Geometric Hitting Sets for Disks: Theory and Practice

Norbert Bus, Nabil Mustafa, Saurabh Ray

Geometric Hitting Sets for Disks: Theory and Practice

Norbert Bus[1], Nabil H. Mustafa[1]*, and Saurabh Ray[2]

[1]Université Paris-Est, LIGM, Equipe A3SI, ESIEE Paris, France
{busn,mustafan}@esiee.fr
[2]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 $\mathcal{D}$. In 1994, Bronniman and Goodrich [6] made an important connection of this problem to the size of fundamental combinatorial structures called ϵ-nets, showing that small-sized ϵ-nets imply approximation algorithms with correspondingly small approximation ratios. Finally, recently Agarwal-Pan [5] showed that their scheme can be implemented in near-linear time for disks in the plane. This current state-of-the-art is lacking in three ways. First, the constants in current ϵ-net constructions are large, so the approximation factor ends up being more than 40. Second, the algorithm uses sophisticated geometric tools and data structures with large resulting constants. Third, these have resulted in a lack of available software for fast computation of small hitting-sets. In this paper, we make progress on all three of these barriers: $i)$ we prove improved bounds on sizes of ϵ-nets, $ii)$ design hitting-set algorithms without the use of these data-structures and finally, $iii)$ present `dnet`, a public source-code module that incorporates both of these improvements to compute small-sized hitting sets and ϵ-nets efficiently in practice.

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

1 Introduction

The minimum hitting set problem is one of the most fundamental combinatorial optimization problems: given a range space $(P, \mathcal{D})$ consisting of a set P and a set $\mathcal{D}$ of subsets of P called the *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. 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. 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. A natural occurrence of the hitting set problem occurs when the range space

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

$\mathcal{D}$ is derived from geometry – e.g., given a set P of n points in $\mathbb{R}^2$, and a set $\mathcal{D}$ of m triangles containing points of P, compute the minimum-sized subset of P that hits all the triangles in $\mathcal{D}$. Unfortunately, for most natural geometric range spaces, 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 [10].

Given a range space $(P, \mathcal{D})$, 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 [9] implies that for a large family of geometric hitting set systems (e.g., disks, half-spaces, k-sided polytopes, r-admissible set of regions in $\mathbb{R}^d$) there exists an ϵ-net of size $O(d/\epsilon \log d/\epsilon)$. For certain range spaces, one can even show the existence of ϵ-nets of size $O(1/\epsilon)$ – an important case being for disks in $\mathbb{R}^2$ [12]. In 1994, Bronnimann and Goodrich [6] proved the following interesting connection between the hitting-set problem, and ϵ-nets: if one can compute an ϵ-net of size c/ϵ for the ϵ-net problem for $(P, \mathcal{D})$ in polynomial time, then one can compute a hitting set of size at most $c \cdot \textsc{Opt}$ for $(P, \mathcal{D})$, where $\textsc{Opt}$ is the size of the optimal (smallest) hitting set, in polynomial time. Until very recently, the best algorithms based on this observation, referred to as rounding techniques, 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 break-through, Agarwal-Pan [5] presented the first near-linear algorithm for computing $O(1)$-approximations for hitting sets for disks.

The limitation of the rounding technique – that it cannot give a PTAS – was overcome using an entirely different technique: local search [11,4]. It has been shown that the local search algorithm for the hitting set problem for disks in the plane gives a PTAS. Unfortunately the running time of the naive algorithm to compute a $(1+\epsilon)$-approximation is $O(n^{O(1/\epsilon^2)})$. Based on local search, an $\tilde{O}(n^{2.34})$ time algorithm was proposed [7] yielding an $(8+\epsilon)$-approximation.

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 ugly trade-offs between running times and approximation factors. The breakthrough algorithm of Agarwal-Pan [5] suffers from two main problems:

- It rounds via ϵ-nets to design a $\tilde{O}(n)$-time algorithm, but the constant in the approximation depends on the constant in the size of ϵ-nets, which are large. For disks in the plane, the current best size of ϵ-net is at least $40/\epsilon$ [12], yielding at best a 40-approximation algorithm. Furthermore, there is no implementation or software solution available that can even compute such ϵ-nets efficiently.
- It uses sophisticated data-structures that have large constants in the running time. In particular, it uses the $O(\log n + k)$-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 polylogarithmic time. We have not been able to find efficient implementations of any of these data-structures.

