Querying Probabilistic Neighborhoods in Spatial Data Sets Efficiently

Moritz von Looz and Henning Meyerhenke

{moritz.looz-corswarem, meyerhenke}@kit.edu
Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany

Abstract. The probability that two spatial objects establish some kind of mutual connection often depends on their proximity. To formalize this concept, we define the notion of a *probabilistic neighborhood*: Let P be a set of n points in $\mathbb{R}^d$, $q \in \mathbb{R}^d$ a query point, dist a distance metric, and $f : \mathbb{R}^+ \to [0, 1]$ a monotonically decreasing function. Then, the probabilistic neighborhood $N(q, f)$ of q with respect to f is a random subset of P and each point $p \in P$ belongs to $N(q, f)$ with probability $f(\text{dist}(p, q))$. Possible applications include query sampling and the simulation of probabilistic spreading phenomena, as well as other scenarios where the probability of a connection between two entities decreases with their distance. We present a fast, sublinear-time query algorithm to sample probabilistic neighborhoods from planar point sets. For certain distributions of planar P, we prove that our algorithm answers a query in $O((|N(q, f)| + \sqrt{n}) \log n)$ time with high probability. In experiments this yields a speedup over pairwise distance probing of at least one order of magnitude, even for rather small data sets with $n = 10^5$ and also for other point distributions not covered by the theoretical results.

1 Introduction

In many scenarios, connections between spatial objects are not certain but probabilistic, with the probability depending on the distance between them: The probability that a customer shops at a certain physical store shrinks with increasing distance to it. In disease simulations, if the social interaction graph is unknown but locations are available, disease transmission can be modeled as a random process with infection risk decreasing with distance. Moreover, the wireless connections between units in an ad-hoc network are fragile and collapse more frequently with higher distance.

For these and similar scenarios, we define the notion of a *probabilistic neighborhood* in spatial data sets: Let a set P of n points in $\mathbb{R}^d$, a query point $q \in \mathbb{R}^d$, a distance metric dist, and a monotonically decreasing function $f : \mathbb{R}^+ \to [0, 1]$ be given. Then, the probabilistic neighborhood $N(q, f)$ of q with respect to f is a random subset of P and each point $p \in P$ belongs to $N(q, f)$ with probability $f(\text{dist}(p, q))$. A straightforward query algorithm for sampling a probabilistic neighborhood would iterate over each point $p \in P$ and sample for each whether it is included in $\mathrm{N}(q, f)$. This has a running time of $\Theta(n \cdot d)$ per query point, which

is prohibitive for repeated queries in large data sets. Thus we are interested in a faster algorithm for such a *probabilistic neighborhood query* (PNQ, spoken as "pink"). We restrict ourselves to the planar case in this work, but the algorithmic principle is generalizable to higher dimensions.

While the linear-time approach has appeared before in the literature for a particular application [2] (without formulating the problem as a PNQ explicitly), we are not aware of previous work performing more efficient PNQs with an index structure. For example, the probabilistic quadtree introduced by Kraetzschmar et al. [10] is designed to store probabilistic occupancy data and gives deterministic results. Other range queries related to (yet different from) our work as well as deterministic index structures are described in Section 2.2. Proofs, details, further experiments, pseudocode and visualizations omitted due to space constraints can be found in the full version of this paper [16].

Contributions. We develop, analyze, implement, and evaluate an index structure and a query algorithm that together provide fast probabilistic neighborhood queries in the Euclidean and hyperbolic plane. Our key data structure for these fast PNQs is a polar quadtree which we adapt from our previous work [17]. Preprocessing for quadtree construction requires $O(n \log n)$ time with high probability[1] (whp).

