive anchors directly (multi-hop) and where no physical distance or angle information whatsoever is available (connectivity-based). Since we envision large-scale sensor networks as an application, we are interested in fast, distributed algorithms. As such, we show that plain hop algorithms are not competitive. Instead, for one-dimensional unit disk graphs we present an optimal algorithm HS. For two or more dimensions, we propose an algorithm GHOST which improves upon the basic hop algorithm in theory and in simulations.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Positioning; Localization; Multi-hop; Connectivity-only; Unit disk graph

1. Introduction

The availability of a global positioning system (GPS) has spawned a multi-billion dollar market for positioning with an enormous variety of transportation, industry, and recreation applications. Apparently "knowing your position'' opens up a multiplicity of exciting possibilities. An increasing research activity in the recent years documents that

[☆] The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under Grant no. 5005-67322. A preliminary version of this paper appeared at the Second IEEE International Conference on Pervasive Computing and Communications (PerCom) in 2004.

[*] Corresponding author. Current address: ETZ G61.1, ETH Zentrum, CH-8092, Zürich, Switzerland. Tel.: +41 44 632 4776.

E-mail addresses: odell@tik.ee.ethz.ch (R. O'Dell), wattenhofer@tik.ee.ethz.ch (R. Wattenhofer).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.06.026

position-awareness is also a key pervasive computing technology—for instance for wireless ad-hoc or sensor networks. Especially in sensor networks positioning is indispensable: Sensing the environment without attaching “coordinates’’ to the sensed data seems unusual.

Unfortunately, not every sensor network node can be equipped with a GPS receiver. A GPS receiver is clumsy, heavy, and expensive—quite the opposite of a sensor node which ought to be small, light, and cheap (“smart dust’’). Because of physical constraints a GPS receiver will remain an order of magnitude more expensive (dimension-, weight-and money-wise) than a sensor node. Moreover, GPS receivers do not function properly indoors.

Nonetheless, realistic sensor networks with positioning information are feasible. The idea is to equip a small fraction of the nodes with a GPS receiver. We call nodes that know their position *anchor* nodes. Clearly an anchor node does not necessarily need to learn its position by means of a GPS receiver; other technologies are as welcome, one might even consider keeping an anchor node immobile at all times and hard-code the anchors’ position into its ROM at deployment.

Since only a small fraction of nodes are anchors, most sensor nodes remain small, light, cheap, and—“*dumb*.’’ A dumb node must learn its (approximate) position with the help of the anchor nodes, and the other nodes.

In this paper we study the problem where most dumb nodes do not receive the signal of any anchor node directly. Instead a dumb node must learn its position through multi-hop paths of other dumb nodes to anchor nodes. We allow the dumb sensor nodes to be truly cost-effective: A node can neither learn distance from nor direction to a direct neighbor, not even approximately. By means of beacon signals, nodes can solely derive connectivity information. In other words, receiving a neighbor’s signal a node can merely conclude that the neighbor is closer than the maximum transmission radius. We name this model “connectivity-based multi-hop.’’ We believe that this most closely resembles realistic situations where questions of cost and even accessibility dominate the design.

To our knowledge, all previous positioning algorithms for the connectivity-based model build their estimations upon hops. A dumb node computes the number of hops to several anchor nodes, and then uses the set of tuples (coordinate of anchor, hops to anchor) to approximate its position. Some algorithms iterate this process to improve their position approximations.

In this paper we show that algorithms based exclusively on the number of hops do not approximate positions well. In fact, already for a simplified pet environment where all nodes lie on a straight line (e.g. a highway), such algorithms will generate larger than necessary errors. Surprisingly, a simple positioning algorithm we call HS (which stands for Hop-Skip) that has the same asymptotic time complexity as the basic hop-based algorithm will guess a position *optimally* in one dimension.

The analysis of the hop-based algorithm and the HS algorithm—and the lessons learned—enable us to devise a new algorithm GHoST for multiple dimensions which improves upon the hop-based algorithms.

The paper is organized as follows. In Section 2 we overview work directly related to localization/positioning algorithms proposed in the literature. In Section 3 we present a formal model for our analysis. In Sections 4, 5, and 6 we study three different positioning algorithms. We first look at a simple hop-based algorithm HOP that will serve as a basis of comparison for the efficiency of the more complex algorithms. Most importantly, we prove in Section 4.3 that HOP is not optimal. We go on to outline an optimal algorithm HS for one-dimensional unit disk graphs in Section 5. Our tour of positioning algorithms ends in Section 6 with GHOST, a general two-dimensional algorithm which improves upon HOP. In Section 7 we conclude the paper.

2. Related work

2.1. Motivation

The global positioning system (GPS) was proposed by the US Navy in the 1960's; the first working prototype was deployed in 1978 [10].[1] GPS is the most successful example of a *single-hop* positioning system. In a single-hop positioning system, a node receives the signals of several anchor nodes directly. A GPS receiver computes its position by means of the time [difference] of arrival (T[D]OA) technology, where distance-to-anchor information is deduced through the time of a signal propagation. Other single-hop positioning systems use the received signal strength indicator (RSSI), or the angle of arrival (AOA) method. For an example of the RSSI method, see [2]; for an example of the AOA method, see [20].

In this paper we study *multi-hop* positioning systems, that is, systems where nodes typically do not receive the anchor nodes' signals directly. We believe that multi-hop networks are more realistic in future scenarios. Additionally, they allow for a lower deployment cost since less powerful anchor nodes are required. Given the influence of single-hop positioning systems, it is not surprising that the first multi-hop proposals tried to adapt the single-hop technologies. T[D]OA, RSSI, and/or AOA information is collected and then the position of each node is computed using triangulation [3,7,25].

Another important aspect of our model is that we are primarily interested in *connectivity-based* scenarios. In other words, the information available to the nodes is whether or not they are connected, without knowing their (approximate) distances. The reasons for studying such a model are manifold. First of all, knowing even estimates of inter-node distances requires precise and specialized hardware. The commonly used signal strength measurements of the radio transceivers are unreliable and unstable in realistic scenarios. If we know how well we can position nodes in the connectivity model, then we can study the cost-benefit tradeoff between more accurate localization and cheaper deployment. Second, it has recently been demonstrated that weak measurement instruments are not a guarantee for improvement. Niculescu and Nath [21] showed that a connectivity-only algorithm outperforms measurement-based ones when the error of the devices is above a certain threshold. Similar simulation results appear in [4]. A third reason for this model is that, in most cases, it is easier to adapt algorithms which are based on connectivity only to incorporate distance estimates than the other way around. In our case, we will discuss this in more detail in Section 6.

