HAL
open science

Practical and efficient algorithms for the geometric hitting set problem

Norbert Bus, Nabil Mustafa, Saurabh Ray

To cite this version:

Norbert Bus, Nabil Mustafa, Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. Discrete Applied Mathematics, 2018, 240, pp.25 - 32. 10.1016/j.dam.2017.12.018 . hal-01797815

HAL Id: hal-01797815

https://hal.science/hal-01797815

Submitted on 22 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Practical and Efficient Algorithms for the Geometric Hitting Set Problem

Norbert Bus[a], Nabil H. Mustafa[1a], Saurabh Ray[b]

[a] *Université Paris-Est, LIGM, Equipe A3SI, ESIEE Paris, France*
{busn,mustafan}@esiee.fr
[b] *Computer Science Department, New York University, Abu Dhabi, United Arab Emirates*
saurabh.ray@nyu.edu

Abstract

The geometric hitting set problem is one of the basic geometric combinatorial optimization problems: given a set P of points and a set $\mathcal{D}$ of geometric objects in the plane, the goal is to compute a small-sized subset of P that hits all objects in $\mathcal{D}$. Recently Agarwal and Pan [7] presented a near-linear time algorithm for the case where $\mathcal{D}$ consists of disks in the plane. The algorithm uses sophisticated geometric tools and data structures with large resulting constants. In this paper, we design a hitting-set algorithm for this case without the use of these data-structures, and present experimental evidence that our new algorithm has near-linear running time in practice, and computes hitting sets within 1.3-factor of the optimal hitting set. We further present `dnet`, a public source-code module that incorporates this improvement, enabling fast and efficient computation of small-sized hitting sets in practice.

Keywords: Geometric Hitting Sets, Approximation Algorithms, Computational Geometry.

1. Introduction

The minimum hitting set problem is one of the fundamental combinatorial optimization problems: given a set system $(P, \mathcal{D})$ consisting of a set P and a set $\mathcal{D}$ of subsets of P (sometimes also called *ranges*), the task is to compute the smallest subset $Q \subseteq P$ that has a non-empty intersection with each of the ranges in $\mathcal{D}$. This problem is strongly NP-hard and if there are no restrictions on the set system $\mathcal{D}$, then it is known that it is NP-hard to approximate the minimum hitting set within a logarithmic factor of the optimal [23].

The problem is NP-complete even for the case where each range has exactly two points, since this problem is equivalent to the vertex cover problem which is known to be NP-complete [16, 13]. A natural case of the hitting set problem occurs when the range space $\mathcal{D}$ is derived from geometry—e.g., given a set P of n points in $\mathbb{R}^2$, and

[1] The work of Nabil H. Mustafa in this paper has been supported by the grant ANR SAGA (JCJC-14-CE25-0016-01).

a set $\mathcal{D}$ of m triangles containing points of P, compute a minimum-sized subset of P that hits all the triangles in $\mathcal{D}$. Unfortunately, for most natural geometric set systems, computing the minimum-sized hitting set remains NP-hard. For example, even the (relatively) simple case where $\mathcal{D}$ is a set of unit disks in the plane is strongly NP-hard [15]. Note that this problem is equivalent to the problem of covering a given set of points in the plane with a minimum number of given unit disks.

Given a set system $(P, \mathcal{D})$, a positive measure μ on P (e.g., the counting measure), and a parameter $\epsilon > 0$, an ϵ-net is a subset $S \subseteq P$ such that $D \cap S \neq \emptyset$ for all $D \in \mathcal{D}$ with $\mu(D \cap P) \geq \epsilon \cdot \mu(P)$. The ϵ-net theorem [14, 19] implies that for a large family of geometric set systems—balls in $\mathbb{R}^d$, half-spaces in $\mathbb{R}^d$, k-sided polytopes, r-admissible set of regions in $\mathbb{R}^d$—there exist ϵ-nets of size $O\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right)$ [14, 17]. We refer the reader to [21] for details on ϵ-nets. For certain range spaces, one can even show the existence of ϵ-nets of size $O\left(\frac{1}{\epsilon}\right)$—an important case being that of disks in $\mathbb{R}^2$ [22].

In 1994, Bronnimann and Goodrich [8] (see also [12]) proved the following interesting connection between the hitting-set problem and ϵ-nets: if one can compute an ϵ-net of size $\frac{c}{\epsilon}$ for a given set system $(P, \mathcal{D})$ in polynomial time, then one can compute a hitting set of size at most $c \cdot \textsc{Opt}$ for $(P, \mathcal{D})$ in polynomial time, where $\textsc{Opt}$ is the size of the optimal (smallest) hitting set. Until very recently, the best algorithms based on this observation had running times of $\Omega(n^2)$, and it had been a long-standing open problem to compute a $O(1)$-approximation to the hitting-set problem for disks in the plane in near-linear time. In a recent breakthrough, Agarwal and Pan [7] presented the first near-linear algorithm for computing $O(1)$-approximations for hitting sets for disks.