To answer PNQs, we first present a simple query algorithm (Section 3). We then improve its time complexity by treating whole subtrees as so-called virtual leaves, see Section 4. As shown by our detailed theoretical analysis, the improved algorithm yields a query time complexity of $O((|\mathrm{N}(q,f)| + \sqrt{n}) \log n)$ whp to find a probabilistic neighborhood $\mathrm{N}(q,f)$ among n points, for n sufficiently large. This is sublinear if the returned neighborhood $\mathrm{N}(q,f)$ is of size $o(n/\log n)$ – an assumption we consider reasonable for most applications. For our theoretical results to hold, the quadtree structure needs to be able to partition the distribution of the point positions in P, i. e. not all of the probability mass may be concentrated on a single point or line. In our case of polar quadtrees, this is achieved if the distribution is continuous, integrable, rotationally invariant with respect to the origin and non-zero only for a finite area.

Experimental results are shown in Section 5: We apply our query algorithm to generate random graphs in the hyperbolic plane [12] in subquadratic time. Graphs with millions of edges can now be generated within a few minutes sequentially. This yields an acceleration of at least one order of magnitude in practice compared to a reference implementation [2] that uses linear-time queries. Compared to our previous work on graph generation [17], our new algorithm is able to generate a more extensive model. Even if the distribution of a given point set P is unknown in practice, running times are fast: As an example of probabilistic spreading behavior, we simulate a simple disease spreading mechanism on real population density geodata. In this scenario, our fast PNQs are at least two orders of magnitude faster than linear-time queries.

[1] We say "with high probability" (whp) when referring to a probability $\geq 1 - 1/n$ for sufficiently large n.

2 Preliminaries

2.1 Notation

Let the input be given as set P of n points. The points in P are distributed in a disk $\mathbb{D}_R$ of radius R in the hyperbolic or Euclidean plane, the distribution is given by a probability density function $j(\phi, r)$ for an angle ϕ and a radius r. Recall that, for our theoretical results to hold, we require j to be known, continuous and integrable. Furthermore, j needs to be rotationally invariant – meaning that $j(\phi_1, r) = j(\phi_2, r)$ for any radius r and any two angles ϕ_1 and ϕ_2 – and positive within $\mathbb{D}_R$, so that $j(r) > 0 \Leftrightarrow r < R$. Due to the rotational invariance, $j(\phi, r)$ is the same for every ϕ and we can write $j(r)$. Likewise, we define $J(r)$ as the indefinite integral of $j(r)$ and normalize it so that $J(R) = 1$ (also implying $J(0) = 0$). The value $J(r)$ then gives the fraction of probability mass inside radius r.

For the distance between two points p_1 and p_2, we use $\text{dist}_{\mathbb{H}}(p_1, p_2)$ for the hyperbolic and $\text{dist}_{\mathbb{E}}(p_1, p_2)$ for the Euclidean case. We may omit the index if a distinction is unnecessary. As mentioned, a point p is in the probabilistic neighborhood of query point q with probability $f(\text{dist}(p, q))$. Thus, a *query pair* consists of a query point q and a function $f : \mathbb{R}^+ \to [0, 1]$ that maps distances to probabilities. The function f needs to be monotonically decreasing but may be discontinuous. Note that f can be defined differently for each query. The query result, the probabilistic neighborhood of q w. r. t. f, is denoted by the set $\text{N}(q, f) \subseteq P$.

For the algorithm analysis, we use two additional sets for each query (q, f):

- $\text{Candidates}(q, f)$: neighbor candidates examined when executing such a query,
- $\text{Cells}(q, f)$: quadtree cells examined during execution of the query.

Note that the sets $\text{N}(q, f), \text{Candidates}(q, f)$ and $\text{Cells}(q, f)$ are probabilistic, thus theoretical results about their size are usually only with high probability.

2.2 Related Work

Fast deterministic range queries. Numerous index structures for fast range queries on spatial data exist. Many such index structures are based on trees or variations thereof, see Samet's book [14] for a comprehensive overview. I/O efficient worst case analysis is usually performed using the EM model, see e. g. [3]. In more applied settings, average-case performance is of higher importance, which popularized R-trees or newer variants thereof, e. g. [9]. Concerning (balanced) quadtrees for spatial dimension d, it is known that queries require $O(d \cdot n^{1-1/d})$ time (thus $O(\sqrt{n})$ in the planar case) [14, Ch. 1.4]. Regarding PNQs our algorithm matches this query complexity up to a logarithmic factor. Yet note that, since for general f and dist in our scenario all points in the set P could be neighbors, data structures for deterministic queries cannot solve a PNQ efficiently without adaptations.