Another reason to study localization, apart from knowing the coordinates for their own sake, is a distinguished application on top of a positioning algorithm, namely geo-routing (a.k.a. geometric, geographic, location, or position-based routing). A geo-routing algorithm

[1] The first GPS receiver for civil use cost $150k in 1984 and required two people to carry it.

needs all nodes to know their coordinates (by means of a GPS, or a local positioning algorithm). The coordinates are then used to route messages towards their destinations *in lieu* of routing tables. Early proposals of geo-routing date back twenty years [27]. The first efficient geo-routing algorithm was GFG/GPSR [6,11], and the currently best geo-routing algorithm is GOAFR+ [14,15].

As noticed by several researchers independently [23,26] it is not essential to have anchor nodes at all. Without anchor nodes available, all nodes get assigned "virtual coordinates'' that reflect the graph topology well. Again, these coordinates will be used to run any geo-routing algorithm. Although not mentioned explicitly in the remainder of the paper, our results also (partially) apply to computing virtual coordinates.

2.2. Heuristics

A number of (almost) connectivity-based solutions have been suggested in the literature. One of the simplest and earliest is DV-Hop [22] (as part of a system known as APS [19]). A node determines how many hops away it is from an anchor node. The anchor nodes compute their hops to other anchors as well and use a simple formula to determine the average hop length (i.e. a hop length is estimated as 0.86 instead of 1). The anchors then broadcast this information. Having such distance estimates from sufficiently many anchors a node locally performs a least square method calculation to determine its position (as it is done in GPS). In APS, additional possibilities for the first distance estimates are suggested which are not connectivity-based. It is, therefore, possible to use the ideas and methods of anchor distance estimation of this paper and combine them with triangulation methods such as in APS.

A method similar to APS has been suggested in [18]. It first determines the hop distance (called gradient) to the anchors (called seeds) and—as a function of the average node density—calculates the average actual hop distance to an anchor by the Kleinrock–Silvester formula [12]. Observe that knowledge of the global average node density (measured as the number of nodes per unit disk) is critical to this algorithm's performance and needs to be calculated and propagated separately.

Simulation results in [19,18] show that these algorithms (in their connectivity-based variants) only perform well under high-density conditions: APS with DV-Hop needs more than 20% of the nodes to be anchors to stabilize at an average error of about 30% of the radio range and no data is available for less than 5% of anchors; the algorithm in [18] needs a node density of more than 15 nodes per disk but already stabilizes at about 8% to 10% of anchors.

A recent proposal by He et al. [9], dubbed "range-free,'' determines whether a node lies inside or outside of the triangles formed by all 3-tuples of anchors (called "APIT test''). This creates an area of possible locations for the node in which the center of gravity is chosen. In order to perform the APIT test, however, information about the *relative* distances of anchors to the nodes is necessary (i.e. whether one anchor, in a certain general direction, is closer to a node than another anchor). Thus it does not fit our criteria for connectivity-based algorithms although it does have less requirements on the physical capabilities of nodes. Additionally, anchor signals need to be received directly, thus [9] is a single-hop positioning system.

Another hop-based approach [24] contains the key concept of *refinement*. Among other heuristics, the main idea is to iterate the position estimation process: Once the nodes have an estimate of their positions along with a confidence interval, information is exchanged again to recompute estimates. The drawback of such an iteration is that it is far more time consuming and it is not clear how many iterations need to be performed until a desired accuracy is achieved. A similar iterative approach is given by Rao et al. in [23], where nodes position themselves as the average of their neighbors' positions, modeling the idea of nodes being connected by (equal) springs. However, their approach was developed primarily for virtual coordinates and the positioning part is only effective in specific scenarios, namely when the anchors are placed along the perimeter of the network.

While the above algorithms are distributed and aim at being efficient in large networks, a number of centralized approaches have been proposed as well. One of the earliest is by Doherty et al. [8], who formulates the positioning problem as a set of convex constraints to be solved. Recently, Biswas and Ye [5] also formulate the problem as a set of constraints, albeit with distance measurements, and show that it can be solved efficiently in a dense network with semidefinite programming techniques. A similar idea can also be found in the multi-dimensional scaling (MDS) approach by Shang et al. [26]. The key issue with these algorithms is that a single node needs to know the entire graph topology and perform a computationally expensive calculation.

The most significant difference of our approach to the above-related work is that we try to ascertain theoretical bounds for connectivity-based algorithms independent of any random distribution assumptions. We will briefly review the other recent advancements in the theoretical understanding of the localization problem in the following section. We also aim for fast and effective algorithms that achieve those lower bounds in any scenario by comparing to an omniscient optimal algorithm as opposed to an optimal but centralized solution. Additionally, as will become evident in Section 6, the algorithm presented can easily incorporate exact (or good estimates of) distances.

2.3. *Hard results*

Heuristic approaches potentially perform poorly in arbitrary (worst-case) scenarios. Provable theoretical results concerning the potential of virtual coordinates have only been given very recently.[2] In [16], the authors show that it is APX-hard to embed a unit disk graph satisfying all the constraints. Independently, [13] show an even stronger result, namely that it cannot even be embedded with quality better than $\sqrt{3/2}$ (where 1 is the optimum), implying that there cannot be a PTAS for the UDG embedding problem. If not only connectivity information is given, but all edge distances, then [1] show that embedding such a unit disk graph is still NP-hard. As mentioned before, [5] show that this is, however, possible when the graph is dense, that is, there are $\Omega(|V|^2)$ edges in the graph. Yet this implies a highly specialized scenario where all nodes know their exact edge lengths and the graph is very dense. The authors of [17] give an algorithm for virtual coordinates with a guaranteed approximation ratio for any unit disk graph. The gap between the bound $O(\log^{2.5} n\sqrt{\log\log n})$ in [17] and $\sqrt{3/2}$ in [13] remains to be closed.

[2] As opposed to positioning (with anchors), the virtual coordinates model (without anchors) is "cleaner'' (less parameterized), and therefore more accessible for hard approximation results.

Note that the above all hold in the anchor-free (i.e., virtual coordinates) setting, yet the lower bounds certainly apply to positioning in the worst case. More importantly, the proposed schemes with provable guarantees are centralized and computationally expensive (at that one node), so the question that remains and is addressed in this paper is how much can be achieved with a distributed algorithm. In other words, what is the tradeoff between the amount of information collected about a graph and the accuracy of the localization.

3. Model

In our paper we model a given physical sensor/ad-hoc network as a graph. A *graph* $G = (V, E)$ is a set of nodes V (representing the nodes of the network) and a set of edges E, connecting the nodes; there is an edge between two nodes u and v if and only if the nodes u and v are within mutual transmission range.

We study Euclidean graphs, that is, graphs where each node has a coordinate in d-dimensional space. More formally, a d-dimensional *embedding* of G is a coordinate function $\mathit{coord} : V \to \mathbb{R}^d$ on the nodes. Throughout the paper we make the standard assumption that the transmission range of each node is 1 (by scaling the coordinate system). A graph G is a *unit disk graph* (UDG) if it has an embedding such that the Euclidean distance $\mathrm{dist}_{\mathrm{E}}(\mathit{coord}(v), \mathit{coord}(u)) \leqslant 1 \Leftrightarrow \{v, u\} \in E$. In the paper, we consider the coordinate embedding as given (but invisible to a positioning algorithm).

