

# Weighted Geometric Set Cover Problems Revisited\*

Sariel Har-Peled<sup>†</sup>      Mira Lee<sup>‡</sup>

December 1, 2008

## Abstract

We study several set cover problems in low dimensional geometric settings. Specifically, we describe a PTAS for the problem of computing a minimum cover of given points by a set of weighted fat objects. Here, we allow the objects to expand by some prespecified  $\delta$ -fraction of their diameter.

Next, we show that the problem of computing minimum weight cover of points by weighted halfplanes (without expansion) can be solved exactly in the plane. We also study the problem of covering  $\mathbb{R}^d$  by weighted halfspaces, and provide approximation algorithms and hardness results. We also investigate the “dual” settings of computing minimum weight simplex that covers a given target point.

Finally, we provide a near linear time algorithm for the problem of solving a LP minimizing the total weight of violated constraints needed to be removed to make it feasible.

## 1 Introduction

Set covering problems are fundamental optimization problems. The general set cover problem is hard to solve, even approximately [Fei98, LY94]. In general, given a set system  $(\mathcal{P}, \mathcal{B})$  with  $n = |\mathcal{P}|$  elements and  $m = |\mathcal{B}|$  weighted sets, one can compute a  $O(\log n)$  approximation to the minimum weight (i.e., cost) of sets of  $\mathcal{B}$  needed to cover  $\mathcal{P}$  [Hoc96]. The generic greedy algorithm that repeatedly picks the set with minimum average weight for the not yet covered elements provides this  $O(\log n)$  approximation, and this approximation can not be improved unless  $\mathcal{P} = \text{NP}$ .

The geometric settings rise naturally in many applications. For example, consider the problem of placing wireless antennas to serve a given set of clients. Every client can be modeled as a point, and every possible placement of an antenna can be represented by the

---

\*The latest version of this paper is available online [HL08].

<sup>†</sup>Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; [sariel@uiuc.edu](mailto:sariel@uiuc.edu); <http://www.uiuc.edu/~sariel/>.

<sup>‡</sup>Department of Computer Science; Korea Advanced Institute of Science and Technology; Gwahangno 335 Yuseong-gu; Daejeon 305-701, Republic of Korea ; [mira@kaist.ac.kr](mailto:mira@kaist.ac.kr)

region it can service. Every antenna has also an associated cost (i.e., weight) of deployment, and we would like to find a minimum weight cover of the points by the given regions. There are of course many other applications where geometric settings of set cover are of interest.

We shortly mention some relevant results to our discussion. Even for geometric settings, most versions of the problem are known (or believed) to be NP-HARD. For example, if  $\mathcal{P}$  is a set of  $n$  points in the plane and  $\mathcal{B}$  is the set of all unit disks in the plane, then finding the (unweighted) minimum cover is NP-HARD [FG88]. Hochbaum and Maas [HM85] used the idea of shifting grids, to show a  $(1 + \varepsilon)$ -approximation algorithm the minimum size cover of a set of points in the plane by a set of unit disks (here, all unit disks are allowable).

The problem becomes considerably harder if one is restricted to use a prespecified finite set of ranges (i.e.,  $\mathcal{B}$ ). In particular, Clarkson [Cla93] developed a reweighting algorithm for approximating a polytope, which solves the associated set cover problem. Later, Brönnimann and Goodrich [BG95] formalized this algorithm in the context of a set system with bounded VC dimension. The algorithm provides a  $O(\log \text{opt})$  approximation, where  $\text{opt}$  is the size of the optimal solution. More recently, Clarkson and Varadarajan [CV07] showed a constant factor approximation if the regions inducing the sets have low union complexity. In fact, their algorithm implies a constant factor approximation for the problem of covering points with a set of disks (out of a set of prespecified disks).

Underlying the difference between the two cases is a stabbing/packing argument. Indeed, if no point is covered by more than a constant number of regions in the optimal solution then one can use standard techniques to get a PTAS. As such, we do not know how to achieve an  $(1 + \varepsilon)$ -approximation for the set cover problem even for the case of unit disks in the plane (where the allowable disks are specified in advance). Furthermore, for the case of weighted unit disks, where every disk has an associated weight, and we are trying to find the minimum weight cover of the points, only a  $O(\log \bar{\omega}(\text{opt}))$  approximation is known [ERS05], where  $\bar{\omega}(\cdot)$  is the total weight of objects.

**Our results.** Motivated by the above, we study some special cases of the weighted set cover problems.

**Expanded cover.** Given a set  $\mathcal{P}$  of  $n$  points and a set  $\mathcal{B}$  of  $m$  weighted fat objects in the plane, we are interested in computing a minimum weight cover of  $\mathcal{P}$  by a subset of  $\mathcal{B}$ . Formally, with each object  $\mathcal{B} \in \mathcal{B}$  we associate a weight  $\bar{\omega}(\mathcal{B}) > 0$ , and we are looking for a subset  $R \subseteq \mathcal{B}$  that **covers**  $\mathcal{P}$ ; that is  $\mathcal{P} \subseteq \bigcup_{\mathcal{B} \in R} \mathcal{B}$ . Furthermore, we are interested in a cover such that the total weight  $\bar{\omega}(R)$  is minimized. (Here we abuse the notation  $\bar{\omega}(\cdot)$  for both the weight of an object and the total weight of the objects in a set.)

As mentioned above, the best known result is a  $O(\log \bar{\omega}(\text{opt}))$  approximation [ERS05]. Since getting a better approximation seems to be quite challenging, we relax the settings as follows: We allow each object to expand by  $\delta$  fraction of its diameter. Note, that for most applications involving such geometric cover (e.g., placement of wireless hotspots to serve clients), this is a reasonable assumption as the region of “acceptable” coverage provided with an antenna degrades slowly and as such the boundaries of this region are blurry to

begin with. Formally, an object  $\mathbf{B}$   $\delta$ -covers a point  $\mathbf{p}$ , if we expand  $\mathbf{B}$  by  $\delta \text{rad}(\mathbf{B})$  then the expanded object covers  $\mathbf{p}$ , where  $\text{rad}(\mathbf{B})$  is the radius of the largest ball enclosed inside  $\mathbf{B}$ .

In Section 2, we provide a PTAS for this problem. For fixed  $\delta > 0$  and  $\varepsilon > 0$ , we show a PTAS that computes a  $\delta$ -cover  $R \subseteq \mathcal{B}$  of  $\mathbf{P}$  such that  $\bar{w}(R) \leq (1 + \varepsilon)\bar{w}(\text{opt})$ , where  $\text{opt}$  is the optimal solution for the original problem (i.e. 0-cover). Note, that the approximation provided is bicriteria, as the solution uses  $\delta$ -expanded objects is a  $(1 + \varepsilon)$ -approximation to the cost of the optimal solution with no expansion.

The solution uses shifted quadtrees and dynamic programming to compute the optimal solution [Aro98]. We also register our objects in the quadtree in such a way that guarantees that they are not being replicated too much by the quadtree. A somewhat similar scheme was used by Erlebach *et al.* [EJS05] for the case of independent set of fat objects. However, it is not clear how to apply their dynamic programming in this case. In particular, we use a new approach based on passing masks through the recursive calls to convey coverage information, which might be of independent interest.

**Exact cover of a point set by weighted halfplanes.** Given a set  $\mathbf{P}$  of points in the plane, and a set  $\mathbf{H}$  of weighted halfplanes, in Section 3, we show how to compute, in polynomial time, the *exact* minimum weight cover of  $\mathbf{P}$  by the halfplanes of  $\mathbf{H}$ . Since computing the minimum weight cover for halfspaces in  $\mathbb{R}^3$  is NP-HARD, and the best approximation known there is  $O(\log \bar{w}(\text{opt}))$ . The dynamic programming is similar to the one used by Ambühl *et al.* [AEMN06]. Ambühl *et al.* [AEMN06] present an exact algorithm for the problem of covering points inside a narrow strip by weighted unit disks, where their centers lie outside the strip. They provide an exact algorithm for this problem which can be modified to solve our problem. Nevertheless, our result is still interesting, as our algorithm is simpler and the problem at hand has additional structure, which we use next.