One limitation of this technique is that the quality of the solution is a function of the size of the ϵ-net, and so the technique cannot give better than constant-factor approximations. This limitation was overcome using an entirely different technique: local search [20, 11, 6]. It has been shown [20] that the local search algorithm for the hitting set problem for disks in the plane gives a PTAS. Unfortunately the running time of the algorithm to compute a $(1+\epsilon)$-approximation is $n^{O(\frac{1}{\epsilon^2})}$. Based on local search, an $\tilde{O}(n^{2.34})$ time algorithm was proposed in [10] yielding an $(8+\epsilon)$-approximation factor.

Our Contributions

All approaches towards approximating geometric hitting sets for disks have to be evaluated on the questions of computational efficiency as well as approximation quality. In spite of all the progress, there remains a large gap, mainly due to the trade-offs between running times and approximation factors. The breakthrough algorithm of Agarwal and Pan [7], henceforth referred to as the AP algorithm, uses complicated data-structures that have large constants in the running time. In particular, it uses a $O(\log n + k)$-time algorithm for range reporting for disk ranges in the plane (alternatively, for halfspaces in $\mathbb{R}^3$) as well as a dynamic data-structure for maintaining approximate weighted range-counting under disk ranges in poly-logarithmic time. These data structures are based on rather sophisticated machinery, including shallow cuttings and the shallow partition theorem. We refer the reader to the survey [5] for the current state-of-the-art on range searching. These typically involve large constants in their

construction and use; perhaps due to their practical shortcomings, we have not been able to find efficient (in fact, any) implementations of any of these data-structures. In practice, methods based on spatial partitioning are commonly used instead, e.g., quad-trees, kd-trees, R-trees and Box-trees. Many of these practical structures are suited for orthogonal searching problems; indeed, the principal library of geometric algorithms—CGAL—contains efficient algorithms for the orthogonal (called 'windowed') queries, but none for the more general and harder problem of arbitrary half-space range queries.

This work is an attempt to address this shortcoming: based on a practical spatial partitioning data structure tailored to the specific problem at hand, we give a new modified elementary algorithm and implement a variant of the algorithm that works well in practice to compute small-sized hitting sets in near-linear time, though with weaker theoretical guarantees: the worst-case running times are quadratic, while experiments indicated near-linear running times. In fact, it will turn out that an efficient practical solution for the geometric hitting set problem for disks relies on one of the basic structures in the study of planar geometry: Delaunay triangulations. A major advantage of Delaunay triangulations is that their behavior has been extensively studied, there are many efficient implementations available, and they exhibit good behavior for various real-world data-sets as well as random point sets. For computation of ϵ-nets, we will rely on the following result:

Theorem 1.1 ([9]). *Given a set P of n points in $\mathbb{R}^2$ and disk ranges, an ϵ-net of size at most $\frac{13.4}{\epsilon}$ can be computed in expected time $O(n \log n)$.*

Broadly, the algorithm for computing the ϵ-net in the above theorem is the following: first pick a sample Q of size $\Theta(\frac{1}{\epsilon})$. All disks that are not hit by Q are contained in two of the Delaunay disks in the Delaunay triangulation of Q. Thus, it suffices to recursively build a $\frac{\epsilon}{2}$-net for the points in each Delaunay disk. The union of all these nets together with Q is the required ϵ-net.

As an additional benefit, the algorithm used for computing ϵ-nets uses the same Delaunay triangulation as our algorithm, enabling us to reduce computations. More precisely, our contributions are:

1. A hitting set algorithm (Section 2). We present a modification of the algorithm of Agarwal and Pan that does not use any complicated data-structures—just Delaunay triangulations (and point-location on it), ϵ-nets and binary search. For example, it turns out that output sensitive range reporting is not required. This comes with a price: although experimental results indicate a near-linear running time, we have been unable to formally prove that the algorithm runs in expected near-linear time.
2. Implementation and experimental evaluation (Section 3). We present `dnet`, a public source-code module to efficiently compute small-sized hitting sets in practice. We give detailed experimental results on both synthetic and real-world data sets, which indicates that the algorithm computes, on average, a 1.3-approximation in near-linear time.

2. A Hitting Set Algorithm

The Agarwal and Pan (AP) algorithm (shown in Algorithm 1) uses an iterative reweighing strategy, where the idea is to assign a weight $w(\cdot)$ to each $p \in P$ such that the total weight of points contained in each $D \in \mathcal{D}$ is relatively high. It starts by setting $w(p) = 1$ for each $p \in P$. If there exists a disk D with small weight, it increases the weight of the points in D by a multiplicative factor $\beta > 1$ until their total weight exceeds a threshold of $c \cdot \frac{W}{\text{OPT}}$, where c is a fixed constant and $W = \sum_{p \in P} w(p)$ is the current total weight. In the AP algorithm, β is set to 2. However, any constant greater than 1 works and our algorithm uses this flexibility. If after any iteration, all disks have weight above the threshold $\frac{c}{2e\text{OPT}} \cdot W$, return a $\frac{c}{2e\text{OPT}}$-net with respect to these weights, ensuring that every disk is hit.