Apart from the Euclidean distance $\mathrm{dist}_{\mathrm{E}}(\cdot)$ between two points in $\mathbb{R}^d$, there is a distance in graphs independent of any embedding. A *hop* between $u, v \in V$ is an edge $e = \{u, v\} \in E$ in a graph $G = (V, E)$. A *path* of length k is a sequence $P = v_0 v_1 \ldots v_k$ where $v_i \neq v_j$ for $i \neq j$ and $\{v_i, v_{i+1}\} \in E$ for $0 \leqslant i < k$. The *graph distance* $\mathrm{dist}_{\mathrm{G}}(u, v)$ between two nodes $u, v \in V$ is the length of a shortest path between u and v in G.

The distinction between graph and Euclidean distance is crucial in the sense that the physical network and any algorithm operating on it see only the graph distances from which they try to ascertain the actual Euclidean distances between nodes. The problem we study can thus be formalized as follows.

Problem 1. Given a graph G with an unknown embedding *coord* as a UDG, the problem of *absolute positioning* is for the nodes V to compute an embedding *pos* such that $\mathrm{dist}_{\mathrm{E}}(\mathit{coord}(v), \mathit{pos}(v))$ is minimized $\forall v \in V$. A subset of nodes $\mathit{Anchors} \subset V$ are called anchor nodes. A node $A \in \mathit{Anchors}$ knows its position, that is $\mathit{pos}(A) = \mathit{coord}(A)$. The *error* of an algorithm for a node v is $\mathit{Error}_{\mathrm{ALG}}(v) = \mathrm{dist}_{\mathrm{E}}(\mathit{coord}(v), \mathit{pos}(v))$. The *maximum error* is then

$$\mathit{MaxErr}_{\mathrm{ALG}}(v) = \max_{\mathit{coord}} \mathit{Error}_{\mathrm{ALG}}(v)$$

ranging over all possible embeddings of G.

We are studying distributed algorithms according to the following (standard) model. When a node v transmits a message (pseudo code "transmit `msg`''), all the neighbors of v (denoted by $N(v) = \{u \mid \{u, v\} \in E\}$) will eventually receive the message.

In a synchronous setting, communication is modeled as proceeding in rounds: In one round, all messages from the previous round are received, processed, and new messages transmitted. Since the real world does not always obey the rules of synchrony, we also study the *asynchronous* model, where the delay of a message is finite, but potentially unbounded. All the algorithms presented in the paper also behave correctly in an asynchronous setting.

Besides the error of a positioning algorithm, defined in Problem 1, we study the standard distributed computing costs, that is, message and time complexity. The message complexity counts the number of messages transmitted by the nodes over an edge. In the synchronous model, the time complexity counts the units of time that passed from the start of the algorithm until the nodes have computed their position. In the asynchronous model, the time complexity is defined likewise, with the assumption that all messages incur at most a delay of one time unit.

4. The HOP algorithm

4.1. General outline of algorithms

The positioning algorithms we consider in this paper consist of two parts: the gathering of connectivity information and a local calculation that computes the position 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 $pos(v)$ in the sense that it minimizes $MaxErr(v)$. The main difference then lies in the information gathering phase. In this section, we will examine first a simple algorithm.

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

Definition 2. The set of graph distance-h nodes $D_h(A)$ for a node $A \in V$ is

$$D_h(A) = \{v \in V \mid \mathrm{dist}_G(A, v) = h\}.$$

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

4.2. The HOP algorithm

The HOP algorithm is described below. To start the algorithm, an anchor node A transmits the message `(pos(A),1)`.

```
1: **hops** := ∞;
2: upon receipt of (pos(A),h)
3: **if** (**h** < **hops**) **then**
4:    **hops** := **h**
5:    transmit (pos(A),h+1)
6: **end if**
```

Lemma 3. *The* HOP *algorithm finds the graph distance h from an anchor node A to a 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 D_1 will eventually receive the transmission from A, thereby correctly setting **hops** to 1. This will be at time $\leqslant 1$ from the point where A sends the first message. For the induction step, assume that all nodes in D_{h-1} have received their distance at time $\leqslant h-1$. Then, by the definition of $\mathrm{dist}_{\mathrm{G}}()$, $v \in D_h$ has at least one neighbor $u \in D_{h-1}$ (and none in D_{h-2}). Since all $w \in D_{h-1}$ transmit exactly `(pos(A),h)`, v will receive this message at least once (from u) and set **hops** to h. The transmission from u to v will take at most 1 time step so that v determines its distance within h time units. □

Lemma 4. *In the asynchronous model, the* HOP *algorithm has message complexity* $2n-1$ *for an edge e, where n is the number of nodes in the graph. In the synchronous model, message complexity is* 2.

Proof. For message complexity, we look at the maximum number of messages exchanged across a link as it is represented by an edge in the graph. If there are n nodes in the graph, then the maximum **hops** that a node v can receive initially in an asynchronous model is n. Consider the edge e leading into v and coming out of a node u with initial hop count $n-1$. Observe that v transmitting `(pos(A),n+1)`, while being rejected by all nodes, will add another message to e. Thereafter, v will accept and transmit only lower-count messages from and to u, in the worst case (where all non-anchors are in D_1) down until its own hop counter is at 2. Then v has received and transmitted $2(n-1)$ messages over e. When v finally hears from A, then it sends one final message to u with hop counter 2 for a total of $2n-1$ transmissions over e.

In the synchronous case, v first hears from nodes $u \in D_{h-1} \cap N(v)$. Thus one message is transmitted over an edge $e = \{u, v\}$ after $h-1$ communication rounds. In the next round, v transmits a hop count of $h+1$ over e, bringing e's transmission count up to 2. That same message with $h+1$ is also received by nodes $w \in D_h$ (and vice verse when w sends to v) so that two messages are exchanged over an edge $e' = \{v, w\}$. □

The interval for a node v at graph distance h from anchor A is then bounded by $h/2 < \mathrm{dist}_{\mathrm{E}}(A, v) \leqslant h$ in one dimension and the position is reconstructed from the mid-point of the intersection of all such intervals. We will postpone the discussion of higher-dimensional "mid-points'' to Section 6.

4.3. Competitive analysis of HOP

We want to compare the HOP algorithm to an optimal one. Let us first define optimality.

Definition 5. An *optimal algorithm* OPT is one which knows the entire combinatorial structure of the graph $G = (V, E)$ and then chooses the position in order to minimize the maximal

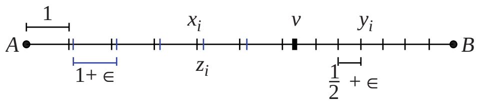


Fig. 1. Instance of a UDG G where the HOP algorithm is significantly outperformed by an optimal algorithm.

possible error. The *competitive ratio* of a positioning algorithm ALG is c if

$$MaxErr_{\text{ALG}}(v) \leqslant c \cdot MaxErr_{\text{OPT}}(v) + k$$

for all $v \in V$ and some constant k. We say that ALG is c-competitive.