**Covering  $\mathbb{R}^d$  with halfspaces.** In Section 4, given a weighted set  $\mathbf{H}$  of  $n$  halfspaces in  $\mathbb{R}^d$ , we study the problem of computing the minimum weight set of  $d + 1$  halfspaces that covers  $\mathbb{R}^d$ . The unweighted version of this problem can be solved in linear time, and it can be  $(d+1)$ -approximated in  $O(n \log n)$  time. For the planar case, we show a  $(1+\varepsilon)$ -approximation algorithm for this problem that runs in near linear time.

The  $r$ SUM problem requires one to decide, given a set  $X$  of  $n$  numbers, if there are  $r$  of them that sum up to 0. The problem is believed to require  $\Omega(n^{\lceil r/2 \rceil})$  time [Eri99] to solve. As such, a problem is 3SUM-HARD if solving it in subquadratic time would imply a subquadratic algorithm for 3SUM [GO95].

Interestingly, the problem of computing (exactly) the minimum weight cover of the plane by halfplanes is 3SUM-HARD, and as such, can not (probably) be solved in subquadratic time. In general, it is  $(d + 1)$  SUM-Hard in  $d$  dimensions.

**Point inside cheapest simplex.** Given a set  $\mathbf{P}$  of  $n$  weighted points in  $\mathbb{R}^d$  and a target point  $\mathbf{p}$ , we study in Section 5, the problem of computing the minimum weight set of  $d + 1$  points of  $\mathbf{P}$  that their convex hull (i.e., simplex) covers  $\mathbf{p}$ . This problem is intuitively “dual” to the previous problem, and we get similar results. However, in this case we are able to show that computing a  $(1 + 1/(d - 1))$ -approximation to the minimum weight simplex (defined by

$d + 1$  points of  $P$ ) containing a target point  $p$ , requires  $\Omega(n^{\lceil (d-1)/2 \rceil})$  time (under the  $r$ SUM running time assumption).

**Weighted linear programming with violations.** As a side comment, in Section 6, we revisit the problem of solving linear programming with minimum number of violations. In our variant, each linear constraint has a weight associated with it, and we would like to find a minimum weight set of constraints, so that if we remove them from the LP, it becomes feasible. We provide a near linear time algorithm for this problem, using the techniques of Aronov and Har-Peled [AH08] that provide a similar result for the unweighted case.

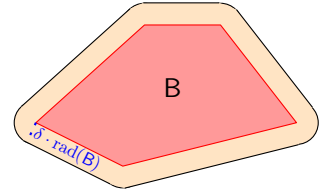
## 2 Expanded Covering by Fat Objects

In this section, we describe a PTAS for computing a  $\delta$ -cover by  $\alpha$ -fat objects. More precisely, the input is a set  $P$  of  $n$  points and a weighted set  $\mathcal{B}$  of  $m$   $\alpha$ -fat objects. We present a polynomial time algorithm that computes a  $\delta$ -cover  $R \subseteq \mathcal{B}$  such that  $\bar{w}(R) \leq (1 + \varepsilon)\bar{w}(\text{opt})$ , where  $\text{opt}$  is the optimal 0-cover.

### 2.1 Preliminaries

A convex body  $B$  is  $\alpha$ -**fat** if  $\text{Rad}(B) \leq \alpha \cdot \text{rad}(B)$ , where  $\text{Rad}(\cdot)$  (resp.  $\text{rad}(\cdot)$ ) denotes the radius of the smallest enclosing (resp. largest enclosed) disk of  $B$ .

**Definition 2.1** For a convex body  $B$ , let its  $\delta$ -**expansion** be the convex region with points in distance at most  $\delta \cdot \text{rad}(B)$  from  $B$ . Formally, the  $\delta$ -expansion of  $B$  is the Minkowski sum of  $B$  with a disk of radius  $\delta \text{rad}(B)$ , see figure on the right. In particular, a point is  $\delta$ -**covered** by  $B$  if it is covered by the  $\delta$ -expansion of  $B$ . As such, if  $p \in B$  then  $B$  **0-covers** (or just covers)  $p$ .



### 2.2 Similarly sized objects

Consider the case where all the objects in  $\mathcal{B}$  are of *similar size*; namely, there exists a constant  $c \geq 1$  such that

$$\text{rad}(B_1) \leq c \cdot \text{rad}(B_2) \quad \text{and} \quad \text{rad}(B_2) \leq c \cdot \text{rad}(B_1),$$

for any  $B_1, B_2 \in \mathcal{B}$ .

**Observation 2.2** Let  $h = \min_{D \in \mathcal{B}} \text{rad}(D)$  and  $H = \alpha c h$ . Then, for any  $B \in \mathcal{B}$ , it holds

$$h \leq \text{rad}(B) \leq \text{Rad}(B) \leq \alpha \cdot \text{rad}(B) \leq \alpha c h = H.$$

That is, any  $B \in \mathcal{B}$  contains a disk with radius  $h$  and is contained in a disk of radius  $H$ .

**Lemma 2.3** *Given a set  $C \subseteq \mathcal{B}$ , there is a subset  $T \subseteq C$ , where any point covered by  $C$  is  $\delta$ -covered by (at least one and) at most  $O(\alpha^2 c^2 / \delta^2)$  objects of  $T$ .*

*Proof:* Let  $G$  denote the grid with sidelength  $\delta h / 10$ . If an object  $\mathbf{B} \in C$  intersects a cell  $\mathbf{g}$  of  $G$ , then the distance between  $\mathbf{B}$  and any point  $\mathbf{p}$  in  $\mathbf{g}$  is at most  $\sqrt{2} \delta h / 10 < \delta \text{rad}(\mathbf{B})$ , and as such  $\mathbf{B}$   $\delta$ -covers all the points in the cell  $\mathbf{g}$ . That is, the  $\delta$ -expansion of  $\mathbf{B}$  properly covers all the cells of  $G$  that intersects  $\mathbf{B}$ .

We compute a subset of  $C$  with the required property. Consider the cells of  $G$  intersecting any object of  $C$ , and let  $C = \{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_l\}$ . Initially, let  $T = \{\mathbf{B}_1\}$ , and mark all the cells that intersect  $\mathbf{B}_1$  as covered. At the  $i$ th step, consider the cells intersecting  $\mathbf{B}_i$ . If all these cells are already marked, then they are (already)  $\delta$ -covered by  $T$ . Otherwise, let  $T = T \cup \{\mathbf{B}_i\}$  and mark all the cells that intersects  $\mathbf{B}_i$  as covered. In the end of this process,  $T$  is a  $\delta$ -cover for  $C$ .