Hu et al. [8] give a query sampling algorithm for one-dimensional data that, given a set P of n points in $\mathbb{R}$, an interval $q = [x, y]$ and an integer, $t \geq 1$, returns t elements uniformly sampled from $P \cap q$. They describe a structure of $O(n)$ space that answers a query in $O(\log n + t)$ time and supports updates in $O(\log n)$ time. While also offering query sampling, PNQs differ from the problem considered by Hu et al. in two aspects: We consider two dimensions instead of one and our sampling probabilities are not necessarily uniform, but can be set by the user by a distance-dependent function.

Range queries on uncertain data. During the previous decade probabilistic queries *different* from PNQs have become popular. The main scenarios can be put into two categories [13]: (i) Probabilistic databases contain entries that come with a specified confidence (e. g. sensor data whose accuracy is uncertain) and (ii) objects with an uncertain location, i. e. the location is specified by a probability distribution. Both scenarios differ under typical and reasonable assumptions from ours: Queries for uncertain data are usually formulated to return *all* points in the neighborhood whose confidence/probability exceeds a certain threshold [11], or computing points that are possibly nearest neighbors [1].

In our model, in turn, the choice of inclusion of a point p is a random choice for every different p. In particular, depending on the probability distribution, *all* nodes in the plane can have positive probability to be part of some other's neighborhood. In the related scenarios this would only be true with extremely small confidence values or extremely large query circles.

Applications in fast graph generation. One application for PNQs as introduced in Section 1 is the hyperbolic random graph model by Krioukov et al. [12]. The n graph nodes are represented by points thrown into the hyperbolic plane at random[2] and two nodes are connected by an edge with a probability that decreases with the distance between them. An implementation of this generative model is available [2], it performs $\Theta(n^2)$ neighborhood tests. Bringmann et al. provide an algorithm to generate hyperbolic random graphs in expected linear time [5]; to our knowledge no implementation of it exists yet.

In previous work we designed a generator [17] faster than [2] for a restricted model; it runs in $O((n^{3/2} + m) \log n)$ time whp for the whole graph with m edges. The range queries discussed there are facilitated by a quadtree which supports only deterministic queries. Consequently, the queries result in unit-disk graphs in the hyperbolic plane and can be considered as a special case of the current work (a step function f with values 0 and 1 results in a deterministic query).

Our major technical inspiration for enhancing the quadtree for probabilistic neighborhoods is the work of Batagelj and Brandes [4]. They were the first to present a random sampling method to generate Erdős-Rényi-graphs with n nodes and m edges in $O(n + m)$ time complexity. Faced with a similar problem of selecting each of n elements with a constant probability p, they designed an

[2] The probability density in the polar model depends only on radii r and R as well as a growth parameter α and is given by $g(r) := \alpha \frac{\sinh(\alpha r)}{\cosh(\alpha R) - 1}$.

efficient algorithm. Instead of sampling each element separately, they use random jumps of length $\delta(p)$, $\delta(p) = \ln(1 - rand)/\ln(1 - p)$, with *rand* being a random number uniformly distributed in $[0, 1)$.

2.3 Quadtree Specifics

Our key data structure is a polar region quadtree in the Euclidean or hyperbolic plane. While they are less suited to higher dimensions as for example k-d-trees, the complexity is comparable in the plane. For the (circular) range queries we discuss, quadtrees have the significant advantage of a bounded aspect ratio: A cell in a k-d-tree might extend arbitrarily far in one direction, rendering theoretical guarantees about the area affected by the query circle difficult to impossible. In contrast, the region covered by a quadtree cell is determined by its position and level.