Lemma 6. *The* HOP *algorithm is not competitive.*

Proof. Let $d_{AB} = d$ be the Euclidean distance between anchors A and B. We will 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.

Consider a unit disk graph G as in Fig. 1. Let h be the graph distance of a node v to both A and B. Suppose G has $n = 3h - 1$ nodes. There are h nodes that form the only shortest path from A to v (excluding v), we call them $x_0 = A, x_1, \ldots, x_{h-1}$; there are h nodes from B to v, $y_0 = B, y_1, \ldots, y_{h-1}$; 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 $z_i \in D_{i+1}(A)$. Setting $coord(A) = 0$, the actual coordinates are

$$\begin{aligned} coord(x_i) &= i, & coord(y_i) &= d - (\tfrac{1}{2} + \varepsilon)i, \\ coord(z_i) &= i(1+\varepsilon), & coord(v) &= (h-1)(1+\varepsilon), \end{aligned}$$

for some arbitrarily small ε with $1/(h-1) > \varepsilon > 0$. This gives $d = (h-1)(1+\varepsilon) + h(\frac{1}{2}+\varepsilon) = \frac{3}{2}h + ((h-1)\varepsilon - 1)$.

The HOP (and also DV-Hop) algorithms will receive the information `(0,h)` about A and `(d,h)` about B. By the symmetry of the hop information, any hop-based algorithm will put $pos(v) = d/2 = \frac{3}{4}h + \frac{1}{2}((h-1)\varepsilon - 1)$. The error for v is $Error_{\text{HOP}}(v) = \frac{h}{4} + \frac{1}{2}((h-1)\varepsilon - 1)$ or almost $d/6$.

An optimal algorithm will be able to deduce from the connectivity information that $\text{dist}_{\text{E}}(z_i, z_{i+1}) > 1$ and therefore $\text{dist}_{\text{E}}(z_1, z_{h-1}) = \text{dist}_{\text{E}}(z_1, v) > h - 2$. Since $A \notin N(z_1)$, the optimal algorithm can conclude that $\text{dist}_{\text{E}}(A, v) > h - 1$. Thus $Error_{\text{OPT}}(v) < (h-1)\varepsilon \ll 1$ and

$$\begin{aligned} Error_{\text{HOP}}(v) &> \frac{h}{4} + \frac{1}{2} Error_{\text{OPT}}(v) - \frac{1}{2}, \\ &> \left(\frac{h}{4} + \frac{1}{2}\right) Error_{\text{OPT}}(v) - \frac{1}{2}, \end{aligned}$$

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

Note that although the counter example against HOP is one-dimensional, the non-competitiveness of HOP holds for all dimensions. □

5. The HS algorithm

In this section, we will examine strictly one-dimensional unit disk graphs. The reason being that we are interested in tight lower bounds for the accuracy of positioning algorithms in *any* situation. Understanding these worst-case scenarios will help us devise better algorithms for "normal'' (i.e. average) scenarios.

5.1. Preliminaries

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

Definition 7 (*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 $\{u, v\}, \{v, w\} \in E$.

Definition 8 (*Skip Distance*). A sequence of nodes $SP = v_0 v_1 \dots v_k$ is a *skip path* of length *k* if
(i) $\{v_i, v_j\} \notin E$ for all $i \neq j$ and $\text{dist}_G(v_0, v_i) < \text{dist}_G(v_0, v_{i+1})$ and
(ii) $\exists u_i$ such that $P = v_0 u_1 v_1 \dots u_k v_k$ is a path.
The length of the longest skip path between $u, v \in V$ is the *skip distance* $\text{dist}_S(u, v)$ between u and v.

To warm up to the idea of skip distance, we conclude this subsection with the following lemma.

Lemma 9. *Let h be the distance and s the skip distance to v from an anchor node A. Then, in one dimension,*

$$\lfloor h/2 \rfloor \leqslant s \leqslant h - 1. \tag{1}$$

Proof. If there is exactly one path from A to v, $P = A x_1 \dots x_{h-1} v$, and assuming for simplicity that h is even, then $SP = A x_2 x_4 \dots x_{h-2} v$ is the longest skip path. Any additional nodes can only lengthen *SP*. A maximal skip path can only start (past *A*) at a node $u_2 \in D_2$ and then there is at most one node $u_j \in D_j$ in the skip path for a total of $h - 1$ nodes from u_2 to $u_h = v$. □

5.2. The algorithm

The HS algorithm is depicted below. To start the algorithm, an anchor node *A* transmits the message `(pos(A),1)∘(A,0)`.

```
1:  **hops** := $\infty$; **skips** := $-1$
2:  upon receipt of (pos(A),h)∘(u,s) from **x** do
3:  **if** (**h** = **hops**+1) **then**
4:      **if** (**u** = **x**) **then**
5:          transmit (pos(A),h)∘(u,s)
6:      **else if** (**u** $\notin N(v)$ and **s**+1 $\geqslant$ **skips**) **then**
```

```
7:          transmit (pos(A),h)∘(v,s+1)
8:          skips := s+1
9:       end if
10:  else if (h < hops or (h = hops and
11:          ((u ∈ $N(v)$ and s ⩾ skips) or (u ∉ $N(v)$ and s+1 ⩾ skips)))) then
12:      hops := h
13:      if (u ∈ $N(v)$) then
14:          transmit (pos(A),h+1)∘(u,s)
15:          skips := s
16:      else
17:          transmit (pos(A),h+1)∘(v,s+1)
18:          skips := s+1
19:      end if
20:  end if
```

Theorem 10. *In one dimension, the* HS *algorithm finds the graph and skip distances, h and s, respectively, from an anchor node A to a node v.*

Proof. Observe that the basic structure of the HOP algorithm is kept (Lines 1–2, 10–12, 5, 7, 14, 17) 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 and skip distance s will eventually know its correct hop and skip count. As in Lemma 3 we know that all D_1 nodes will eventually hear the message from A, setting their skip count to 0.

Going from $h - 1 \to h$, we assume that all D_{h-1} nodes will know their correct distance and skip distance. By Lemma 3, we know that then the D_h nodes will learn their distance as well. Based on that, we claim that the D_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.