For the purpose of analysis, Agarwal and Pan conceptually divide the reweighings into $O(\log n)$ phases, where each phase (except perhaps the last) performs $\Theta(\text{OPT})$ reweighings. The implementation of the AP algorithm requires two ingredients: **A)** a range reporting data structure and **B)** a dynamic approximate range counting data structure. The former is used to construct the set of points to be reweighed and the latter is required for figuring out whether a disk needs reweighing. As a pre-processing step, the AP algorithm first computes a $\frac{1}{\text{OPT}}$-net Q to be returned as part of the hitting set. This ensures that the remaining disks not hit by Q contain less than $\frac{n}{\text{OPT}}$ points. Additionally they observe that in any iteration, if less than OPT disks are reweighed, then all disks have weight more than $\frac{cW}{2e\text{OPT}}$.

Algorithm 1: AP algorithm for computing hitting sets [7].

```
Data: A point set P, a set of disks D, a fixed constant c, and the value of OPT.
Compute a (1/OPT)-net, Q, of P and remove disks hit by Q from D.
Set w(p) = 1 for all p ∈ P.
repeat
    foreach D ∈ D do
        if w(D) ≤ cW/OPT then
            reweigh D repeatedly until the weight w(D) exceeds cW/OPT.
    flag = false.
    foreach D ∈ D do
        if w(D) < (c/2e) · (W/OPT) then flag = true.
until flag = false.
return Q along with a (c/(2eOPT))-net of P with respect to w(·).
```

The AP algorithm is simple and has a clever theoretical analysis that proceeds as follows: each reweighing increases the total weight of the points by a relatively small amount, i.e., by $O(\frac{W}{\text{OPT}})$. On the other hand, since each reweighing *must* involve reweighing a point present in an optimal hitting set, the weight of the points in an optimal solution increases relatively quickly. A quick calculation then shows that after a logarithmic number of steps, the total weight would be smaller than the weight of the optimal hitting set, a contradiction. Its main drawback is that the two data structures

it uses are sophisticated with large constants in the running time. This unfortunately renders the AP algorithm impractical. Our goal is to find a method that avoids these sophisticated data structures and to develop additional heuristics which lead to not only a fast implementation but also one that generally gives an approximation ratio smaller than that guaranteed by the theoretical analysis of the AP algorithm. As part of the algorithm, we will use Theorem 1.1 for constructing small-sized ϵ-nets.

*Removing **A)***. Just as Agarwal and Pan do, we start by picking a $\frac{c_1}{\text{OPT}}$-net, for some constant c_1. The idea for getting rid of the range-reporting data structure is to observe the following. If a disk D is not hit by Q, then D is contained in the union of two Delaunay disks in the Delaunay triangulation of Q. We will show that these two disks can be found quickly by a simple binary search. We can then check all points in the two disks for containment in D and thus enumerate all points in D. Since each of the Delaunay disks contain at most ϵn points, this takes $O(\epsilon n)$ time.

*Removing **B)***. Our approach towards removing the dependence on dynamic approximate range counting data structure is the following: at the beginning of *each* phase, we pick a $\frac{c_2}{\text{OPT}}$-net R, for some constant c_2. We then compute the set of disks that are not hit by R by going over the disks one by one and checking if they are hit by R. Checking whether a disk D is hit by R can be done by finding the closest point in R to the center of D and checking whether the distance between the two points is at most the radius of D. Finding the closest point in R to a query point can be done efficiently by doing point location in the Voronoi diagram of R. We reweigh the disks that are not hit by R, which are guaranteed to have weight at most $\frac{c_2 W}{\text{OPT}}$.

While this avoids having to use data-structure **B)**, there are two problems with this:

a) disks with small weight hit by R are not reweighed, while they should have been reweighed, and
b) a disk whose initial weight was less than $\frac{c_2 W}{\text{OPT}}$ could have its current weight more than $\frac{c_2 W}{\text{OPT}}$ in the middle of a phase, and so it is erroneously reweighed. This would imply that we reweigh disks of large weight, which would then be insufficient to derive a contradiction as in the AP algorithm analysis.