We mostly reuse our previous definition [17] of the quadtree: A node in the quadtree is defined as a tuple $(\min_\phi, \max_\phi, \min_r, \max_r)$ with $\min_\phi \leq \max_\phi$ and $\min_r \leq \max_r$. It is responsible for a point $p = (\phi_p, r_p)$ exactly if $(\min_\phi \leq \phi_p < \max_\phi)$ and $(\min_r \leq r_p < \max_r)$. We call the region represented by a particular quadtree node its quadtree *cell*. The quadtree is parametrized by its radius R, the $\max_r$ of the root cell. If the probability distribution j is known (which we assume for our theoretical results), we set the radius R to $\arg\min_r J(r) = 1$, i. e. to the minimum radius that contains the full probability mass. If only the points are known, the radius is set to include all of them. While in this latter case the complexity analysis of Section 3 and 4 does not hold, fast running times in practice can still be achieved (see Section 5).

3 Baseline Query Algorithm

We begin the main technical part by describing adaptations in the quadtree construction as well as a baseline query algorithm. This latter algorithm introduces the main idea, but is asymptotically not faster than the straightforward approach. In Section 4 it is then refined to support faster queries.

3.1 Quadtree Construction

At each quadtree node v, we store the size of the subtree rooted there. We then generalize the rule for node splitting to handle point distributions j as defined in Section 2.1: As is usual for quadtrees, a leaf cell c is split into four children when it exceeds its fixed capacity. Since our quadtree is polar, this split happens once in the angular and once in the radial direction. Due to the rotational symmetry of j, splitting in the angular direction is straightforward as the angle range is halved: $\text{mid}_\phi := \frac{\max_\phi + \min_\phi}{2}$. For the radial direction, we choose the splitting radius to result in an equal division of probability mass. The total probability mass in a ring delimited by $\min_r$ and $\max_r$ is $J(\max_r) - J(\min_r)$. Since $j(r)$ is positive for r between R and 0, the restricted function $J|_{[0,R]}$ defined above is a

bijection. The inverse $(J|_{[0,R]})^{-1}$ thus exists and we set the splitting radius mid_r to $(J|_{[0,R]})^{-1}\left(\frac{J(\max_r)+J(\min_r)}{2}\right)$.

Figure 1 visualizes a point distribution on a hyperbolic disk with 200 points and Figure 2 its corresponding quadtree.

Two results on quadtree properties help to establish the time complexity of quadtree operations. They are generalized versions of our previous work [17, Lemmas 1 and 2] and state that each quadtree cell contains the same expected number of points and that the quadtree height is $O(\log n)$ whp.

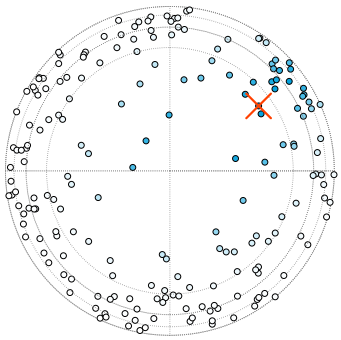

Fig. 1. Query over 200 points in a polar hyperbolic quadtree, with $f(d) := 1/(e^{(d-7.78)}+1)$ and the query point q marked by a red cross. Points are colored according to the probability that they are included in the result. Blue represents a high probability, white a probability of zero.

Lemma 1. *Let $\mathbb{D}_R$ be a hyperbolic or Euclidean disk of radius R, j a probability distribution on $\mathbb{D}_R$ which fulfills the properties defined in Section 2.1, p a point in $\mathbb{D}_R$ which is sampled from j, and T be a polar quadtree on $\mathbb{D}_R$. Let C be a quadtree cell at depth i. Then, the probability that p is in C is 4^{-i}.*

Proposition 1. *Let $\mathbb{D}_R$ and j be as in Lemma 1. Let T be a polar quadtree on $\mathbb{D}_R$ constructed to fit j. Then, for n sufficiently large,* $\text{height}(T) \in O(\log n)$ *whp.*

A direct consequence from the results above and our previous work [17] is the preprocessing time for the quadtree construction. The generalized splitting rule and storing the subtree sizes only change constant factors.