First, observe that we can ignore all **skips** values *before* the time that a node receives the correct **hops** value since at that point **skips** is set to the sent value (Lines 15 and 18). Since we know that the nodes in D_{h-1} and D_h eventually obtain their correct 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, illegally incrementing itself in the process. Line 10 prevents v from even considering messages from nodes with equal or higher hop count. Line 3 allows messages from same-hop nodes. Observe that all nodes in D_h are neighbors, otherwise they would be farther or closer away from A. We have to distinguish the two possibilities in the **if** statement. If v forwards the message in Line 5, then any receivers in D_h will ignore the message (since $v \neq u$) and only the legitimate receivers in D_{h+1} consider it. If, on the other hand, v has updated its counter (legally, since u is at distance $< h$) (Lines 7, 8) and subsequently another node $w \in D_h$ has picked it up, then we are back at Line 5 and all the next nodes in D_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). Say v's skip distance is in fact s. There are necessarily two more nodes involved, namely, $u \notin N(v)$ at skip distance $s-1$ and a node w *between* them. In order for v to have skip distance s, all such u's must at some point send out $msg_1 =$ `(u,s-1)` or $msg_2 =$ `(x,s-1)` (or both) (Lines 6, 10 guarantee that they pass on this information even if their **skips** counter is already set correctly). By virtue of v being at skip distance s, either there is a msg_1 which w will pass on (Line 5 or 15) as it is (since $u \in N(w)$) and v updates **skips** (Line 8 or 18); or there is a msg_2 and $x \notin N(w)$, upon which w sends `(w,s)` in Line 7 or 17. Observe that if w is also in D_h, then v will not set its counter in Line 5 (to prevent higher skip distance neighbors from wrongly influencing it). In that case, however, we encounter the last possibility: v must have some node $z \in D_{h-1}$ as a neighbor (since v's true skip distance is in fact s), which will also have heard msg_2 (since $w \in D_h$ node already did) and—if z has not incremented the counter—passed it on to v, at which point it will correctly be in Line 18 and update its counters (since now $x \notin N(v)$ given that $x \notin N(w)$). This corresponds to having z as the intermediary node with $x \in N(z)$ but $x \notin N(v)$. □

The following theorem indicates that the time complexity for the improved HS algorithm has not increased significantly over the simple HOP algorithm.

Theorem 11. *After time* $O(h)$, *a node* v *at distance h has received a message with the correct hop and skip count in the one-dimensional* HS *algorithm. In an asynchronous model, for any edge e, the maximum number of messages exchanged on e is* $O(n + sp)$, *where n is the number of nodes and sp the number of shortest paths from A to* v. *In a synchronous model, message complexity is in* $O(1)$ *but with increased message size by a factor of sp.*

Proof. For the time complexity, we will again lean on our analysis of the simple HOP algorithm of Lemma 4. Let the maximal time unit be 1. We claim that by time $2h$, all nodes in D_h will have their correct **hops** and **skips** values. By time 1, the D_1 nodes will learn their hop as well as skip count from node A. Assume now that the nodes in D_k for $k < h$ have been informed of all their correct values by time $T(h-1) = 2(h-1)$. Then by time $T(h-1)+1$, all $v \in D_h$ will have received their hop and some skip value from *all* $u \in D_{h-1} \cap N(v)$. Since all D_{h-1} nodes have already obtained their correct skip distance, it takes at most two more time steps from $T(h-1)$ for $v \in D_h$ to learn (all of) its skip distance(s): (a) a message from some $u \in D_{h-1}$ to another $w \in D_{h-1}$ to v; (b) from $u \in D_{h-1}$ to $w \in D_h$ to v; or (c) directly from $u \in D_{h-1}$ to v. Thus $T(h) = T(h-1) + 2 = 2h$. □

For the message complexity, we have to first count the number of messages until v obtains its hop count, which is, as for the hop algorithm, in $O(n)$. Additionally, v's neighbors have to forward potentially information about all shortest paths to determine which one has the highest skip distance at v. In the synchronous case, a node can first bundle all the messages with the smallest hop count and send it off as one packet.

The interval for v is now bounded by $s < \mathrm{dist}_E(A, v) \leqslant h$ and the position is again computed as the mid-point of the intersection of all such intervals.

5.3. 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 12. *Take a one-dimensional unit disk graph. Assume there is only one anchor node and all nodes know they are to its right. For the position of a node v as determined by the* HS *algorithm, we have*

$$MaxErr(pos_{\mathrm{HS}}(v)) \leqslant MaxErr(pos_{\mathrm{OPT}}(v)) + \varepsilon$$

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

In order to prove Lemma 12, we will need the following two lemmas.

Lemma 13. *If a node v is distance h from an anchor node A at* 0, *then it is possible to construct a one-dimensional UDG based on $G = (V, E)$ such that $pos(v) = h - \varepsilon$ for some arbitrarily small $\varepsilon > 0$.*

Proof. Let the origin of our one-dimensional coordinate axis be at A (i.e. $pos(A) = 0$), increasing to the right. Consider stretching the graph G to its maximum possible position at v. We will use induction on the number of hops h from the anchor node A at $pos(A) = 0$ to v. Let $D_h = \{v_0^h, \ldots, v_{n_h}^h\}$. Let the ordering be such that, in their actual positions, we have (setting $v = pos(v)$ for readability) $v_{n_h}^h \leqslant \cdots \leqslant v_1^h \leqslant v_0^h$, (i.e., v_0^h is the rightmost node in D_h). Observe that all nodes in D_h are neighbors, otherwise they would have a different distance to A. Furthermore, 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 $v_{n_h}^h < \cdots < v_1^h < v_0^h$.

For $h = 1$ place the n_1 (different) nodes at positions $pos(v_i^1) = 1 - i \cdot \varepsilon$ for some sufficiently small $1 \gg \varepsilon > 0$ (i.e., $\varepsilon_{i,1} = i \cdot \varepsilon$). Then $\varepsilon_{i+1,1} - \varepsilon_{i,1} = \varepsilon$. (Note that this is not the same ε as in the lemma.)

Assume now that we have placed all nodes within $h - 1$ hops such that $pos(v_i^{h-1}) = (h-1) - \varepsilon_{i,h-1}$, and, with the labeling from above, $\varepsilon_{i,h-1} < \varepsilon_{i+1,h-1}$ for all appropriate i. Then, For each of the v_i^h we consider the maximal (leftmost) j for which $v_j^{h-1} \in N(v_i^h)$ and $v_{j+1}^{h-1} \notin N(v_i^h)$. Now put

$$pos(v_i^h) = pos(v_j^{h-1}) + 1 = h - \varepsilon_{j,h-1} \tag{2}$$

and set $\varepsilon_{i,h} = \varepsilon_{j,h-1}$. It remains to be shown that all the neighbors of node $v = v_i^h$ in G are within distance 1 (and only those). Note that $|v_i^h - v_l^h| = |\varepsilon_{i,h} - \varepsilon_{l,h}| < 1$ by choosing the initial ε's sufficiently small. For nodes v_l^{h-1} in D_{h-1}, we have

$$\Delta_l := pos(v) - pos(v_l^{h-1}) = 1 - (\varepsilon_{j,h-1} - \varepsilon_{l,h-1})$$

and, for readability, set $\varepsilon_i := \varepsilon_{i,h-1}$. Then $\varepsilon_j - \varepsilon_l \geqslant 0$ for $l \leqslant j$, thus $\Delta_l \leqslant 1$ for $v_l^{h-1} \in N(v)$, by construction. Similarly, $\varepsilon_j - \varepsilon_l < 0$ when $l > j$, thus $\Delta_l > 1$ for $v_l^{h-1} \notin N(v)$. $\square$