Towards solving these problems, the idea is to maintain an additional set S which is empty at the start of each phase. When a disk D is reweighed, we add a random point of D, sampled according to the probability distribution induced by $w(\cdot)$, to S. We reweigh a disk only if it is not hit by $R \cup S$. Checking whether a disk is hit by $R \cup S$ can again be done using point location in Voronoi diagrams as described before. We already have such a data structure for R. We additionally need to maintain a similar data structure for S. Now, if during a phase, there are $\Omega(\text{OPT})$ reweighings, then as in the Agarwal and Pan algorithm, we move on to the next phase, and $a)$ is not a problem. Otherwise, there have been less than OPT reweighings, which implies that less than OPT disks were not hit by R. Then we can return R together with the set S consisting of one point from each of these disks. This will still be a hitting set.

To remedy $b)$, before reweighing a disk, we compute the set of points inside D, and only reweigh if the total weight is at most $\frac{c_2 W}{\text{OPT}}$. Consequently we sometimes waste $O(\frac{n}{\text{OPT}})$ time to compute this list of points inside D without performing a reweighing.

Algorithm 2: Algorithm for computing small-sized hitting sets.

Data: A point set P, a set of disks $\mathcal{D}$, and the size of the optimal hitting set OPT.

Compute a $(\frac{c_1}{\text{OPT}})$-net Q of P and the Delaunay triangulation $\Xi(Q)$ of Q.
foreach $q \in Q$ **do** construct $\Psi(Q)(q)$.
foreach $D \in \mathcal{D}$ **do**
  if D *not hit by* Q **then** add D to $\mathcal{D}_1$. `// using` $\Xi(Q)$
$P_1 = P \setminus Q$.
foreach $p \in P_1$ **do** set $w(p) = 1$.
repeat
  Compute a $(\frac{c_2}{\text{OPT}})$-net, R, of P_1 and the Delaunay triangulation $\Xi(R)$ of R.
  Set $S = \emptyset$, $\Xi(S) = \emptyset$.
  foreach $D \in \mathcal{D}_1$ *in a random order* **do**
    if D *not hit by* $R \cup S$ **then** `// using` $\Xi(R)$ `and` $\Xi(S)$
      foreach $p \in D$ **do** set $w(p) = w(p) + c_3 w(p)$. `// using` $\Psi(Q)$
      Add a random point lying in D to S; update $\Xi(S)$.
until $|S| \leq c_4 \text{OPT}$.
return $\{Q \cup R \cup S\}$.

Due to this, the worst-case running time increases to $O(\frac{n^2}{\text{OPT}})$. In practice, this is unlikely to happen for the following reason: in contrast to the AP algorithm, a disk is responsible for only one reweighing in our algorithm. Therefore if the weight of any disk D increases significantly, and yet D is not hit by S, the increase must have been due to the increase in weight of many disks intersected by D which were reweighed before D *and* for which the picked points (added to S) did not hit D. Reweighing in a random order makes these events very unlikely (in fact we suspect this gives an expected linear-time algorithm, though we have not been able to prove it).

See Algorithm 2 for the new algorithm (the data-structure $\Psi(Q)$ will be defined later).

Lemma 2.1. *Algorithm 2 terminates, $Q \cup R \cup S$ is a hitting set of size at most* $(13.4 + \delta) \cdot \text{OPT}$*, for any* $\delta > 0$.

Proof. By construction, if the algorithm terminates, then $Q \cup R \cup S$ is a hitting-set. Set $c_1 = 13.4 \cdot 3/\delta$, $c_2 = 1/(1+\delta/(13.4 \cdot 3))$, $c_3 = \delta/10000$ and $c_4 = \delta/3$. Now we apply a standard reweighing argument (see [7]) as follows. The weight increases by at most $c_3 c_2 \cdot \frac{W}{\text{OPT}}$ at each reweighing, and as the initial weight was n, after t steps, the total weight of the points of P is upper-bounded by $n \cdot (1 + \frac{c_2 c_3}{\text{OPT}})^t$. On the other hand, each reweighing will reweigh a point of an optimal hitting set. Noting that a point reweighed a total of r times will have weight $(1 + c_3)^r$ at the end, the total weight of the points in an optimal hitting set is, by the convexity of the exponential function, lower-bounded by $\text{OPT}\,(1 + c_3)^{\frac{t}{\text{OPT}}}$. Thus after t reweighings, we have:

$$\text{OPT}\,(1 + c_3)^{\frac{t}{\text{OPT}}} \leq n \cdot \left(1 + \frac{c_2 c_3}{\text{OPT}}\right)^t, \tag{1}$$

which solves to $t = O\left(\frac{\textsc{Opt}\log n}{\delta}\right)$. Each iteration of the repeat loop, except the last one, does at least $c_4\textsc{Opt}$ reweighings. Then the repeat loop can run for at most $O(\frac{\textsc{Opt}\log n}{c_4\textsc{Opt}\delta}) = O(\frac{\log n}{\delta})$ times.