We claim that any point  $\mathbf{p}$  covered by  $C$  is  $\delta$ -covered by at most  $O(\alpha^2 c^2 / \delta^2)$  objects of  $T$ . Indeed, consider the disk  $D$  centered at  $\mathbf{p}$  with radius  $3(1 + \delta)H$ . By Observation 2.2, any body  $\mathbf{B} \in T$  covering  $\mathbf{p}$  is fully contained inside  $D$ . In fact, even the  $\delta$ -expansion of  $\mathbf{B}$  is contained inside  $D$ . By construction, we charged  $\mathbf{B}$  to some grid cell of  $G$  that it intersects such that any grid cell gets charged at most once. As such, the number of objects of  $T$  covering  $\mathbf{p}$  can be bounded by the number of grid cells of  $G$  that intersect  $D$ , which is at most  $O(\alpha^2 c^2 / \delta^2)$ . ■

## 2.3 Handling objects of different sizes

In the following, we assume that the point set  $\mathbf{P}$  and the object set  $\mathcal{B}$  are contained inside the square  $[1/2, 1]^2$ . Let  $(a, b)$  be a random point in the square  $[0, 1/2]^2$  and let  $\Phi$  be the unit square having  $(a, b)$  as its bottom left corner. Note that  $\mathbf{P} \subset \Phi$  and  $\mathcal{B} \subset \Phi$ . For  $\lambda = 2^{-i}$ , let denote by  $\mathcal{G}_\lambda$  the canonical grid of sidelength  $\lambda$  having  $(a, b)$  as the origin, where  $i$  is a non-negative integer number.

In the following, we are going to use the hierarchical grids  $\mathcal{G}_{1/2}, \mathcal{G}_{1/4}, \dots$  to store the objects. For every object, we will store it in the relevant cells in the grid having the “right” resolution. Formally, for  $\mathbf{B} \in \mathcal{B}$ , let

$$\lambda = \lambda(\mathbf{B}) = \text{pow}_2 \left( \text{Rad}(\mathbf{B}) \cdot \frac{100}{\varepsilon} \right),$$

where  $\text{pow}_2(x) = 2^{\lceil \lg x \rceil}$  is the smallest number which is a power of 2 larger than or equal to  $x$ . Let  $\text{cell}(\mathbf{B})$  be the set of all the cells of  $\mathcal{G}_\lambda$  intersecting  $\mathbf{B}$ , and register  $\mathbf{B}$  as being stored in each one of the cells of  $\text{cell}(\mathbf{B})$ . We have  $|\text{cell}(\mathbf{B})| \leq 4$  (in fact, usually this set is a singleton). Overall, the set of grid cells used by the objects is  $\mathbf{X} = \bigcup_{\mathbf{B} \in \mathcal{B}} \text{cell}(\mathbf{B})$  and we have  $|\mathbf{X}| = O(m)$ , where  $m = |\mathcal{B}|$ . Next, build a compressed quadtree  $\mathcal{T}$  having  $\Phi$  as the root node and all the squares of  $\mathbf{X}$  appear in it as nodes. Such a compressed quadtree  $\mathcal{T}$  can easily be constructed using standard techniques in  $O(m \log m)$  time [Har08]. Next, store the points of  $\mathbf{P}$  in the leaves of  $\mathcal{T}$  containing them, and register every object  $\mathbf{B} \in \mathcal{B}$  in

the corresponding nodes to  $\text{cell}(\mathbf{B})$ . If a subtree of  $\mathcal{T}$  does not store any points, then we remove it from  $\mathcal{T}$ . For a subset  $C \subseteq \mathcal{B}$  and a node  $v \in \mathcal{T}$ , we define by  $C_v$  the set of objects in  $C$  registered in the node  $v$ .

**Lemma 2.4** *For any  $\mathbf{B} \in \mathcal{B}$ , we have  $\mathbf{E}\left[|\text{cell}(\mathbf{B})|\right] \leq 1 + \varepsilon$ . In words, the expected number of times an object get replicated in the quadtree is at most  $1 + \varepsilon$ .*

*Proof:* An object  $\mathbf{B}$  is stored in a node whose cell sidelength  $\lambda \geq 100\text{Rad}(\mathbf{B})/\varepsilon$ , and the probability that a horizontal or vertical grid line of  $\mathcal{G}_\lambda$  intersects  $\mathbf{B}$  is at most  $2\text{Rad}(\mathbf{B})/\lambda + 2\text{Rad}(\mathbf{B})/\lambda < \varepsilon/3$ . Hence, we get  $\mathbf{E}\left[|\text{cell}(\mathbf{B})|\right] \leq 1 + 3 \cdot \Pr\left[|\text{cell}(\mathbf{B})| > 1\right] < 1 + \varepsilon$ .  $\blacksquare$

The next lemma testifies that we need to consider only small subsets of  $\mathcal{B}_v$ , where  $\mathcal{B}_v$  are all the objects registered at  $v$ .

**Lemma 2.5** *For a node  $v \in \mathcal{T}$ , let  $C \subseteq \mathcal{B}_v$  be any set of objects, and consider the set cover instance  $I_v = (\mathbf{P}', C)$ , where  $\mathbf{P}'$  is an arbitrary point set covered by  $C$ . Then, there is a set  $R_v \subseteq C$  where any point is  $\delta$ -covered by (at least one and) at most  $O(\alpha^2/\delta^2)$  objects of  $R_v$ .*

*Proof:* By construction of  $\mathcal{T}$ , the objects in  $\mathcal{B}_v$  are similar sized and so are the objects in  $C$ . Therefore the proof of Lemma 2.3 implies the claim.  $\blacksquare$

**Lemma 2.6** *There exists a  $\delta$ -cover  $R \subset \mathcal{B}$  of  $\mathbf{P}$ , such that for every node  $v$  of  $\mathcal{T}$ , we have that at most  $\kappa = O(\alpha^2/(\varepsilon^2\delta^2))$  objects of  $R$  belong to  $\mathcal{B}_v$ ; formally,  $|R \cap \mathcal{B}_v| \leq \kappa$ . Furthermore, we have  $\mathbf{E}\left[\sum_{v \in \mathcal{T}} \mathbf{w}(R_v)\right] \leq (1 + \varepsilon)\bar{\mathbf{w}}(\text{opt})$ , where  $\text{opt}$  is the optimal solution to the cover problem (without expansion).*

*Proof:* We subdivide the square associated with  $v$  in  $\mathcal{T}$  into a grid whose sidelength is

$$u = \text{pow}_2\left(\frac{\delta \varepsilon}{800 \alpha} \ell\right),$$

and let denote this grid by  $\mathcal{G}_v$ , where  $\ell$  is the sidelength of the cell associated with  $v$ . For an object  $\mathbf{B} \in \mathcal{B}_v$ , we have (by construction) that

$$\begin{aligned} \ell = \text{pow}_2\left(\text{Rad}(\mathbf{B}) \cdot \frac{100}{\varepsilon}\right) &\Rightarrow \ell \geq \text{Rad}(\mathbf{B}) \cdot \frac{100}{\varepsilon} \geq \frac{\ell}{2}. \\ \Rightarrow \text{rad}(\mathbf{B}) &\geq \frac{\varepsilon}{2 \cdot 100 \cdot \alpha} \ell \quad \Rightarrow \quad \delta \text{rad}(\mathbf{B}) \geq \frac{\delta \varepsilon}{2 \cdot 100 \cdot \alpha} \ell \geq 2u. \end{aligned}$$

This implies that  $\mathbf{B}$   $\delta$ -covers all the cells of  $\mathcal{G}_v$  that it intersects.

Let  $\text{opt}_v$  denote the subset of the optimal solution stored in  $v$ ; formally,  $\text{opt}_v = \text{opt} \cap \mathcal{B}_v$ . We repeat the argument of Lemma 2.3 to select a “small” subset of  $\text{opt}_v$  that with expansion cover the same cells as  $\text{opt}_v$ , and furthermore, we charge each such object to a grid cell of  $\mathcal{G}_v$ . As before, the resulting set  $R_v$  can be constructed by selecting objects in the order in which they intersect the unmarked cells. Therefore,  $|R_v| = O(\alpha^2/(\varepsilon^2\delta^2))$ . Let  $R = \cup_{v \in \mathcal{T}} R_v \subseteq \text{opt}$ .