Lemma 14. *If a node v has skip distance s from an anchor node A at* 0, *then it is possible to construct a one-dimensional UDG based on $G = (V, E)$ such that $pos(v) = s + \varepsilon$ for some arbitrarily small $\varepsilon > 0$.*

Proof. We will again proceed by induction, this time on the number of skips s. The notation is adapted from the proof of Lemma 13, i.e. $v_0^h > v_1^h > \cdots > v_{n_h}^h$ represent the different nodes at h hops. Recall that all v_i^h are neighbors for the same h. Analogously, let $w_0^s > w_1^s > \cdots > w_{n_s}^s$ represent the different nodes at s skips. Observe that their hop counts differ by at most one, and they are all neighbors as well. (Otherwise there would be a skip from $w_{n_s}^s$ to w_0^s.)

Apart from A, all $v_i^1 = w_i^0$ and we can place them at $pos(w_i^0) = (n_0 - i + 1)\varepsilon$ and $\varepsilon_{i+1,0} - \varepsilon_{i,0} = \varepsilon$. Again, for a sufficiently small $0 < \varepsilon \ll 1$, all nodes are within Euclidean distance 1.

By induction hypothesis, we have that $pos(w_i^{s-1}) = (s-1) + \varepsilon_{i,s-1}$, where $\varepsilon_{i,s-1} - \varepsilon_{i+1,s-1} \geqslant \varepsilon > 0$. By definition, every w_i^s has a minimal (rightmost) j for which $w_j^{s-1} \notin N(w_i^s)$. Thus we set

$$pos(w_i^s) = pos(w_j^{s-1}) + 1 + \delta = s + (\varepsilon_{j,s-1} + \delta) \tag{3}$$

for some $0 < \delta < \varepsilon$ and we will argue that we can satisfy all the neighboring requirements for $w = w_i^s$. Note that, again, all nodes with the same skip distance are within one unit: $|pos(w_i^s) - pos(w_l^s)| = |\varepsilon_{i,s} - \varepsilon_{l,s}| < 1$ (choosing the initial ε sufficiently small). For the remaining nodes we have

$$\begin{aligned}\Delta_l &= pos(w) - pos(w_l^{s-1}),\\ &= 1 + (\varepsilon_{j,s-1} - \varepsilon_{l,s-1}) + \delta,\end{aligned}$$

and set $\varepsilon_i := \varepsilon_{i,s-1}$. Since now $\varepsilon_j - \varepsilon_l \geqslant 0$ for $l \geqslant j$, it follows that $\Delta_l \geqslant 1 + \delta > 1$ for $w_l^{s-1} \notin N(w)$. Similarly, $\varepsilon_j - \varepsilon_l \leqslant -\varepsilon$ for $l < j$ and $\delta < \varepsilon$, thus $\Delta_l \leqslant 1 + \delta - \varepsilon < 1$ when $w_l^{s-1} \in N(w)$. $\square$

We are now ready for the proof of Lemma 12.

Proof (*of Lemma 12*). Given the knowledge of the entire graph structure $G = (V, E)$, we can construct two instances of a UDG(G) where v's true position is $h - \varepsilon_1$ at the maximum and $s + \varepsilon_2$ at the minimum ($\varepsilon_i > 0$). Therefore, the optimal algorithm cannot distinguish between these extremes and is thus forced to return $pos(v) \approx (h-s)/2$ which is the position returned by HS who knows only h and s. $\square$

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

Theorem 15. HS *is optimal in one dimension up to an additive constant.*

Proof. We have shown in Lemma 12 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 more general scenario.

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 12, 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 that may 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. Since this happens locally in v's neighborhood, one could improve HS so as to eliminate this potential adjustment.

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 $pos(A_1) > \cdots > pos(A_l)$ to the left of v, where again the coordinates increase to the right. Let $h_i = \mathrm{dist}_G(A_i, v)$ and $s_i = \mathrm{dist}_S(A_i, v)$. Set

$$L_i = A_i + s_i, \quad R_i = A_i + h_i,$$

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

$$L = \max_i L_i, \quad R = \min_i R_i,$$

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.

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 $d_i = \mathrm{dist}_E(A_i, A_1)$ and $g_i = \mathrm{dist}_G(A_i, A_1)$. Then $h_i \geqslant g_i + h_1 - 1$ 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 $A_i = A_1 - d_i$, we get

$$\begin{aligned} R_i &\geqslant A_i + g_i + h_1 - 1 \\ &= (A_1 + h_1) + \underbrace{g_i - d_i}_{\geqslant 0} - 1 \\ &\geqslant R_1 - 1, \end{aligned}$$

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.

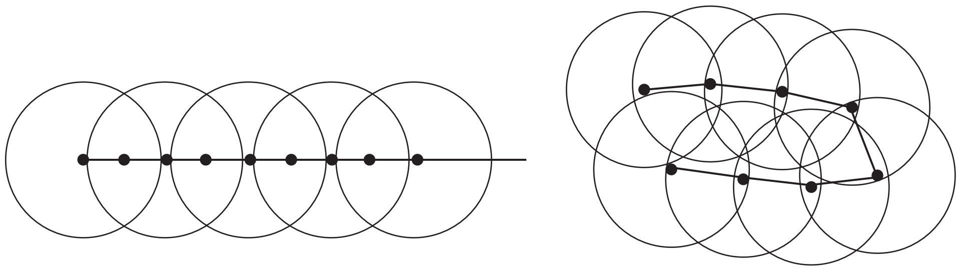


Fig. 2. Minimum Euclidean distance between two nodes given their hop distance *h*. The circles have radius 1, the dots are the nodes. On the left is the one-dimensional case where $\text{dist}_E(\cdot,\cdot) > h/2$ (not all circles are shown). On the right, in two or more dimensions, only $\text{dist}_E(\cdot,\cdot) > 1$ can be assumed.

The case for the left boundary is similar. Here, let $t_i = \text{dist}_S(A_i, A_1)$. Then

$$\begin{aligned} L_i &\leqslant A_i + t_i + s_1 + 1 \\ &= (A_1 + s_1) + \underbrace{t_i - d_i}_{\leqslant 0} + 1 \\ &\leqslant L_1 + 1, \end{aligned}$$

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 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 bigger by at most 2 in the general case, proving Theorem 15. □

6. The GHoST algorithm

We now move on to higher dimensions. From Section 4.3 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 Fig. 2. 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.