Corollary 1. *Since a point insertion takes $O(\log n)$ time whp, constructing a quadtree on n points distributed as in Section 2.1 takes $O(n \log n)$ time whp.*

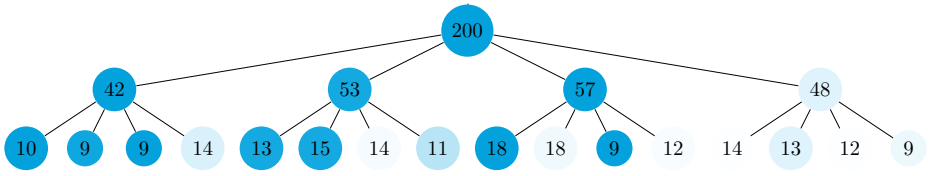

Fig. 2. Visualization of the data structure used in Figure 1. Quadtree nodes are colored according to the upper probability bound for points contained in them. The color of a quadtree node c is the darkest possible shade (dark = high probability) of any point contained in the subtree rooted at c. Each node is marked with the number of points in its subtree.

Algorithm 1: QuadNode.getProbabilisticNeighborhood

Input: query point q, prob. function f, quadtree node c
Output: probabilistic neighborhood of q

```
1  N = {};
2  b_ = dist(q, c);
   /* Distance between point and cell                                  */
3  b¯=f(b_);
   /* Since f is monotonically decreasing, a lower bound for the
      distance gives an upper bound b¯ for the probability.            */
4  s = number of points in c;
5  if c is not leaf then
   |  /* internal node: descend, add recursive result to local set    */
6  |  for child ∈ children(c) do
7  |  |  add getProbabilisticNeighborhood(q, f, child) to N;
8  else
   |  /* leaf case: apply idea of Batagelj and Brandes [4]            */
9  |  for i=0; i < s ; i++ do
10 |  |  δ = ln(1 − rand)/ ln(1 − b¯);
11 |  |  i += δ;
12 |  |  if i ≥ s then
13 |  |  |  break;
14 |  |  prob = f(dist(q, c.points[i]))/b¯;
15 |  |  add c.points[i] to N with probability prob
16 return N
```

3.2 Algorithm

The baseline version of our query (Algorithm 1) has unfortunately a time complexity of $\Theta(n)$, but serves as a foundation for the fast version (Section 4). It takes as input a query point q, a function f and a quadtree cell c. Initially, it is called with the root node of the quadtree and recursively descends the tree. The algorithm returns a point set $\mathrm{N}(q, f) \subseteq P$ with

$$\mathbf{Pr}\,[\,p \in \mathrm{N}(q, f)\,] = f(\mathrm{dist}(q, p)). \tag{1}$$

Algorithm 1 descends the quadtree recursively until it reaches the leaves. Once a leaf l is reached, a lower bound $\underline{b}$ for the distance between the query point q and all the points in l is computed (Line 2). Such distance calculations are detailed in the full version [16]. Since f is monotonically decreasing, this lower bound for the distance gives an upper bound $\overline{b}$ for the probability that a given point in l is a member of the returned point set (Line 3). This bound is used to select *neighbor candidates* in a similar manner as Bategelj and Brandes [4]: In Line 10, a random number of vertices is skipped, so that every vertex in l is selected as a neighbor candidate with probability $\overline{\mathrm{b}}$. The actual distance $\mathrm{dist}(q, a)$ between a candidate a and the query point q is at least $\underline{\mathrm{b}}$ and the probability of $a \in \mathrm{N}(q, f)$ thus at most $\overline{\mathrm{b}}$. For each candidate, this actual distance $\mathrm{dist}(q, a)$

is then calculated and a neighbor candidate is confirmed as a neighbor with probability $f(\mathrm{dist}(q,a))/\overline{\mathrm{b}}$ in Line 14.