As for the second claim, observe that, by linearity of expectation, we have

$$\begin{aligned}
\mathbf{E} \left[ \sum_{v \in \mathcal{T}} w(R_v) \right] &\leq \mathbf{E} \left[ \sum_{v \in \mathcal{T}} w(\text{opt}_v) \right] \leq \mathbf{E} \left[ \sum_{B \in \text{opt}} |\text{cell}(B)| w(B) \right] \\
&\leq \sum_{B \in \text{opt}} w(B) \mathbf{E} \left[ |\text{cell}(B)| \right] \leq \sum_{B \in \text{opt}} (1 + \varepsilon) w(B) \\
&\leq (1 + \varepsilon) w(\text{opt}),
\end{aligned}$$

by Lemma 2.4. ■

## 2.4 The dynamic programming

The structural Lemma 2.6 implies that a good approximation exists that have a very specific structure. As such, we are going to compute it using dynamic programming over the quadtree  $\mathcal{T}$ . Now, when computing the coverage inside a node  $v$ , we need to know the coverage provided (inside this node) by bigger objects (registered higher in the quadtree), and by smaller objects (registered below it in the quadtree). The information about coverage from above would be passed directly from the parent, in the recursive call. Similarly, the coverage from below would be determined by the recursive calls issued by  $v$ . All these sets would be combined into a valid covering set via dynamic programming.

Formally, let  $u$  be the parent of  $v$ . When  $u$  calls  $v$  to compute its coverage, it would also pass it a mask of the cells of  $\mathcal{G}_v$  that are completely covered by objects selected higher in the tree. That is,  $u$  would pass  $v$  a list  $\zeta_v$  (i.e., the mask) of at most  $\kappa$  squares of  $\mathcal{G}_v$  that are covered. As such,  $v$  is going to be called at most  $2^\kappa$  times (with different parameters) to compute the coverage.

Next, when  $v$  is invoked with a given mask, its going to enumerate all possible subsets of  $\mathcal{B}_v$  of size at most  $\kappa$ . For each such subset  $S$ , it computes which cells of  $\mathcal{G}_v$  are intersected (i.e.,  $\delta$ -covered) by  $S$ , combining it with the cells in the list  $\zeta_v$ . Now, for each child  $w$  of  $v$  we perform a recursive call, providing them with the list  $\zeta_w$  of nodes of  $\mathcal{G}_w$  that are already covered.

Computing this list requires us to scan  $\zeta_v$ , and generate the list of the cells of  $\mathcal{G}_w$  covered by the squares of  $\zeta_v$ . Note, that the list  $\zeta_v$  might contain cells outside the region of  $w$ , and furthermore, each square of  $\zeta_v$  covers (at least) 4 nodes of  $\mathcal{G}_w$ , since the grid  $\mathcal{G}_w$  has at most half the sidelength of the grid  $\mathcal{G}_v$  (note, that this grid might have much smaller sidelength if  $v$  is a compressed node). Let  $\zeta_w$  denote the resulting list of cells.

The recursive call to  $w$  with  $\zeta_w$  returns the cost of the covering the points stored in the subtree of  $w$  not yet covered by  $\zeta_w$ . Note, that if such cover is not possible (since, for example, there are some points in  $\mathbf{P}$  stored in this subtree that have to be covered by objects higher in the tree), then the recursive call would return  $+\infty$  as the cost of the coverage.

Next, adding up the costs of all the recursive calls, together with the cost of the objects in  $S$ , yields one possible cover with its associated cost. We perform this for all “small” subsets  $S$ , and return the cheapest cover computed.

**Lemma 2.7** *This algorithm uses  $O(2^\kappa m + n)$  space, and its running time is  $m^{O(\alpha^2/(\delta^2 \varepsilon^2))} n$ .*

*Proof:* For each node  $v \in \mathcal{T}$ , let  $\mathbf{P}_v$  be the set of points stored in the cell  $v$  and  $\mathcal{B}_v$  denotes the set of objects stored in  $v$ . (Note, that  $\mathbf{P}_v$  is not empty only if  $v$  is a leaf.) Since the grid  $\mathcal{G}_v$  associated with  $v$  has at most  $\kappa$  cells, the number of possible subsets (i.e., the number of different combination of cells) that the subtree rooted at  $v$  has to consider is bounded by  $2^\kappa$ . For such a subset of grid cells of  $\mathcal{G}_v$ , we calculate the minimum weight covering only by using the objects stored in the subtree rooted at  $v$ . For this optimal (minimum) calculation,  $O(m^\kappa)$  possible subsets of  $\mathcal{B}_v$  are generated. For each such subset we compute the subset of nodes of  $\mathcal{G}_v$  that this subset  $\delta$ -covers. Thus, for every possible subset of the cells of  $\mathcal{G}_v$ , we have also the cheapest subset of nodes of  $\mathcal{B}_v$  that  $\delta$ -cover this subset. Thus, the dynamic programming stores at  $v$   $O(2^\kappa)$  possible coverings. Since the quadtree  $\mathcal{T}$  has  $O(m)$  nodes, the total required space is  $O(2^\kappa m + n)$ .

Of course, in a leaf  $v$ , one needs to verify whether the current suggested cover indeed covers the given points. If it does not, then the price of the suggested coverage is infinite, otherwise, it is the cost of the objects realizing it.

The running time of the dynamic programming is  $O(mn \cdot 2^\kappa m^{O(\kappa)}) = m^{O(\alpha^2/(\delta^2 \varepsilon^2))} n$ . ■

Putting the above together, we get the following result.

**Theorem 2.8** *Let  $\mathbf{P}$  be a set of  $n$  points and let  $\mathcal{B}$  be a set of  $m$   $\alpha$ -fat objects in the plane. Then, for given  $\delta > 0$  and  $\varepsilon > 0$ , one can compute a  $\delta$ -cover  $R$  of  $\mathbf{P}$  such that  $R \subseteq \mathcal{B}$  and  $\bar{w}(R) \leq (1 + \varepsilon)\bar{w}(\text{opt})$  in  $m^{O(\kappa)} n$  time and  $O(m + n)$  space, where  $\kappa = O(\alpha^2/(\delta^2 \varepsilon^2))$ ,  $\text{opt} \subseteq \mathcal{B}$  is the optimal cover of  $\mathbf{P}$ , and  $\bar{w}(\cdot)$  is the weight of the cover.*

This result can be easily extended to higher dimensions when the dimension  $d$  is constant.

### 3 Exact minimum weight cover by halfplanes

In this section, we present an exact algorithm for computing the minimum weight cover of points in the plane by weighted half-planes. Our approach is somewhat similar to the algorithm of Ambühl *et al.* [AEMN06].

Let  $\mathbf{P}$  be a set of  $n$  points in the plane, and  $\mathbf{H}$  be a set of  $n$  weighted halfplanes. In this section, we show how to compute (in polynomial time) the minimum weight cover of  $\mathbf{P}$  by the halfplanes of  $\mathbf{H}$ . Formally, we are looking for a subset  $\text{opt} \subseteq \mathbf{H}$ , such that  $\mathbf{P} \subseteq \bigcup_{h \in \text{opt}} h$ , and  $\bar{w}(\text{opt}) = \sum_{h \in \text{opt}} \bar{w}(h)$  is minimized, where  $\bar{w}(h) > 0$  is the weight associated with a halfplane  $h$ .