By Theorem 1.1, $|Q| \leq (13.4/c_1)\textsc{Opt}$, $|R| \leq (13.4/c_2)\textsc{Opt}$, and $|S| \leq c_4\textsc{Opt}$. Thus the overall size is $13.4\textsc{Opt} \cdot \left(1/c_1 + 1/c_2 + c_4/13.4\right) \leq (13.4 + \delta) \cdot \textsc{Opt}$.

Remark. The values of c_1, c_2 and c_4 are chosen so that the terms $1/c_1$, $1/c_2$ and $c_4/13.4$ in the last expression above are roughly the same (about $\delta/3$). It is not necessary to set them this way. Similarly, c_3 is chosen so that it is a small fraction of δ. The constant 10000 is arbitrarily. We have not optimized these constants. □

Algorithmic details. Computing an ϵ-net takes $O(n \log n)$ time using Theorem 1.1. Checking if a disk D is hit by an ϵ-net (Q, R, or S) reduces to finding the closest point in the set to the center of D, again accomplished in $O(\log n)$ time using point-location in Delaunay/Voronoi diagrams $\Xi(\cdot)$. It remains to show how to compute, for a given disk $D \in \mathcal{D}_1$, the set of points of P contained in D:

Lemma 2.2. *Given a disk $D \in \mathcal{D}_1$, the set of points of P contained in D can be reported in time $O(\frac{n}{\textsc{Opt}} \log n)$.*

Proof. Each disk in $\mathcal{D}_1$ is not hit by Q, and so contains at most $\frac{c_1 n}{\textsc{Opt}}$ points of P. We now show how, given any disk D with $D \cap Q = \emptyset$, one can find two disks whose union covers D in $O(\log n)$ time. Given D, compute, using $\Xi(Q)$, the nearest neighbor $p \in Q$ to the center of D. Consider the list of Delaunay triangles incident to p, sorted by their circumcenters radially around p. Denote this list for the point p by $\Psi(Q)(p)$. Let D'_1 and D'_2 be the two Delaunay disks of $\Xi(Q)$ whose triangles are adjacent to p, and whose circumcenters are immediately before and after the center of D in this radially sorted order.

Claim 2.1. $D \subseteq D'_1 \cup D'_2$.

Proof. The proof of Theorem 1.1 shows that for any disk D not hit by the ϵ-net Q, there exist two Delaunay disks of $\Xi(Q)$, say D_1 and D_2, such that $D \subseteq D_1 \cup D_2$. In particular, the proof shows that given D, D_1 and D_2 are circumcircles of two adjacent Delaunay triangles, say Δ_e^1 and Δ_e^2, where $e = \{p_1, p_2\} \in \Xi(Q)$ is the shared Delaunay edge. Moreover, one of the vertices of e, say p_1, is the closest point in Q to the center of D.

We finish the proof by showing that $i)$ the circumcenters of D_1 and D_2 are consecutive in the radially sorted list $\Psi(Q)(p_1)$, and $ii)$ the center of D lies between $c(D_1)$ and $c(D_2)$ in this consecutive order. Thus $\{D'_1, D'_2\} = \{D_1, D_2\}$. See Figure 1(a) for the geometric configuration.

For contradiction assume a disk D' whose $c(D')$ lies between $c(D_1)$ and $c(D_2)$. D' passes through p_1, and there are two cases:

- D' enters/leaves p_1 through D_1/D_2 (Figure 1(b) bottom). Then either $D' \subset D_1 \cup D_2$, and the two points of Q on $\partial D'$ lie inside $D_1 \cup D_2$, contradicting emptiness of $int(D_1 \cup D_2)$. Or D' must contain p_2, contradicting emptiness of D'.

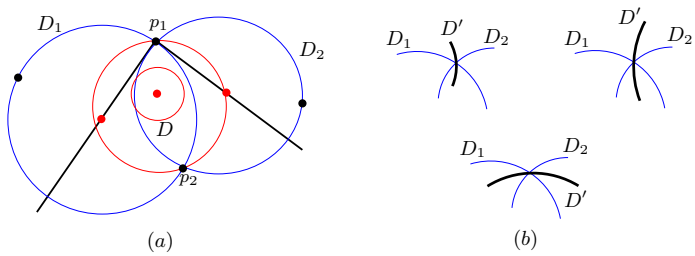

Figure 1: Computing D_1 and D_2.

- D' enters/leaves p_1 through outside ((Figure 1(b) top). Then $c(D')$ must lie radially outside the interval of $c(D_1)$ and $c(D_2)$.