It will turn out that all ideas for an efficient practical solution for the geometric hitting set problem for disks are unified by one of the basic structures in the study of planar geometry: Delaunay triangulations. Delaunay triangulations will be the key structure for computing these improved ϵ-nets, and the Delaunay structures *already computed* for constructing these nets will turn out to be crucial in computing small-sized hitting sets. More precisely, our contributions are:

1. Constructing small ϵ-nets (Section 2). We show that the sample-and-refine approach of Chazelle-Friedman [8] together with additional structural properties of Delaunay triangulation results in ϵ-nets of surprisingly low size:

Theorem 1. *Given a set P of n points in $\mathbb{R}^2$, there exists an ϵ-net under disk ranges of size at most $13.4/\epsilon$. Furthermore it can be computed in expected time $O(n \log n)$.*

The algorithm is simple to implement. We have implemented it, and present the sizes of ϵ-nets for various real-world data-sets; the results indicate that our theoretical analysis closely tracks the actual size of the nets.

2. Engineering a hitting-set algorithm (Section 3). Together with the result of Agarwal-Pan, this immediately implies:

Corollary 1. *For any $\delta > 0$, one can compute a $(13.4 + \delta)$-approximation to the minimum hitting set for $(P, \mathcal{D})$ in time $\tilde{O}(n)$.*

We then present a modification of the algorithm of Agarwal-Pan that does not use any complicated data-structures – just Delaunay triangulations, ϵ-nets and binary search (e.g., 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 theoretically prove that the algorithm runs in expected near-linear time.

3. Implementation and experimental evaluation (Section 4). We present `dnet`, a public source-code module that incorporates all these ideas 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. This surprisingly low approximation factor compared to the proven worst case bound is the result of fine tuning the parameters of the algorithm.

Due to lack of space, most of the proofs are left for the full paper.

2 A Near Linear Time Algorithm for Computing ϵ-nets for Disks in the Plane

Through a more careful analysis, we present an algorithm for computing an ϵ-net of size $\frac{13.4}{\epsilon}$, running in expected near linear time. The method, shown in Algorithm 1, computes a random sample and then solves subproblems involving subsets of the points located in pairs of Delaunay disks circumscribing adjacent triangles in the Delaunay triangulation of the random sample. The key to improved bounds is $i)$ using additional structural properties of Delaunay triangulations, and $ii)$ new improved constructions of ϵ-nets for large values of ϵ. The presented algorithm can be extended to handle the case when the ϵ-net is with respect to a measure on the point set taking only rational values.

Let $\Delta(abc)$ denote the triangle defined by the three points a, b and c. D_{abc} denotes the disk through a, b and c, while $D_{ab\overline{c}}$ denotes the halfspace defined by a and b not containing the point c. Let $c(D)$ denote the center of the disk D.

Let $\Xi(R)$ be the Delaunay triangulation of a set of points $R \subseteq P$ in the plane. We will use Ξ when R is clear from the context. For any triangle $\Delta \in \Xi$, let D_Δ be the Delaunay disk of Δ, and let P_Δ be the set of points of P contained in D_Δ. Similarly, for any edge $e \in \Xi$, let Δ_e^1 and Δ_e^2 be the two triangles in Ξ adjacent to e, and $P_e = P_{\Delta_e^1} \bigcup P_{\Delta_e^2}$. If e is on the convex-hull, then one of the triangles is taken to be the halfspace whose boundary line is supported by e and not containing R.