Regarding correctness and time complexity of Algorithm 1, we can state:

Proposition 2. *Let T be a quadtree as defined above, q be a query point and $f : \mathbb{R}^+ \to [0,1]$ a monotonically decreasing function which maps distances to probabilities. The probability that a point p is returned by a PNQ (q, f) from Algorithm 1 is $f(dist(q,p))$, independently from whether other points are returned.*

Proposition 3. *Let T be a quadtree with n points. The running time of Algorithm 1 per query on T is $\Theta(n)$ in expectation.*

4 Queries in Sublinear Time by Subtree Aggregation

One reason for the linear time complexity of the baseline query is the fact that every quadtree node is visited. To reach a sublinear time complexity, we thus aggregate subtrees into *virtual leaf cells* whenever doing so reduces the number of examined cells and does not increase the number of candidates too much.

To this end, let S be a subtree starting at depth l of a quadtree T. During the execution of Algorithm 1, a lower bound $\underline{b}$ for the distance between S and the query point q is calculated, yielding also an upper bound $\overline{b}$ for the neighbor probability of each point in S. At this step, it is possible to treat S as a *virtual leaf cell*, sample jumping widths using $\overline{b}$ as upper bound and use these widths to select candidates within S. Aggregating a subtree to a virtual leaf cell allows skipping leaf cells which do not contain candidates, but uses a weaker bound $\overline{b}$ and thus a potentially larger candidate set. Thus, a fast algorithm requires an aggregation criterion which keeps both the number of candidates and the number of examined quadtree cells low.

As stated before, we record the number of points in each subtree during quadtree construction. This information is now used for the query algorithm: We aggregate a subtree S to a virtual leaf cell exactly if $|S|$, the number of points contained in S, is below $1/f(\mathrm{dist}(S,q))$. This corresponds to less than one expected candidate within S. The changes required in Algorithm 1 to use the subtree aggregation are minor. Lines 5, 14 and 15 are changed to:

---

5 **if** *c is inner node and $|c| \cdot \overline{b} \geq 1$* **then**

---

17 neighbor = maybeGetKthElement(q, f, i, $\overline{b}$, c);
18 add neighbor to N if not null

The main change consists in the use of the function maybeGetKthElement. Given a subtree S, an index k, q, f, and $\overline{b}$, this function descends S to the leaf cell containing the kth element. This element p_k is then accepted with probability $f(\mathrm{dist}(q,p_k))/\overline{b}$.

Since the upper bound calculated at the root of the aggregated subtree is not smaller than the individual upper bounds at the original leaf cells, Proposition 2 also holds for the virtual leaf cells. This establishes the correctness.

The time complexity is given by the following theorem, whose proof can be found in the full version [16].

Theorem 1. *Let T be a quadtree with n points and (q, f) a query pair. A query (q, f) using subtree aggregation has time complexity $O((|\mathrm{N}(q, f)| + \sqrt{n}) \log n)$ whp.*

5 Application Case Studies

In order to test our algorithm for PNQs, we apply it in two application case studies, one for Euclidean, the other one for hyperbolic geometry. For the Euclidean case study we build a simple disease spread simulation as an example for a probabilistic spreading process. The probability distribution of points is in this case non-uniform and unknown. The hyperbolic application, in turn, is a generator for complex networks with a known point distribution.

5.1 Probabilistic Spreading

When both contact graph and travel patterns of a susceptible population are not known in detail, the resulting spreading behavior of an infectious disease seems probabilistic. Contagious diseases usually spread to people in the vicinity of infected persons, but an infectious person occasionally bridges larger distances by travel and spreads the disease this way. We model this effect with our probabilistic neighborhood function f, giving a higher probability for small distances and a lower but non-zero probability for larger distances. Note that this scenario is meant as an example of the probabilistic spreading simulations possible with our algorithm and not as highly realistic from an epidemiological point of view.

In the simulation, the population is given as a set P of points in the Euclidean plane. In the initial step, exactly one point (= person) from P is marked as infected. Then, in each round, a PNQ is performed for each infected person q. All points in $N(q, f)$ become infected in the next round. We use an SIR model [7], i. e. previously infected persons recover with a certain probability in each round and stay infectious otherwise. In our simulation, persons recover with a rate of 0.8 and are then immune.