□

Finally note that $\Psi(Q)$ can be constructed in the pre-processing phase in expected $O(n \log n)$ time: for each point $q \in Q$, extract its set of adjacent Delaunay triangles from $\Xi(Q)$, and radially sort their circumcenters around q to get the list $\Psi(Q)(q)$. As the number of triangles is $O(|Q|)$, this takes time $O(|Q| \log |Q|) = O(\frac{n}{\text{OPT}} \log n)$. And for each D, one can find the two Delaunay disks D'_1 and D'_2 by binary search in $\Psi(Q)(q)$ in time $O(\log n)$. The points of P lying in D can be reported by going over all points of P in D'_1 and D'_2 and checking whether they lie in D. □

3. Implementation and Experimental Evaluation

In this section we present experimental results for our algorithms implemented in C++ and running on a machine equipped with an Intel Core i7 870 processor (2.93 GHz) and with 16 GB main memory. All our implementations are single-threaded, but we note that our hitting set algorithm can be easily multi-threaded. For nearest-neighbors and Delaunay triangulations, we use CGAL [24]. It computes Delaunay triangulations in expected $O(n \log n)$ time. The parameters used by our algorithm are set as follows: $c_0 = 10, c_1 = 30, c_2 = 12, c_3 = 2$ and $c_4 = 0.6$. These parameters are chosen taking into account various trade-offs: e.g., higher value of c_1 results in a smaller net at step 1, but that increases the number of violations at step 4, thus increasing the size of $R \cup S$.

For evaluating the practical usability of our approximate hitting set algorithm we compare it to the optimal solution. Our algorithm needs a guess for OPT, and so we run it with $O(\log n)$ guesses for the value of OPT. To calculate the optimal solution for the hitting set problem we use the IP solver SCIP [4] (with the linear solver SoPlex [25]). Creating the linear program is carried out efficiently by using the Delaunay triangulation of the points for efficient range searching.

Data sets. In order to empirically validate our algorithms we have utilized several real-world point sets. All our experiments' point sets are scaled to a unit square. The *World* dataset [3] contains locations of cities on Earth (except for the US) having around 10M records. For our experiments we use only the locations of cities in China having

1M records (the coordinates have been obtained from latitude and longitude data by applying the Miller cylindrical projection). The dataset *ForestFire* contains 700K locations of wildfire occurrences in the United States [2]. The following data sets were obtained from [1]. The *KDDCUP04Bio* dataset (*KDDCU* for short) contains the first 2 dimensions of a protein dataset with 145K entries. The *MOPSI Finland* dataset contains 13K location records of users in Finland. The *Europe* and *Birch3* data sets have 169K and 100K entries respectively. The former consists of differential coordinates of the map of Europe while the latter has random sized clusters in random locations. We have created a random data set *Uniform* with 50K points sampled from a uniform distribution. We have also created a random data set *Gauss9* with 90K points sampled from 9 different Gaussian distributions with random mean and covariance matrices. In order to provide more detailed data with optimal solutions we have restricted the size of our point sets to 50K, otherwise the memory required by the IP solver is more than what is available on our machine. Our data sets only contain points and in order to create disks for the hitting set problem we have utilized two different strategies. In the first approach we create uniformly distributed disks in the unit square with uniformly distributed radius within the range $[0, r]$. Let us denote this test case as $RND(r)$. In the second approach we added disks centered at each point of the dataset with a fixed radius of 0.001. Let us denote this test case by $FIX(0.001)$.

Table 1: Hitting sets for the test case $RND(0.1)$.

	# of points	# of disks	Q size	R size	S size	# of phases	IP solution	*dnet* solution	ap-prox.	IP time(s)	*dnet* time(s)
China	50K	50K	367	809	604	11	1185	1780	1.5	60	12
ForestFire	50K	16K	43	85	224	11	267	352	1.3	54.3	6.9
KDDCU	50K	22K	171	228	786	11	838	1185	1.4	40.9	9.8
Gauss9	50K	35K	322	724	1035	11	1493	2081	1.4	52.5	11.7
Europe	50K	31K	185	322	419	11	630	926	1.5	87.6	9.8
Birch3	50K	29K	166	233	1036	11	1026	1415	1.4	51.4	10.4
Uniform	50K	48K	665	1109	2169	11	2824	3943	1.4	49.8	12.3
Mopsi	13K	5K	40	55	197	10	228	292	1.3	2.4	1.7

Speed and approximation quality. The results are shown in Table 1, 2 and 3 for $RND(0.1)$, $RND(0.01)$ and $FIX(0.001)$ respectively. Our algorithm provides a 1.3 approximation on average. With small radius the solver seems to outperform our algorithm but this is most likely due to the fact that the problems become relatively simpler and various branch-and-bound heuristics become efficient. With bigger radius and therefore more complex constraint matrix our algorithm clearly outperforms the IP solver. Our method obtains a hitting set for all point sets, while in some of the cases the IP solver was unable to compute a solution in reasonable time (we terminate the solver after 1 hour).