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

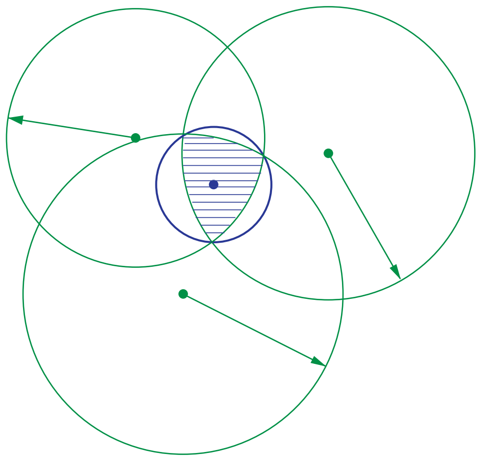


Fig. 3. Construction of the "mid point'' of a two-dimensional area. The outer circles of radius h_i (in green) represent the reach of each anchor (being h_i hops away). Their intersection is shaded in the center. The smallest circle enclosing the entire intersection area is depicted in the center (in blue). The center is the computed position.

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 Fig. 3. 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.

6.1. Lessons learned from one dimension

The crucial insight of the one-dimensional optimal HS algorithm was that there exist certain local structures in the unit disk graph (e.g. a skip) from 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 which one as follow:

- 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 $\{v, x\} \notin E$, $\mathrm{dist}_{\mathrm{E}}(u, w) \leqslant \sqrt{3}$. See Fig. 4.
- T_k—a generalization of T_0: 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 $\{v_0, x_0\}, \{v_k, x_k\} \notin E$. 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 $\mathrm{dist}_{\mathrm{E}}(u, v) \leqslant k + \sqrt{3}$ as opposed to $k + 2$.

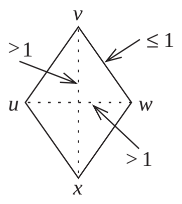


Fig. 4. A trimmer for the path from u to w (and from x to v). The dashed lines indicate that there is *no* connection. With a simple geometric argument one can impose a maximum length on the distance of u to w.

- MT_{k_1,k_2}—a trimmer resulting from the merging of two paths from two different anchors. As an exemplary case, consider two paths from anchors A_1 and A_2 that merge after just one hop at node m ($MT_{1,1}$). 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 $\mathrm{dist}_{\mathrm{E}}(A_i, v) \leqslant \sqrt{1+(h-1)^2} = \sqrt{h^2-2(h-1)} < h$. The constant adjustment accounts for the possibility of m being in the opposite direction of v with respect to the A_i. An analogous case can be made if the paths merge at m after k_1 hops from A_1 and k_2 hops from A_2.

6.2. The algorithm

Based on the arguments of Section 6.1, 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 D_{h-1} obtained their correct paths of length $h-1$, they send it on to nodes in D_h. In one time unit, v receives all those transmissions from neighboring nodes u in D_{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.

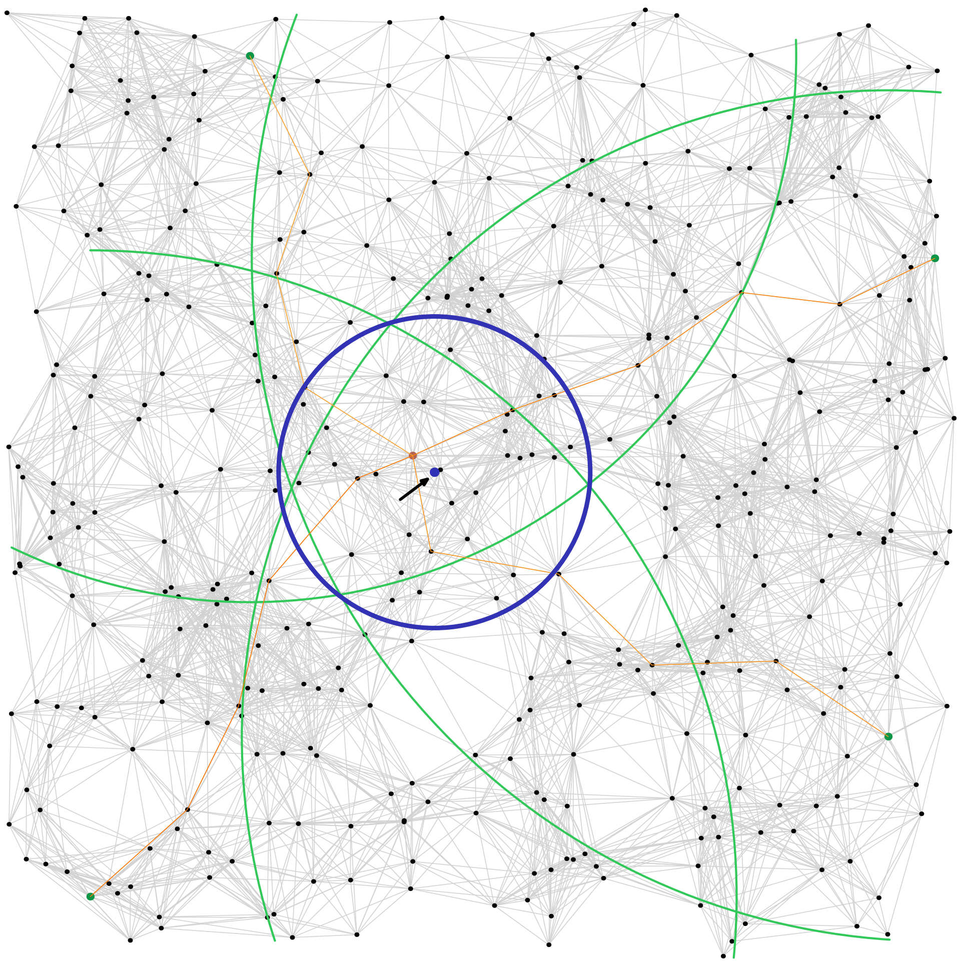
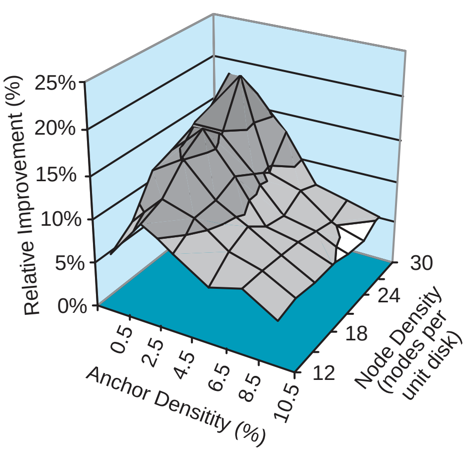


Fig. 5. The visualization of the GHoST algorithm. The intersection of circles in the center is the area of all possible positions as calculated by the algorithm. The center of the circle (marked by the arrow) is chosen as the computed position which minimizes the maximum error.

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.

Altogether, with the remarks of this section, we can conclude the following.