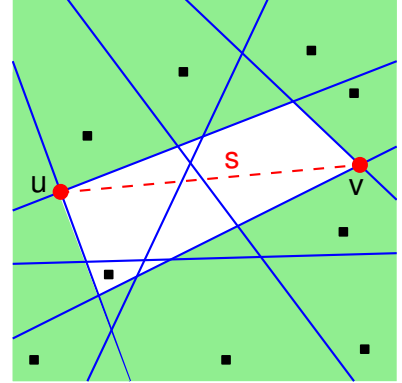
Consider the optimal solution  $\text{opt}$ . If the union of the halfplanes of  $\text{opt}$  cover the whole plane, then the intersection of the complement halfplanes of  $\text{opt}$  is empty. By Helly's theorem, this implies that there are three complement halfplanes of  $\text{opt}$  that their intersection is empty. Namely, there are three halfplanes in  $\text{opt}$  that cover the whole plane. As such, in this case, we can solve the problem by brute force enumeration over all subsets of  $\text{opt}$  of size 3. This takes  $O(n^3)$  time naively, but with a bit of cleverness one can solve this case in  $O(n^2 \log n)$  time.

Otherwise, the hole  $\mathcal{P} = \mathbb{R}^2 \setminus \bigcup_{h \in \text{opt}} \text{int}(h)$  is a non-empty convex (closed) polygon. For the sake of simplicity of exposition, we assume that  $\mathcal{P}$  is a bounded polygon.

Note, that any halfplane  $h \in \text{opt}$  in the optimal solution defines an edge of  $\mathcal{P}$ . indeed, if not, this halfplane is redundant, and one can improve the optimal solution by throwing it away.

Our solution uses dynamic programming to compute  $\mathcal{P}$ . The vertices of  $\mathcal{P}$  are vertices of the arrangement  $\mathcal{A} = \mathcal{A}(\mathbf{H})$  formed by the lines bounding the halfplanes of  $\mathbf{H}$ . As such, we can assume we know the two  $x$ -extreme vertices of  $\mathcal{P}$  by checking all  $O(n^4)$  possibilities.

A pair of such vertices  $u$  and  $v$ , together with the four halfplanes defining them, induces a quadrangle. Consider the diagonal  $s$  connecting these two vertices. The interior of  $s = uv$  lies inside the interior of  $\mathcal{P}$ , and as such no halfplane that splits  $s$  can be part of the optimal solution. Consider the two triangles  $\Delta$  and  $\Delta'$  that have  $s$  as their base, and together cover the quadrangle induced by these two vertices.



Next, solve the set cover recursively in both triangles. To this end, compute the set of points of  $\mathcal{P}$  inside a triangle  $\Delta$ , and all the halfplanes of  $\mathbf{H}$  that intersect  $\Delta$  but do not cover any portion of the base  $s$ . Let  $\mathbf{H}(\Delta, uv)$  denote this set of planes, and let  $\mathbf{H}(\Delta', uv)$  denote the respective set of halfplanes for  $\Delta'$ . Clearly, these two sets are disjoint, and as such we can solve the problem independently on these two subproblems. Such a subproblem requires to cover the point set  $\Delta \cap \mathcal{P}$  with a subset of the halfplanes of  $\mathbf{H}(\Delta, uv)$  with minimum weight.

We now can each such problem recursively. Indeed, consider an edge  $e$  of  $\Delta$  adjacent to  $u$  (a portion of  $e$ , starting at  $u$ , is an edge of  $\mathcal{P}$ ), and guess the vertex  $x$  along  $e$  where the edge of  $\mathcal{P}$  along  $e$  ends. Clearly, this must be a point formed by the intersection of  $e$  with a line induced by a halfplane  $h$  of  $\mathbf{H}(\Delta, uv)$ , and there are (at most)  $n$  such possibilities. The intersection of the complement of  $h$  and  $\Delta$  is a quadrangle, which we split into two triangles in the natural way. Indeed, consider the triangle  $\Delta uvx$ . This triangle, if  $x$  is guessed correctly, is contained inside  $\mathcal{P}$ . As such, if this triangle contains any point of  $\mathcal{P}$ , we know that this is impossible, and we reject this guess for  $x$ . Otherwise, we consider the other triangle  $\Delta''$  forming the quadrangle  $h \cap \Delta$ , and we compute the optimal cover for this triangle recursively. As before, we can throw away any halfplane that splits the base  $ux$  from consideration in the cover of this recursive problem. Thus, the recursive problem we need to solve is to compute minimum weight cover to  $\Delta'' \cap \mathcal{P}$  by the halfplanes of  $\mathbf{H}(\Delta'', ux)$ . The weight of the resulting solution, is the weight of recursive solution together with the weight of  $h$ . We return the minimum weight solution among all such solutions.

The recursive problem as such is defined by a triangle. Such a triangle is defined by the two vertices of  $\mathcal{A}(\mathbf{H})$  defining the base, together with a bit that specifies which one of the two possible triangles are under consideration. As such, there are  $O(n^4)$  possible subproblems being considered.

**Theorem 3.1** *Given a set  $\mathcal{P}$  of  $n$  points in the plane, and a set  $\mathbf{H}$  of  $n$  weighted halfplanes*

(in general position), one can compute in  $O(n^5)$  time the minimum weight cover of  $P$  by the halfplanes of  $H$ .

*Proof:* The algorithm is described above, so we only need to provide the running time analysis. Each recursive subproblem is defined by a triangle. Such a triangle is specified by its two  $x$ -extreme vertices (i.e., the vertices also define the edges of the triangle, and thus also the remaining vertex). Each vertex is defined by two input halfplanes, which implies that there are  $O(n^4)$  subproblems considered by the dynamic program. Next, during the recursive solution, we need to consider  $n$  possible vertices along an edge adjacent to a vertex of the current triangle. By a careful (but tedious) implementation of the recursive algorithm and doing appropriate preprocessing, the overall direct work inside a single recursive call is  $O(n)$ . As such, the overall running time of this algorithm, once we use memoization, is

$$O(n^4 \times n) = O(n^5).$$

■

## 4 Minimum weight cover of $\mathbb{R}^d$ by halfspaces

An interesting subproblem rising from the problem of the previous section, is the computation of a minimum weight cover of  $\mathbb{R}^d$  by a set of halfspaces. As implied by the discussion in the previous section, this problem can be solved in  $O(n^{d+1})$  time by using brute force enumeration.

**Lemma 4.1** *Given a set  $\mathcal{F} = \{\xi_1, \dots, \xi_n\}$  of halfspaces in  $\mathbb{R}^d$ , one can decide if they cover  $\mathbb{R}^d$  in  $O(n)$  time. In this case, one also can compute (also in linear time)  $d + 1$  halfspaces covering  $\mathbb{R}^d$ .*

*Proof:* These halfspaces cover  $\mathbb{R}^d$ , if and only if, the intersection of the complement halfspaces is empty. But deciding if the intersection is empty can be done in linear time by deciding if the associated linear program is feasible or not. Here, every complement halfspace induces a linear inequality in  $d$  variables. Since the resulting linear program is in  $\mathbb{R}^d$  dimensions, it can be solved in linear time [Meg84, MSW96].

If the LP is not feasible, then the LP solver would return us a  $d + 1$  complement halfspaces that have empty intersection (i.e., this is the proof that this LP is infeasible). This implies that the union of the original corresponding  $(d + 1)$  halfspaces cover the whole plane. ■

### 4.1 A $(d + 1)$ -approximation

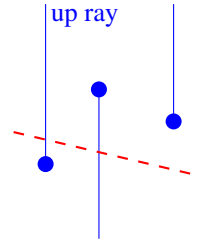
**Lemma 4.2** *Given a set of  $n$  weighted halfspaces in  $\mathbb{R}^d$ , one can compute a  $(d + 1)$ -approximation to the minimum weight cover of  $\mathbb{R}^d$  by the given halfspaces in  $O(n \log n)$  time.*

*Proof:* Let the input be  $n$  halfspaces  $\xi_1, \dots, \xi_n$  in  $\mathbb{R}^d$  with associated weights  $\bar{\omega}_1, \dots, \bar{\omega}_n$ , respectively. Assume that we had sorted the halfspaces in ascending order of weight.