Memory usage and efficiency of our algorithm. In Table 4 we have included the memory consumption of both methods and statistics for range reporting. It is clear that the IP solver requires significantly more memory than our method. The statistics for range reporting includes the total number of range reportings (calculating the points inside a disk) and the number of range reportings when the algorithm doubles the weight of the points inside a disk (the doubling column in the table). It can be seen that only a fraction of the computations are wasted since the number of doublings is almost as

Table 2: Hitting sets for the test case $RND(0.01)$.

	# of points	# of disks	Q size	R size	S size	# of phases	IP solution	*dnet* solution	ap-prox.	IP time(s)	*dnet* time(s)
China	50K	49K	673	1145	4048	11	4732	5862	1.2	4.5	14.5
ForestFire	50K	25K	162	268	1021	11	1115	1451	1.3	6.2	9.5
KDDCU	50K	102K	1326	2492	6833	11	8604	10651	1.2	12.5	22.2
Gauss9	50K	185K	2737	6636	9867	11	15847	19239	1.2	22.4	36.0
Europe	50K	85K	683	1491	3138	11	4211	5312	1.3	13.2	18.1
Birch3	50K	117K	1359	3358	7223	11	9683	11940	1.2	15.4	25.6
Uniform	50K	387K	5549	13081	16826	11	31787	35446	1.1	34.7	66.3
MOPSI	13K	6K	179	314	656	10	762	1009	1.3	0.6	2.5

Table 3: Hitting sets for the test case $FIX(0.001)$.

	# of points	# of disks	Q size	R size	S size	# of phases	IP solution	*dnet* solution	ap-prox.	IP time(s)	*dnet* time(s)
China	50K	50K	2765	7376	7851	11	–	17329	–	–	19.8
ForesFire	50K	50K	331	602	1273	11	–	2206	–	–	11.9
KDDCU	50K	50K	2764	5824	15734	11	22368	24318	1.1	8.9	22.3
Gauss9	50K	50K	5380	13321	19153	11	36302	37827	1.0	19.1	26.7
Europe	50K	50K	1376	2644	5161	11	–	9181	–	–	16.2
Birch3	50K	50K	2709	7492	14434	11	–	24630	–	–	22.1
Uniform	50K	50K	5442	13417	27573	11	46124	46420	1.0	19.4	30.4
MOPSI	13K	13K	354	673	870	10	1294	1646	1.3	127.2	2.9

high as the total number or range reportings. This in fact shows that the running time of our algorithm is near-linear in n.

Table 4: Memory usage in MB (left) and range reporting statistics (right).

	RND(0.01)		RND(0.1)		FIX(0.001)	
	IP	*dnet*	IP	*dnet*	IP	*dnet*
China	243	21	4282	19	434	20
ForesFire	524	28	3059	18	5470	24
KDDCU	458	30	2999	23	175	22
Gauss9	569	33	3435	24	158	24
Europe	734	30	4418	25	659	24
Birch3	523	31	3655	24	960	26
Uniform	693	39	4083	25	155	24
MOPSI	30	11	294	8	1735	10

	RND(0.01)		RND(0.1)		FIX(0.001)	
	total	doubling	total	doubling	total	doubling
China	44014	43713	9406	9184	96335	95846
ForesFire	11167	11086	2767	2728	15648	15020
KDDCU	75448	75016	8485	8364	173147	173044
Gauss9	121168	120651	14133	13906	217048	217019
Europe	38112	37886	5224	5160	58644	57014
Birch3	85399	85063	11219	11049	167384	167197
Uniform	198168	197388	26970	26431	309286	309284
MOPSI	1883	1863	2283	2249	10360	10235

Scalability. In order to test the scalability of our method compared to the IP solver we have used the *ForestFire* and *China* dataset with limiting the number of points to 10K, 20K, 30K... and repeating exactly the same experiments as above (while increasing the number of disks in a similar manner). In Figure 2 we plot the running time of the methods. The solid lines represent the case $RND(0.1)$ while the dashed ones denote $RND(0.01)$. One can see that as the number of points and disks increases our method becomes more efficient even though for small instances this might not hold. It can be seen that for the *China* dataset and $RND(0.01)$ the IP solver is faster than our method but after 500K points our method becomes faster. In Figure 2 the dotted line represents the running time of our algorithm for $FIX(0.001)$. In this case the IP running time is not shown because the solver was only able to solve the problem with 10K points within a reasonable time (for 20K and 30K points it took 15 and 21 hours respectively).

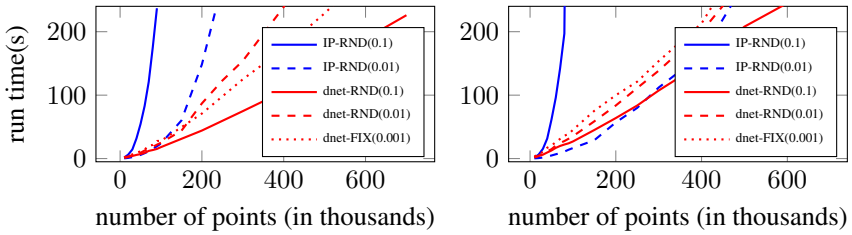

Figure 2: Different point set sizes for the *ForestFire* (left) and *China* (right) datasets.