Theorem 16. *In two dimensions, the* GHoST *algorithm with trimmers T_k has less or equal MaxErr(v) as* HOP *(for all nodes v) and has the same time complexity* $O(h)$ *as* HOP (*where h is the graph distance from an anchor node to the node in question*).



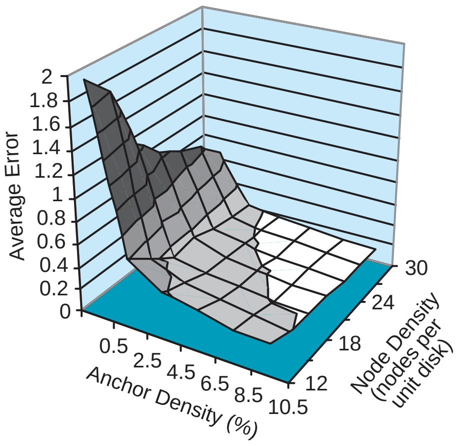


Fig. 6. The graph on the left shows the improvement of GHoST over hops in the depicted anchor and node density ranges. The graph on the right shows the absolute errors of GHoST (in units of the radio range).

6.3. Simulation

The trimmers of Section 6.1 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 4.2 and the GHOST algorithm with the trimmer T_0 only. A screen-shot of the visual part of the application can be seen in Fig. 5. 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 hosts). For each combination we collect 300 position estimates along with the error for both HOP and GHOST.

Since we are interested in the influence of T_0 on HOP, we calculate the *average relative errors* of GHOST to HOP in Fig. 6). The absolute errors of GHOST can also be seen in Fig. 6 (right). 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).

7. Conclusions

Our goal is to understand the fundamentals of positioning, without relying on heuristics. For a simplified one-dimensional model, we manage to present an algorithm which solves positioning optimally. In contrast, we show that the computationally equivalent hop-based algorithm does not render competitive results.

In the second part of the paper, we showed how to apply the underlying ideas from the optimal one-dimensional algorithm to improve hop-based algorithms. Our main focus was on fast distributed algorithms with worst-case guarantees. The simulations then show that such a worst-case approach also yields promising improvements in average scenarios as well. In addition, GHOST can be substituted into more sophisticated algorithms where now only a hop-based approach is used.

References

[1] J. Aspnes, D. Goldenberg, Y. Yang, On the computational complexity of sensor network localization, in: Proc. of ALGOSENSORS, 2004.

[2] P. Bahl, V.N. Padmanabhan, RADAR: An in-building RF-based user location and tracking system, in: Proc. of Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM), 2000.

[3] J. Beutel, Geolocation in a pico radio environment, Master's thesis, ETH Zurich, EE Laboratory, December 1999.

[4] J. Beutel, Location management in wireless sensor networks, Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press, Boca Raton, 2004.

[5] P. Biswas, Y. Ye, Semidefinite programming for ad hoc wireless sensor network localization, in: Proc. of Internat. Workshop on Information Processing in Sensor Networks (IPSN), 2004.

[6] P. Bose, P. Morin, I. Stojmenovic, J. Urrutia, Routing with guaranteed delivery in ad hoc wireless networks, in: Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), 1999.

[7] S. Capkun, M. Hamdi, J.-P. Hubaux, GPS-free positioning in mobile ad-hoc networks, in: Proc. of Hawaii Internat. Conf. on System Sciences (HICSS), 2001.

[8] L. Doherty, K. Pister, L.E. Ghaoui, Convex optimization methods for sensor node position estimation, in: Proc. of Joint Conf. the IEEE Computer and Communications Societies (INFOCOM), 2001.

[9] T. He, C. Huang, B. Blum, J. Stankovic, T. Abdelzaher, Range-free localization schemes in large scale sensor networks, in: Proc. of Mobile Computing and Networking (MOBICOM), 2003.

[10] B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins, Global Positioning Systems: Theory and Practice, Fifth Edition, Springer, Berlin, 2001.

[11] B. Karp, H. Kung, GPSR: Greedy perimeter stateless routing for wireless networks, in: Proc. of Mobile Computing and Networking (MOBICOM), 2000.

[12] L. Kleinrock, J. Silvester, Optimum transmission radii for packet radio networks or why six is a magic number, in: Nat'l Telecommunications Conference, 1978.

[13] F. Kuhn, T. Moscibroda, R. Wattenhofer, Unit disk graph approximation, in: Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), 2004.

[14] F. Kuhn, R. Wattenhofer, Y. Zhang, A. Zollinger, Geometric ad-hoc routing: Of theory and practice, in: Proc. of Symp. on Principles of Distributed Computing (PODC), 2003.

[15] F. Kuhn, R. Wattenhofer, A. Zollinger, Worst-case optimal and average-case efficient geometric ad-hoc routing, in: Proc. of Internat. Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC), 2003.

[16] Z. Lotker, M.M. de Albeniz, S. Perennes, Range-Free Ranking in Sensors Networks and Its Applications to Localization, in: Proc. of Ad-Hoc, Mobile, and Wireless Networks: Third International Conference (ADHOC-NOW), 2004.

[17] T. Moscibroda, R. O'Dell, M. Wattenhofer, R. Wattenhofer, Virtual coordinates for ad hoc and sensor networks, in: Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), 2004.

[18] R. Nagpal, H. Shrobe, J. Bachrach, Organizing a global coordinate system from local information on an ad hoc sensor network, in: Proc. of Information Processing in Sensor Networks (IPSN), 2003.

[19] D. Niculescu, B. Nath, Ad hoc positioning system (APS), in: Proc. of IEEE Global Communications (GLOBECOM), 2001.

[20] D. Niculescu, B. Nath, Ad hoc positioning system (APS) using AoA, in: Proc. of Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2003.
[21] D. Niculescu, B. Nath, Error characteristics of ad hoc positioning systems, in: Proc. of Internat. Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC), 2004.
[22] D. Niculescu, B. Nath, DV based positioning in ad hoc networks, J. Telecomm. Syst. 22(1–4) (2003).
[23] A. Rao, C. Papadimitriou, S. Ratnasamy, S. Shenker, I. Stoica, Geographic routing without location information, in: Proc. of Mobile Computing Networking (MOBICOM), 2003.
[24] C. Savarese, J. Rabaey, K. Langendoen, Robust positioning algorithms for distributed ad-hoc wireless sensor networks, in: Proc. of USENIX Technical Conference, 2002.
[25] A. Savvides, C.-C. Han, M. Srivastava, Dynamic fine-grained localization in ad-hoc networks of sensors, in: Proc. of Mobile Computing and Networking (MOBICOM), 2001.
[26] Y. Shang, W. Ruml, Y. Zhang, M. Fromherz, Localization from mere connectivity, in: Proc. of Internat. Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC), 2003.
[27] H. Takagi, L. Kleinrock, Optimal transmission ranges for randomly distributed packet radio terminals, IEEE Trans. Comm. 32(3) (1984).