Consider the first index  $k$  for which  $\xi_1, \dots, \xi_k$  covers  $\mathbb{R}^d$ . Using a binary search over the prefix of halfspaces and the algorithm of Lemma 4.1, we can compute  $k$  in  $O(n \log n)$  time. Now, any cover of  $\mathbb{R}^d$  must contain at least one halfspace of weight  $\bar{\omega}_k$  or more. On the other hand, Lemma 4.1 returns a cover of  $\mathbb{R}^d$  with weight at most  $(d + 1)\bar{\omega}_k$ . Thus, we get a  $(d + 1)$  approximation. ■

## 4.2 An $(1 + \varepsilon)$ -approximation for the plane

Consider the dual settings in the plane. Here a halfplane becomes a vertical ray (either downward or upward). A **blocking triplet** of rays intersects any non-vertical line in the plane. As such, the dual problem is to compute the blocking triplet of rays of minimum total weight. We will refer to an upward (resp., downward) vertical ray as an **up ray** (resp. **down ray**).

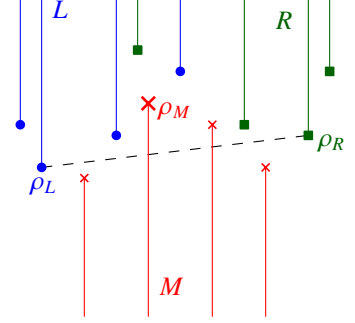


Given a set of up rays, we preprocess it by first building a balanced binary tree  $\mathcal{T}$  on the points defining the rays, where the points are sorted by their  $x$  coordinate. Next, for a node  $v$  in this tree, we compute and store the convex-hull of the point set  $P_v$  of points stored in the subtree of  $v$  in  $\mathcal{T}$ . Thus, given a vertical slab, we can in logarithmic time, return a logarithmic number of sets that their union form the points of  $P$  inside this slab, and furthermore, we also can access the convex-hulls of these  $O(\log n)$  sets efficiently.

**Lemma 4.3** *Let  $L$  and  $R$  be two sets of up rays preprocessed as above, of size at most  $n$ . Let  $M$  be a set of down rays of size at most  $m$ . Then, one can decide, in  $O(m \log^3 n)$  time, if there exists a blocking triple of rays, such that the blocking triplet has one ray from each of these sets.*

*Proof:* Clearly, if there is such a triplet, it must be formed by two upward rays (take from  $L$  and  $R$ ), and the downward ray (of  $M$ ) must lie in the middle between these two rays. As such, let us consider a down ray  $\rho_M \in M$ , and assume that the required upward ray  $\rho_L$  to its left is taken from  $L$ , and as such the up ray to its right  $\rho_R$  (all forming the desired triplet) must belong to  $R$ . (The other possibility,  $\rho_L \in R$  and  $\rho_R \in L$ , is handled in a symmetric fashion.)

Clearly, the segment connecting the endpoint of  $\rho_L$  to the endpoint of  $\rho_R$  must intersect the ray  $\rho_M$ , and this is a required and sufficient condition for these three rays to intersect any non-vertical line in the plane. As such, consider the points of  $L$  to the left of  $\rho_M$  (denote this point set by  $U$ ), and all the points of  $R$  to the right of  $\rho_M$  (denote this point set by  $V$ ). Consider the convex hulls  $\mathcal{CH}(U)$  and  $\mathcal{CH}(V)$ , and their common convex hull  $X = \mathcal{CH}(U \cup V)$ . The polygon  $X$  is formed by adding two bridges to connect the two convex hulls together, and let  $s$  denote the lower bridge. Clearly, there exists a triplet as required, only if the bridge  $s$  intersects the ray  $\rho_M$ .



Given two convex hulls (represented in an array or a binary tree), we can compute their common bridge in  $O(\log n)$  time. Since our data-structure enable us to retrieve  $U$  and  $V$  as a collection of  $O(\log n)$  convex-hulls, we can find the desired bridge by computing a bridge for all these  $O(\log^2 n)$  pairs, and return the one with the lowest intersection point with  $\rho_M$ .

Thus, given a downward ray  $\rho_M \in M$ , we can decide if there is a ray  $\rho_L \in L$  and  $\rho_R \in R$  such that these rays form a blocking triplet in  $O(\log^3 n)$  time. Now, we apply this algorithm to all the rays of  $M$ , and return a blocking triplet, if such a triplet was found. ■

**Lemma 4.4** *Given a weighted set  $H$  of  $n$  halfplanes, one can compute  $(1 + \varepsilon)$ -approximation to the minimum weight cover of the plane by three halfplanes of  $H$  in  $O((n/\varepsilon^2) \log^3 n)$  time.*

*Proof:* Let  $\beta$  be a 3-approximation to the minimum cost of a blocking triplet, computed in linear time using the algorithm of Lemma 4.1 (i.e., the cost of the optimal solution is between  $\beta/3$  and  $\beta$ ). We partition the given set of rays into groups, where  $R_0$  contains all the rays of cost smaller than  $x = \varepsilon\beta/9$ , and  $R_i$  is the set of rays (in the given set of rays) with cost in the range  $x + i[0, x]$ , for  $i = 1, \dots, M$ , where  $M = 9/\varepsilon = O(1/\varepsilon)$ . Now enumerate over all such possible triplets of such sets  $i, j, k$ , and for each such triplet  $R_i, R_j, R_k$ , use the algorithm of Lemma 4.3 to compute a blocking triplet of rays from these three sets of ray. The algorithm returns the cheapest blocking triplet computed.

As for the quality of approximation, observe that the optimal solution  $\text{opt}$  costs at least  $\beta/3$ , and the algorithm returns a set of halfplanes of cost at most  $\text{opt} + 3x \leq (1 + \varepsilon)\text{opt}$ , implying the claim. As for the running time, observe that every halfplane (i.e., ray) participates in  $O(1/\varepsilon^2)$  applications of Lemma 4.3, implying the bound on the running time. ■

### 4.3 Lower bound in the plane

In the **3SUM** problem, you are given three sets of (non-negative) integers  $X, Y, Z$ , each of size  $n$ , and one has to decide if there are numbers  $x \in X, y \in Y$  and  $z \in Z$  such that  $x + y = z$ . It is believed that any algorithm that solves this problem requires  $\Omega(n^2)$  time, and there is a class of **3SUM-HARD** problems, such that if one solves any of them in subquadratic time, it would imply a subquadratic algorithm for the **3SUM** problem [GO95],

Given an instance of **3SUM**, let  $\alpha$  be a large enough number, such that  $\alpha/2$  is larger than any number in this instance. We generate three sets of rays from this instance. First, let

$$L = \left\{ \rho_X(x) \mid x \in X \right\}, \quad M = \left\{ \rho_Z(z) \mid z \in Z \right\}, \quad \text{and} \quad R = \left\{ \rho_Y(y) \mid y \in Y \right\}.$$

Here,  $\rho_X(x)$  is a up ray with apex at  $(-1, x)$ , and the weight of this ray is  $\alpha - x$ . Similarly,  $\rho_Y(y)$  is a up ray with apex at  $(1, y)$  with weight  $\alpha - y$ . Finally,  $\rho_Z(z)$  is a down ray with apex at  $(0, z/2)$  with weight  $z$ .

**Lemma 4.5** *There is a blocking triplet in  $L \cup M \cup R$  of weight  $2\alpha$  if and only if there are  $x \in X, y \in Y$  and  $z \in Z$  such that  $x + y = z$ . Furthermore, the price of any blocking triplet is at least  $2\alpha$ .*

*Proof:* The proof is easy and we include it for the sake of completeness.