5.2 Random Hyperbolic Graph Generation

Random hyperbolic graphs (RHGs, also see Section 2.2) are a generative graph model for complex networks. For graph generation one places n points (= vertices) randomly in a hyperbolic disk. The radius R of the disk can be used to control the average degree of the network. A pair of vertices is connected by an edge with

Country	5000 PDP queries	Construction QT	5000 QT queries
France	1007 seconds	1.6 seconds	1.2 seconds
Germany	1395 seconds	2.8 seconds	1.3 seconds
USA	4804 seconds	8.7 seconds	0.7 seconds

Table 1. Running time results for polar Euclidean quadtrees on population data. The query points were selected uniformly at random from P, the probabilistic neighborhood function is $f(x) := (1/x) \cdot e^7/n$.

a probability that depends on the vertices' hyperbolic distance. This connection probability is given in [12, Eq. (41)] and parametrized by a temperature $T \geq 0$:

$$f(x) = \frac{1}{e^{(1/T)\cdot(x-R)/2} + 1} \tag{2}$$

This definition of random hyperbolic graphs is a generalized version of the one considered in our previous work, which was restricted to the special case of $T = 0$.

5.3 Experimental Settings and Results

Our implementation is included in the open-source toolkit NetworKit [15] and is written in C++ 11. Running time measurements were made with g++ 4.8 -O3 on a machine with 128 GB RAM and an Intel Xeon E5-1630 v3 CPU with four cores at 3.7 GHz base frequency. Our code is sequential, as is the reference implementation for random hyperbolic graph generation [2].

Disease Spread Simulation. We experimented on three data sets taken from NASA population density raster data [6] for Germany, France and the USA. They consist of rectangles with small square cells (geographic areas) where for each cell the population from the year 2000 is given. To obtain a set of points, we randomly distribute points in each cell to fit 1/20th of the population density. The data sets of France and USA have roughly 3 and 14 million points, respectively.

The number of required queries naturally depends heavily on the simulated disease. For our parameters, a number of 5000 queries is typically reached within the first dozen steps. To evaluate the algorithmic speedup, Table 1 compares running times for 5000 pairwise distance probing (PDP) queries against 5000 fast PNQs on the three country datasets. To obtain a similar total number of infections, we use a slightly different probabilistic neighborhood function for each country and divide by the population: $f(x) := (1/x) \cdot e^7/n$. This results in a slower initial progression for the US. Our algorithm achieves a speedup factor of at least two orders of magnitude, even including the quadtree construction time.

Random Hyperbolic Graph Generation. We compare our generator using PNQs with the only (to our knowledge) previously existing generator for general random hyperbolic graphs [2], i. e. those not only following the threshold model. As seen in Figure 3, our implementation is faster by at least one order of magnitude and the experimental running times support our theoretical time complexity of $O((n^{3/2} + m) \log n)$.