Algorithm 1: Compute ϵ-nets

```
Data: Compute $\epsilon$-net, given $P$: set of $n$ points in $\mathbb{R}^2$, $\epsilon > 0$ and $c_0$.
if $\epsilon n < 13$ then
    Return $P$
Pick each point $p \in P$ into $R$ independently with probability $\frac{c_0}{\epsilon n}$.
if $|R| \leq c_0/2\epsilon$ then
    restart algorithm.
Compute the Delaunay triangulation $\Xi$ of $R$.
for triangles $\Delta \in \Xi$ do
    Compute the set of points $P_\Delta \subseteq P$ in Delaunay disk $D_\Delta$ of $\Delta$.
for edges $e \in \Xi$ do
    Let $\Delta_e^1$ and $\Delta_e^2$ be the two triangles adjacent to $e$, $P_e = P_{\Delta_e^1} \cup P_{\Delta_e^2}$.
    Let $\epsilon' = (\frac{\epsilon n}{|P_e|})$ and compute an $\epsilon'$-net $R_e$ for $P_e$ depending on the cases below:
    if $\frac{1}{2} < \epsilon' < 1$ then
        compute using Lemma 1.
    if $\epsilon' \leq \frac{1}{2}$ then
        compute recursively.
Return $\left(\bigcup_e R_e\right) \cup R$.
```

In order to prove that the algorithm gives the desired result, the following lemma regarding the size of an ϵ-net will be useful. Let $f(\epsilon)$ be the upper bound of the size of the smallest ϵ-net for any set P of points in $\mathbb{R}^2$ under disk ranges.

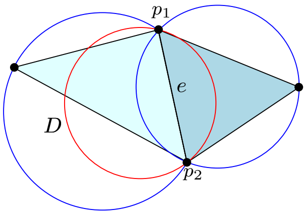
Lemma 1. *For $\frac{2}{3} < \epsilon < 1$, $f(\epsilon) \leq 2$, and for $\frac{1}{2} < \epsilon \leq \frac{2}{3}$, $f(\epsilon) \leq 10$. In both cases the ϵ-net can be computed in $O(n \log n)$ time.*

Call a tuple $(\{p, q\}, \{r, s\})$, where $p, q, r, s \in P$, a *Delaunay quadruple* if $int(\Delta(pqr)) \cap int(\Delta(pqs)) = \emptyset$ where $int(\cdot)$ denotes the interior of a set. Define its *weight*, denoted $W_{(\{p,q\},\{r,s\})}$, to be the number of points of P in $D_{pqr} \cup D_{pqs}$. Let $\mathcal{T}_{\leq k}$ be the set of Delaunay quadruples of P of weight at most k and similarly $\mathcal{T}_k$ denotes the set of Delaunay quadruples of weight exactly k. Similarly, a *Delaunay triple* is given by $(\{p,q\},\{r\})$, where $p,q,r \in P$. Define its *weight*, denoted $W_{(\{p,q\},\{r\})}$, to be the number of points of P in $D_{pqr} \cup D_{pq\overline{r}}$. Let $\mathcal{S}_{\leq k}$ be the set of Delaunay triples of P of weight at most k, and $\mathcal{S}_k$ denotes the set of Delaunay triples of weight exactly k. One can upper bound the size of $\mathcal{T}_{\leq k}$, $\mathcal{S}_{\leq k}$ and using it, we derive an upper bound on the expected number of sub-problems with a certain number of points.

Lemma 2. *If* $\epsilon n \geq 13$, $\boldsymbol{E}\Big[\big|\{e \in \varXi \mid k_1\epsilon n \leq |P_e| \leq k_2\epsilon n\}\big|\Big] \leq \frac{(3.1)c_0^3}{\epsilon e^{k_1 c_0}}(k_1^3 c_0 + 3.7k_2^2)$.

Lemma 3. *Algorithm* COMPUTE ϵ-NET *computes an* ϵ*-net of expected size* $13.4/\epsilon$.

Proof. First we show that the algorithm computes an ϵ-net. Take any disk D with center c containing ϵn points of P, and not hit by the initial random sample R. Increase its radius while keeping its center c fixed until it passes through a point, say p_1 of R. Now further expand the disk by moving c in the direction $\boldsymbol{p_1 c}$ until its boundary passes through a second point p_2 of R. The edge e defined by p_1 and p_2 belongs to $\varXi$, and the two extreme disks in the pencil of empty disks through p_1 and p_2 are the disks $D_{\Delta_e^1}$ and $D_{\Delta_e^2}$. Their union covers D, and so D contains ϵn points out of the set P_e. Then the net R_e computed for P_e must hit D, as $\epsilon n = (\epsilon n/|P_e|) \cdot |P_e|$.