Robustness. We have varied the radius of the disks for the fixed radius case to see how the algorithms behave. See Figure 3. With bigger radius the IP solver becomes very quickly unable to solve the problem (for radius 0.002 it was unable to finish within a day), showing that our method is more robust.

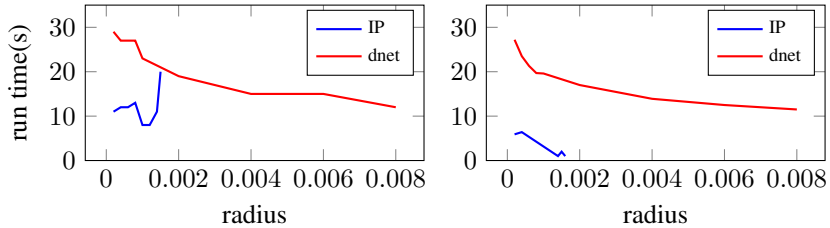

Figure 3: Different radii settings for the *KDDCU* (left) and *China* (right) datasets.

In order to test the extremes of our algorithm we have taken the *World* dataset containing 10M records. Our algorithm was able to calculate the solution of the $FIX(0.001)$ problem of size around 100K in 3.5 hours showing that the algorithm has the potential to calculate results even for extremely big data sets with a more optimized (e.g., multi-threaded) implementation.

4. Conclusion

We have presented a novel algorithm for efficiently computing small-sized hitting sets for disks in the plane, even for very large data sets. The algorithm achieves near linear-running time in practice. This research opens up several avenues of future work:

- It would be interesting to extend the algorithm to half-spaces in three dimensions, as well as for more general pseudo-disks in the plane.
- Given the recent advances in GPU-based algorithms, the next natural step is to consider algorithms on graphics card architectures, e.g., CUDA-based algorithms.
- The main bottleneck for extending these results to more general geometric objects is that the size of ϵ-nets can become super-linear (in $\frac{1}{\epsilon}$), e.g. [18], and thus the reweighing algorithms can no longer guarantee constant-factor approximation ratios.

[1] Clustering Datasets. http://cs.joensuu.fi/sipu/datasets/. Accessed: 2015-02-05.

[2] Federal Wildland Fire Occurence Data, United States of America Geological Survey. http://wildfire.cr.usgs.gov/firehistory/data.html. Accessed: 2015-02-05.

[3] Geographic Names Database, maintained by the National Geospatial-Intelligence Agency. http://geonames.nga.mil/gns/html/. Accessed: 2015-02-05.

[4] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[5] P. K. Agarwal. Range searching. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 2017.

[6] P. K. Agarwal and N. H. Mustafa. Independent set of intersection graphs of convex objects in 2d. *Comput. Geom.*, 34(2):83–95, 2006.

[7] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Symposium on Computational Geometry*, page 271, 2014.

[8] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[9] N. Bus, S. Garg, N. H. Mustafa, and S. Ray. Tighter estimates for ϵ-nets for disks. *Comput. Geom.*, 53:27–35, 2016.

[10] N. Bus, S. Garg, N. H. Mustafa, and S. Ray. Limits of local search: Quality and efficiency. *Discrete & Computational Geometry*, pages 1–18, 2017.

[11] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

[12] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Inf. Process. Lett.*, 95:358–362, 2005.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[14] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[15] D. S. Hochbaum and W. Maass. Fast approximation algorithms for a nonconvex covering problem. *J. Algorithms*, 8(3):305–323, 1987.

[16] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[17] J. Komlós, J.Pach, and G. J. Woeginger. Almost tight bounds for epsilon-nets. *Discrete & Computational Geometry*, 7:163–173, 1992.

[18] A. Kupavskii, N. H. Mustafa, and J. Pach. Near-optimal lower bounds for epsilon-nets for half-spaces and low complexity set systems. In Martin Loebl, Jaroslav Nešetřil, and Robin Thomas, editors, *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*. Springer, 2017.

[19] N. H. Mustafa, K. Dutta, and A. Ghosh. A simple proof of optimal epsilon nets. *Combinatorica*, 2017. https://doi.org/10.1007/s00493-017-3564-5.

[20] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[21] N. H. Mustafa and K. Varadarajan. Epsilon-approximations and epsilon-nets. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 2017.

[22] E. Pyrga and S. Ray. New existence proofs for epsilon-nets. In *Proceedings of Symposium on Computational Geometry*, pages 199–207, 2008.

[23] R. Raz and M. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of STOC*, pages 475–484, 1997.

[24] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015.

[25] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.