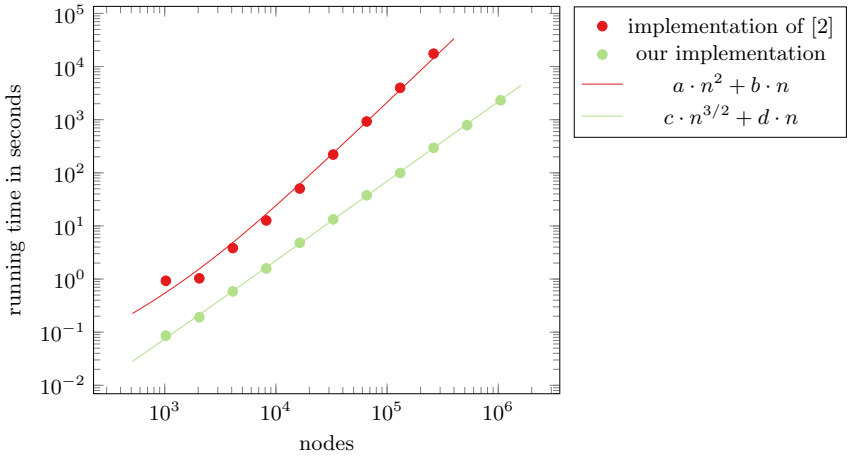

Fig. 3. Comparison of running times to generate networks with 2^{10}-2^{20} vertices, $\alpha = 1$, $T = 0.5$ and average degree $\overline{k} = 6$. The gap between the running times widens, which in the loglog-plot implies a different exponent in the time complexities. Running times are fitted with $a = 2.089 \cdot 10^{-7}$, $b = 3.311 \cdot 10^{-4}$, $c = 2.18 \cdot 10^{-6}$ and $d = 5.6 \cdot 10^{-6}$.

6 Conclusions

After formally defining the notion of probabilistic neighborhoods, we have presented a quadtree-based query algorithm for such neighborhoods in the Euclidean and hyperbolic plane. Our analysis shows a time complexity of $O((|N(q, f)| + \sqrt{n}) \log n)$, our algorithm is to the best of our knowledge the first to solve the problem asymptotically faster than pairwise distance probing. With two example applications we have shown that our algorithm is also faster in practice by at least one order of magnitude.

Acknowledgements. This work is partially supported by German Research Foundation (DFG) grant ME 3619/3-1 within the Priority Programme 1736 *Algorithms for Big Data*. The authors thank Mark Ortmann for helpful discussions.

References

1. Pankaj K Agarwal, Boris Aronov, Sariel Har-Peled, Jeff M Phillips, Ke Yi, and Wuzhou Zhang. Nearest neighbor searching under uncertainty II. In *Proc. 32nd symposium on Principles of database systems*, PODS, pages 115–126. ACM, 2013.
2. Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. In *Computer Physics Communications*, volume 196, pages 492–496. Elsevier, Amsterdam, The Netherlands, Nov 2015.
3. Lars Arge and Kasper Green Larsen. I/O-efficient spatial data structures for range queries. In *SIGSPATIAL Special*, volume 4, pages 2–7. ACM, New York, NY, USA, July 2012.

4. Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.
5. Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *arXiv preprint arXiv:1511.00576*, 2015.
6. Center for International Earth Science Information Network CIESIN Columbia University; Centro Internacional de Agricultura Tropical CIAT. Gridded population of the world, version 3 (gpwv3): Population density grid, 2005.
7. Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.
8. Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proc. 33rd symposium on Principles of database systems*, PODS, pages 246–255. ACM, 2014.
9. Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proc. 20th International Conference on Very Large Data Bases*, VLDB, pages 500–509, San Francisco, USA, 1994. Morgan Kaufmann Publishers Inc.
10. Gerhard K Kraetzschmar, Guillem Pages Gassull, Klaus Uhl, Guillem Pags, and Gassull Klaus Uhl. Probabilistic quadtrees for variable-resolution mapping of large environments. In *Proc. 5th IFAC/EURON symposium on intelligent autonomous vehicles*, 2004.
11. Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Advances in databases: concepts, systems and applications*, pages 337–348. Springer, 2007.
12. Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, Sep 2010.
13. Jian Pei, Ming Hua, Yufei Tao, and Xuemin Lin. Query answering techniques on uncertain and probabilistic data: tutorial summary. In *Proc. 2008 ACM SIGMOD International Conference on Management of data*, pages 1357–1364. ACM, 2008.
14. Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
15. Christian L Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. In *Network Science*. Cambridge University Press, 2016. To appear.
16. Moritz von Looz and Henning Meyerhenke. Querying Probabilistic Neighborhoods in Spatial Data Sets Efficiently. *ArXiv preprint arXiv:1509.01990*.
17. Moritz von Looz, Roman Prutkin, and Henning Meyerhenke. Generating random hyperbolic graphs in subquadratic time. In *ISAAC 2015: Proc. 26th Int'l Symp. on Algorithms and Computation*, Heidelberg, Nov 2015. Springer.