For the expected size, clearly, if $\epsilon n < 13$ then the returned set is an ϵ-net of size $\frac{13}{\epsilon}$. Otherwise we can calculate the expected number of points added to the ϵ-net during solving the sub-problems. We simply group them by the number of points in them. Set $E_i = \{e \mid 2^i\epsilon n \leq |P_e| < 2^{i+1}\epsilon n\}$, and let us denote the size of the ϵ-net returned by our algorithm with $f'(\epsilon)$. Then

$$\begin{aligned}\mathbf{E}\left[f'(\epsilon)\right] &= \mathbf{E}[|R|] + \mathbf{E}\Big[\big|\bigcup_{e\in\varXi} R_e\big|\Big] = \frac{c_0}{\epsilon} + \mathbf{E}[|\{e \mid \epsilon n \leq |P_e| < 3\epsilon n/2\}|] \cdot f(2/3) \\ &+ \mathbf{E}[|\{e \mid 3\epsilon n/2 \leq |P_e| < 2\epsilon n\}|] \cdot f\left(\frac{1}{2}\right) + \sum_{i=1} \mathbf{E}\left[\sum_{e\in E_i} f'\left(\frac{\epsilon n}{|P_e|}\right)\right]. \quad (1)\end{aligned}$$

Noting that $\mathbf{E}[\sum_{e\in E_i} f'(\frac{\epsilon n}{|P_e|}) \mid |E_i| = t] \leq t\mathbf{E}[f'(1/2^{i+1})]$, we get

$$\begin{aligned}\mathbf{E}\left[\sum_{e\in E_i} f'\left(\frac{\epsilon n}{|P_e|}\right)\right] &= \mathbf{E}\left[\mathbf{E}[\sum_{e\in E_i} f'\left(\frac{\epsilon n}{|P_e|}\right) |E_i|]\right] \leq \mathbf{E}\left[|E_i| \cdot \mathbf{E}[f'(1/2^{i+1})]\right] \\ &= \mathbf{E}[|E_i|] \cdot \mathbf{E}[f'(1/2^{i+1})] \qquad (2)\end{aligned}$$

as $|E_i|$ and $f'(\cdot)$ are independent. As $\epsilon' = \frac{\epsilon n}{|P_e|} > \epsilon$, by induction, assume $\mathbf{E}[f'(\epsilon')] \leq \frac{13.4}{\epsilon'}$. Then by using Lemma 1 and 2

$$\begin{aligned}\mathbf{E}\left[f'(\epsilon)\right] \leq \frac{c_0}{\epsilon} &+ \frac{(3.1)\cdot c_0^3(c_0+8.34)}{\epsilon e^{c_0}} \cdot 2 + \frac{(3.1)\cdot c_0^3((3/2)^3 c_0 + 14.8)}{\epsilon e^{3c_0/2}} \cdot 10 \\ &+ \sum_i \frac{(3.1)\cdot c_0^3(2^{3i}c_0 + 3.7\cdot 2^{2i+2})}{\epsilon e^{c_0 2^i}} \cdot 13.4 \cdot 2^{i+1} \leq \frac{13.4}{\epsilon} \qquad (3)\end{aligned}$$

by setting $c_0 = 12$. □

Lemma 4. *Algorithm* COMPUTE ϵ-NET *runs in expected time* $O(n \log n)$.

We have shown that the expected size of the returned ϵ-net is $13.4/\epsilon$. Furthermore, by Markov's inequality and repeatedly running the algorithm, an ϵ-net of size $(1+\delta) \cdot 13.4/\epsilon$ is returned in exptected time $O(n/\delta \cdot \log n)$ for any constant $\delta > 0$. Setting δ small enough finishes the proof of Theorem 1.