If there is  $x, y, z$  such that  $x + y = z$ , then clearly, the triplet  $\rho_X(x), \rho_Y(y)$  and  $\rho_Z(z)$  are a blocking triplet. Indeed,  $\rho_X(x)$  and  $\rho_Y(y)$  have their apexes at  $(-1, x)$  and  $(1, y)$ , respectively. As such, the segment connecting them intersect the  $y$ -axis at the point  $(0, (x + y)/2)$ , which is just  $\rho_Z(z)$ . Since  $\rho_Z(z)$  is a down ray, and  $\rho_X(x), \rho_Y(y)$  are up rays, it follows that this triplet is a blocking triplet, and its weight is  $(\alpha - x) + (\alpha - y) + z = 2\alpha$ .

As for the other direction, clearly a blocking triplet must have a ray from each set  $L, M$  and  $R$ , and assume their apices are at  $(-1, x'), (0, z'/2)$ , and  $(1, y')$ , respectively. Since this is a blocking triplet, we must have that  $(0, z'/2)$  is above the segment connecting  $(-1, x')$  and  $(1, y')$ . That is,  $z'/2 \geq (x' + y')/2$ . On the other hand, the total cost of these three rays is  $2\alpha = \alpha - x' + \alpha - y' + z'$ . This implies that  $x' + y' = z'$ , where  $x' \in X, y' \in Y$  and  $Z' \in Z$ , as required.

As for the second claim, by the above, if the triplet is blocking we must have  $z' \geq x' + y'$ . This implies that the cost of any blocking triplet is  $\alpha - x' + \alpha - y' + z' \geq 2\alpha$ . ■

Thus, computing a blocking triplet of minimum cost is equivalent to **3SUM**.

**Lemma 4.6** *Given a set of weighted halfplanes, computing the cheapest cover of the plane by these halfplanes is 3SUM-HARD.*

As such, solving this problem exactly requires at least quadratic time. Notice, that if we allow negative weights, then for  $\alpha = 0$ , the target weight is 0 (but all blocking triplets have non-negative weight), which implies that no multiplicative approximation is possible in this case, since we could use the approximation algorithm in this case to decide if there exists a solution of total weight 0, which would imply that there exists an exact solution with this weight.

## 4.4 Lower bound in higher dimensions

In the following, let  $\Delta_d$  denote the vertices of the *regular simplex* (of edge length 1). Formally,  $\Delta_d$  is a set of  $d + 1$  points in  $\mathbb{R}^d$  that are: (i) in general position, (ii) their center of mass is in the origin, and (iii) the distance of every pair of them is exactly 1.

The above construction can be extended to higher dimensions. In the  $r$ SUM problem, for  $r \geq 3$ , given a set  $X$  of  $n$  numbers, we would like to decide if there are  $r$  numbers in this set, such that their sum is zero (you are allowed to use the same number several times). It is believed that solving the  $r$ SUM problem requires  $\Omega(n^{\lceil r/2 \rceil})$  time. As such, we get the following result.

**Lemma 4.7** *Given a set of weighted halfspaces in  $\mathbb{R}^d$  computing the (exact) minimum weight cover of  $\mathbb{R}^d$  by these halfspaces requires as much time as solving the  $(d+1)$  SUM problem. As such, it requires  $\Omega(n^{\lceil d/2 \rceil})$  time (under the  $r$ SUM running time assumption).*

*Proof:* Let  $X$  be an instance of  $r$ SUM containing  $n$  numbers, and  $\alpha = 2(d+2) \max_{x \in X} |x|$ . Take the simplex  $\Delta_{r-2}$  in  $\mathbb{R}^{r-2}$ , and consider its  $r-1$  vertices and its center of mass (which is the origin). Next, we replicate the set  $X$  along  $r-1$  vertical lines in  $\mathbb{R}^{r-1}$  that are parallel to the  $(r-1)$ th axis and pass through the vertices of  $\Delta_{r-2}$ , and assign them appropriate weights. Formally, we create the set of points

$$U = \left\{ \mathbf{r} \mid \begin{array}{l} \mathbf{p} \in \Delta_{r-2}, x \in X, \\ \mathbf{r} = (\mathbf{p}, x) \text{ and } \mathbf{w}(\mathbf{r}) \leftarrow (\alpha - x) \end{array} \right\},$$

where each point  $\mathbf{r} \in U$  induces an upward ray. Similarly, we define the set of points

$$D = \left\{ \mathbf{r} = \left( 0, \dots, 0, -\frac{x}{r-1} \right) \mid x \in X, \text{ and let } \mathbf{w}(\mathbf{r}) \leftarrow \alpha - x \right\},$$

each one of them induces a downward ray. Our instance is the set of rays induced by  $Z = U \cup D$ .

Consider a blocking triplet in  $Z$ . It must be made out of  $r-1$  upward rays from  $U$ ; namely,  $(\mathbf{p}_1, x_1), \dots, (\mathbf{p}_{r-1}, x_{r-1})$  each located on the vertical line passing through the vertices of  $\Delta_{r-2}$ , and one downward ray with apex at  $\mathbf{r} = (0, \dots, 0, -\frac{x_r}{r-1})$ . It must be that  $\mathcal{CH}((\mathbf{p}_1, x_1), \dots, (\mathbf{p}_{r-1}, x_{r-1}))$  intersects the vertical line through the origin (that contains the ray of  $\mathbf{r}$ ), and furthermore, the coordinates of this intersection point is

$$\mathbf{s} = \left( 0, \dots, 0, \frac{\sum_{i=1}^{r-1} x_i}{r-1} \right),$$

since the origin is the center of mass of  $\Delta_{r-2}$ . For this to be a blocking triplet, it must be that  $\mathbf{s}$  lies below  $\mathbf{r}$ . Namely, we have the additional constraint that

$$\frac{\sum_{i=1}^{r-1} x_i}{r-1} \leq -\frac{x_r}{r-1} \quad \Rightarrow \quad x_1 + x_2 + \dots + x_r \leq 0.$$

Now, the total weight of this blocking triplet is

$$\sum_i (\alpha - x_i) = r\alpha - \sum_{i=1}^r x_i.$$

As such, there is a blocking triplet in  $Z$  of total weight  $r\alpha$ , if and only if, the given instance  $r$ SUM is satisfiable. Furthermore, the cost of any blocking triplet is at least  $r\alpha$ .

Thus, computing the minimum weight blocking triplet of vertical rays, in  $\mathbb{R}^{r-1}$ , is equivalent to solving the  $r$ SUM problem.

Mapping this problem to the primal, we get a set of weighted halfspaces in  $\mathbb{R}^{r-1}$  such that computing the lowest weight cover of  $\mathbb{R}^{r-1}$  by these halfspaces is equivalent to solving the original  $r$ SUM problem. ■

## 5 Point inside a simplex

One possible “dual” to the problem studied in the previous section seems to be the following natural problem. Given a set of weighted points  $P \subseteq \mathbb{R}^d$ , and a target point  $\mathbf{p}$ , compute the cheapest  $d + 1$  points of  $P$  such that the simplex they form (i.e., their convex-hull) contains  $\mathbf{p}$ . (Note, that by Carathéodory’s theorem since the costs are non-negative, the optimal solution must be formed by at most  $d + 1$  points.)

This problem can easily be solved in linear time in the unweighted case, by writing down a linear program for finding the hyperplane separating  $\mathbf{p}$  from the points of  $P$ . If such a separating hyperplane does not exist, the LP would return  $d + 1$  points that their convex-hull contains  $\mathbf{p}$ . This implies the following.