3 Engineering the Agarwal-Pan Algorithm

The Agarwal-Pan (AP) algorithm (shown in Algorithm 2) 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 until their total weight exceeds a threshold of $cW/\textsc{Opt}$, where c is some constant and $W = \sum_{p \in P} w(p)$ is the current total weight. If after any iteration, all disks have weight above the threshold $\frac{cW}{2e\textsc{Opt}}$, return a $\frac{c}{2e\textsc{Opt}}$-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 $\Theta(\textsc{Opt})$ 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 $1/\textsc{Opt}$-net Q to be returned as part of the hitting set. This ensures that the remaining disks not hit by Q contain less than $n/\textsc{Opt}$ points. Additionally they observe that in any iteration, if less than $\textsc{Opt}$ disks are reweighed, then all disks have weight more than $\frac{cW}{2e\textsc{Opt}}$.

Algorithm 2: AP algorithm for computing hitting sets

```
Data: A point set $P$, a set of disks $\mathcal{D}$, a fixed constant $c$, and the value of $\textsc{Opt}$.
Compute a $(1/\textsc{Opt})$-net, $Q$, of $P$ and remove disks hit by $Q$
Set $w(p) = 1$ for all $p \in P$
repeat
    foreach $D \in \mathcal{D}$ do
        if $w(D) \leq cW/\textsc{Opt}$ then
            reweigh $D$ repeatedly until the weight $w(D)$ exceeds $cW/\textsc{Opt}$
    flag = false
    foreach $D \in \mathcal{D}$ do
        if $w(D) < (c/2e) \cdot W/\textsc{Opt}$ then flag = true
until flag = true
return $Q$ along with a $(c/2e\textsc{Opt})$-net of $P$ with respect to $w(\cdot)$
```

The AP algorithm is simple and has a clever theoretical analysis. 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 use the algorithm for constructing ϵ-nets described in the previous section, which already reduces the approximation factor significantly.

*Removing **A)**.* Just as Agarwal and Pan do, we start by picking a $c_1/\textsc{Opt}$-net, for some constant c_1. The idea for getting rid of range-reporting data-structure is to observe that the very fact that a disk D is not hit by Q, *when* Q *is an* ϵ*-net*, makes it possible to use Q in a simple way to efficiently enumerate the points in D. We will show that D lies in the union of two Delaunay disks in the Delaunay triangulation of Q, which, as we show later, can be found by a simple binary search. The resulting algorithm still has worst-case near-linear running 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 $c_2/\textsc{Opt}$-net R, for some constant c_2. The set of disks that are not hit by R are then guaranteed to have weight at most $c_2W/\textsc{Opt}$, which we can then reweigh during that phase. 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, and $b)$ a disk whose initial weight was less than $c_2W/\textsc{Opt}$ could have its current weight more than $c_2W/\textsc{Opt}$ in the middle of a phase, and so it is erroneously reweighed.

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. Additionally we maintain a nearest-neighbor structure for S, enabling us to only reweigh D if it is not hit by $R \cup S$. Now, if during a phase, there are $\Omega(\textsc{Opt})$ reweighings, then as in the Agarwal-Pan algorithm, we move on to the next phase, and $a)$ is not a problem. Otherwise, there have been less than $\textsc{Opt}$ reweighings, which implies that less than $\textsc{Opt}$ 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 $c_2W/\textsc{Opt}$. Consequently we sometimes waste $O(n/\textsc{Opt})$ time to compute this list of points inside D without performing a reweighing. Due to this, the worst-case running time increases to $O(n^2/\textsc{Opt})$. In practice, this does not happen for the following reason: in contrast to the AP algorithm, our algorithm reweighs any disk *at most once* during a phase. 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).

Algorithm 3: 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 $\textsc{Opt}$.
Compute a $(c_1/\textsc{Opt})$-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 $(c_2/\textsc{Opt})$-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 in $D$ to $S$; update $\Xi(S)$.
until $|S| \leq c_4\textsc{Opt}$
return $\{Q \cup R \cup S\}$
```

See Algorithm 3 for the new algorithm (the data-structure $\Psi(Q)$ will be defined later).

Lemma 5. *The algorithm terminates, $Q \cup R \cup S$ is a hitting set, of size at most $(13.4 + \delta) \cdot \textsc{Opt}$, 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 $c_4 = \delta/3$. By the standard reweighing argument, we know that after t reweighings, we have:

$$\textsc{Opt}\,(1 + c_3)^{\frac{t}{\textsc{Opt}}} \leq n \cdot (1 + \frac{c_2 c_3}{\textsc{Opt}})^t \tag{4}$$

which solves to $t = O(\frac{\textsc{Opt} \log n}{\delta})$. 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 $O(\frac{\textsc{Opt} \log n}{c_4 \textsc{Opt} \delta}) = O(\log n/\delta)$ times.

By Theorem 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 $13.4\textsc{Opt} \cdot \big(1/c_1 + 1/c_2 + c_4/13.4\big) \leq (13.4 + \delta) \cdot \textsc{Opt}$.

Algorithmic details. Computing an ϵ-net takes $O(n \log n)$ time using Theorem 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 6. *Given a disk $D \in \mathcal{D}_1$, the set of points of P contained in D can be reported in time $O(n/\textsc{Opt} \log n)$.*

4 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. The source code can be obtained from the authors' website [1]. For nearest-neighbors and Delaunay triangulations, we use CGAL. It computes Delaunay triangulations in expected $O(n \log n)$ time. To calculate the optimal solution for the hitting set problem we use the IP solver SCIP (with the linear solver SoPlex). Creating the linear program is carried out efficiently by using the Delaunay triangulation of the points for efficient range searching.

Datasets. 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 *KDDCUP04Bio* dataset [1] (*KDDCU* for short) contains the first 2 dimensions of a protein dataset with 145K entries. We have also created a random data set *Gauss9* with 90K points sampled from 9 different Gaussian distributions with random mean and covariance matrices.

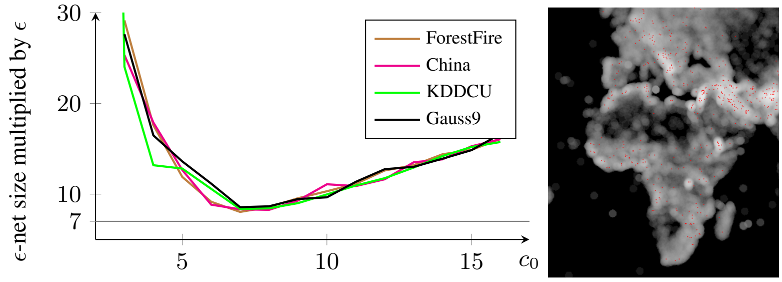


Fig. 1. ϵ-net size multiplied by ϵ for the datasets, $\epsilon = 0.01$ (left) and a subset of the ϵ-net for the *World* dataset (right).

Sizes of ϵ-nets. Setting the probability for random sampling to $\frac{12}{\epsilon \cdot n}$ results in approximately $\frac{12}{\epsilon}$ sized nets for nearly all datasets, as expected by our analysis. We note however, that in practice setting c_0 to 7 gives smaller size ϵ-nets, of size around $\frac{9}{\epsilon}$. See Figure 1 for the dependency of the net size on c_0 for $\epsilon = 0.01$. It also includes an ϵ-net calculated with our algorithm for a subset of the *World* data (red points denote the ϵ-net and each pixel's color is the logarithm of the number of disks it is contained in). See Table 1 for the ϵ-net sizes for different values of ϵ while c_0 is set to 7 and 12. This table

[1] http://perso.esiee.fr/~busn/#hittingsetApplication

also includes the size of the first random sample (R), which shows that the number of subproblems to solve increases as the random sample is more sparse.

Table 1. The size of the ϵ-net multiplied by ϵ (left value in a column for a fixed ϵ) and the size of R, the first random sample multiplied by ϵ (right value in a column) for various point sets with $c_0 = 7$ or 12.

	$c_0 = 7$								$c_0 = 12$							
ϵ	0.2		0.1		0.01		0.001		0.2		0.1		0.01		0.001	
China	7.8	6.6	8.3	6.1	8.28	6.80	8.426	7.090	14.2	14.2	10.6	10.6	12.33	12.33	12.152	12.138
ForestFire	7.4	7.4	8.3	7.3	8.46	7.46	8.522	6.892	13	13	11.6	11.6	12.01	12.01	12.103	12.077
KDDCU	7.4	7.4	8.4	7.4	8.31	7.29	8.343	6.989	10.2	10.2	9.8	9.8	11.65	11.57	12.006	11.978
Gauss9	7.4	5.8	7.8	7.6	8.00	7.18	8.100	6.882	9.8	9.8	12.0	12.0	11.61	11.43	11.969	11.965

Approximate hitting sets. 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. The parameters are set as follows: $c_0 = 10, c_1 = 30, c_2 = 12, c_3 = 2$ and $c_4 = 0.6$.

Our datasets 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 $FIX(0.001)$. The results are shown in Table 2 for two values $r = 0.1$ and $r = 0.01$. 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).

Table 2. Hitting sets. From top to bottom, $RND(0.1)$, $RND(0.01)$.

	# 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	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
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

In Table 3 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 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 3. 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

	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

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).

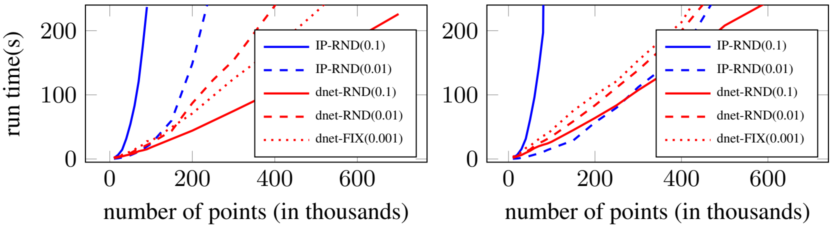


Fig. 2. Different point set sizes for the *ForestFire* (left) and *China* (right) datasets.

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.

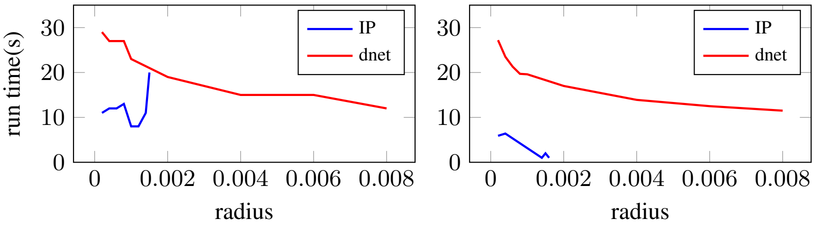


Fig. 3. Different radii settings for the *KDDCU* (left) and *China* (right) datasets.

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 datasets with a more optimized (e.g., multi-threaded) implementation.

References

1. Clustering datasets, http://cs.joensuu.fi/sipu/datasets/.
2. Federal Wildland Fire Occurence Data, United States Geological Survey. http://wildfire.cr.usgs.gov/firehistory/data.html.
3. Geographic Names Database, maintained by the National Geospatial-Intelligence Agency. http://geonames.nga.mil/gns/html/.
4. Pankaj K. Agarwal and Nabil H. Mustafa. Independent set of intersection graphs of convex objects in 2d. *Comput. Geom.*, 34(2):83–95, 2006.
5. Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Symposium on Computational Geometry*, page 271, 2014.
6. Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
7. Norbert Bus, Shashwat Garg, Nabil H. Mustafa, and Saurabh Ray. Improved local search for geometric hitting set. In *32st International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
8. Bernard Chazelle and Joel Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.
9. D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.
10. D. S. Hochbaum and W. Maass. Fast approximation algorithms for a nonconvex covering problem. *J. Algorithms*, 8(3):305–323, 1987.
11. Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.
12. E. Pyrga and S. Ray. New existence proofs for epsilon-nets. In *Proceedings of Symposium on Computational Geometry*, pages 199–207, 2008.