**Lemma 5.1** *Let  $P$  be a set of  $n$  weighted points in  $\mathbb{R}^d$ . One can compute in  $O(n \log n)$  time a  $(d + 1)$ -approximation to the minimum weight simplex (defined by  $d + 1$  points) of  $P$  containing  $\mathbf{p}$ .*

A similar algorithm to Lemma 4.4 can be applied in to this case. The main difference is that instead of sorting the points by their  $x$ -axis, we need to sort them angularly around  $\mathbf{p}$ . The modifications are easy, and we only state the result.

**Lemma 5.2** *Given a weighted set  $H$  of  $n$  points in the plane, and a target point  $\mathbf{p}$ , one can compute  $(1 + \varepsilon)$ -approximation to the minimum weight triangle (with vertices in  $P$ ) containing  $\mathbf{p}$  in  $O((n/\varepsilon^2) \log^3 n)$  time.*

As for the lower bound, we will modify the construction of Lemma 4.7.

**Lemma 5.3** *Given a set  $P$  of  $n$  weighted points in  $\mathbb{R}^d$  computing the (exact) minimum weight simplex (defined by  $d + 1$  points of  $P$ ) containing a target point  $\mathbf{p}$ , requires  $\Omega(n^{\lceil d/2 \rceil})$  time (under the  $r$ SUM running time assumption).*

*Proof:* For the sake of simplicity of exposition we assume  $\mathbf{p}$  is the origin.

Consider an instance  $X$  of  $d$ SUM, and let  $\Delta_{d-1}$  be the  $(d - 1)$ -simplex (with  $d$  vertices) lying on the hyperplane  $x_d = 0$ , and having the origin as its center of mass. As before, we erect  $d$  vertical lines (parallel to the  $d$ th axis) through the vertices of  $\Delta_{d-1}$ , and sprinkle on these lines  $d$  copies of  $X$ . Let  $Q$  be the resulting set of points. Clearly, there is a  $d$ SUM in

the original instance, if and only if, there are  $d$  points of  $\mathbf{Q}$  that their convex-hull contains the origin (note, that these are  $d$  points and not  $d + 1$ ).

Thus, let  $\alpha = 4 \max_{x \in X} |x|$ , and assign every point  $(\mathbf{q}, x) \in \mathbf{Q}$  the weight  $\alpha - x$ , where  $\mathbf{q}$  is a vertex of  $\Delta_{d-1}$ , and  $x \in X$ . Finally, add an extra point  $(0, \dots, 0, 10\alpha)$  with weight 0. Clearly, there is a simplex of total weight  $d\alpha$  containing the origin, if and only if there is a  $d$ SUM in the original instance  $X$ . ■

Interestingly, we can even prove that its  $r$ SUM-hard to approximate the optimal solution within a constant factor.

**Lemma 5.4** *Given a set  $\mathbf{P}$  of  $n$  weighted points in  $\mathbb{R}^d$  computing  $1+1/(d-1)$ -approximation to the minimum weight simplex (defined by  $d + 1$  points of  $\mathbf{P}$ ) containing a target point  $\mathbf{p}$ , requires  $\Omega(n^{\lceil (d-1)/2 \rceil})$  time (under the  $r$ SUM running time assumption).*

*Proof:* As before, we assume that the target point is the origin.

Let  $X$  be an instance of  $(d - 1)$  SUM. As before, we can generate from  $X$  a set  $\mathbf{P}$  of points in  $\mathbb{R}^{d-1}$  that contains the origin if and only if there are  $d - 1$  points in  $\mathbf{Q}$  that their convex hull contain the origin. We observe that since the coordinates of the points of  $\mathbf{P}$  are integers, one can compute a threshold  $\delta$ , such that either such a  $(d - 2)$ -simplex (with  $d - 1$  vertices) contains the origin, or alternatively the distance of this simplex from the origin is at least  $\delta$ .

Thus, consider the point set

$$\mathbf{P} = \left\{ (\mathbf{p}, -1) \mid \mathbf{p} \in \mathbf{Q} \right\} \cup \left\{ ((\delta/2, 0 \dots, 0, 1)), ((-\delta/2, 0 \dots, 0, 1)) \right\}.$$

Its now straightforward to verify that a  $d$ -simplex (with  $d$  vertices) of  $\mathbf{P}$  contains the origin, and this simplex uses the two new extra vertices, if and only if the original point-set has a triangle containing the origin. So, let us assign weight 0 to the two extra points, and weight 1 to all other points. Clearly, there is a solution of weight  $d - 1$  to this problem if and only if the given  $(d - 1)$  SUM problem have a solution. Furthermore, any other solution has weight at least  $d$ , implying the claim. ■

Since the  $r$ SUM problem is NP-HARD when  $r$  is part of the input, this implies that in high dimension computing the minimum weight simplex containing a point is NP-HARD.

## 6 Linear programming with weighted violations

Let  $L$  be a linear programming with  $n$  constraints in  $\mathbb{R}^d$ . The given LP  $L$  might not be feasible, and we look for a solution that minimizes the total weight of the constraints it violates. Here, we assume has that each constraint has weight associated with it. The problem is thus to find a set of constraints of minimum weight, such that remove these constraints results in a feasible LP.

**Theorem 6.1** *Given a linear program  $L$  in  $\mathbb{R}^d$  with  $n$  constraints, with weight associated with each constraint, and  $\varepsilon > 0$ , one can compute a set of constraints  $S$ , such that the total*

weight of the constraints in  $S$  is at most  $(1 + \varepsilon)\bar{w}(\text{opt})$ , such that the LP  $L \setminus S$  is feasible (i.e.,  $L$  after removing the constraints of  $S$ ), and  $\bar{w}(\text{opt})$  is the minimum weight of a set that makes  $L$  feasible. The running time of the algorithm is  $O(n\varepsilon^{-2(d+1)} \log^{d+2} n)$ . The algorithm succeeds with high probability.

*Proof:* As before, we sort the constraints by their weights, and let  $\mathbf{h}_1, \dots, \mathbf{h}_n$  be the constraints sorted in *decreasing* order by their weight, where  $\bar{w}_i = \bar{w}(\mathbf{h}_i)$ , for  $i = 1, \dots, n$ . Given a set of linear constraints we can decide if it is feasible in linear time, using a LP solver in low dimension [Meg84]. Thus, using binary search, we can find, in  $O(n \log n)$  time, the (largest)  $k$  such that  $\mathbf{h}_1, \dots, \mathbf{h}_k$  is feasible; namely,  $\mathbf{h}_1, \dots, \mathbf{h}_{k+1}$  form an infeasible set of constraints.

Now, the optimal solution has weight at least  $\alpha = \bar{w}_{k+1}$ , since the optimal solution must throw out one of the first  $k + 1$  constraints. On the other hand, the solution throwing away all the constraints after the  $k$ th constraint has weight at most  $\beta = (n - k)\bar{w}_{k+1}$ . We conclude that we had computed a  $n$  approximation of the optimal solution. Next we renormalize the weights; specifically, we set

$$\bar{\tau}_i = \left\lceil \frac{\min(\bar{w}_i, \beta)}{\varepsilon\alpha/4n} \right\rceil.$$

Clearly, every halfplane is assigned a weight which is an integer in the range 1 to  $O(n^2/\varepsilon)$ . Furthermore, solving the linear programming with weight violations with the new weights provide an  $(1+\varepsilon)$ -approximation to the original problem, since the maximum error introduced in the weight renormalization is at most of size  $n(\varepsilon\alpha/4n)$ , which is smaller than  $\varepsilon\bar{w}(\text{opt})$ , where  $\bar{w}(\text{opt}) \geq \alpha$  is the weight of the optimal solution.

Now, we solve the instance by replicating the  $i$ th constraint  $\bar{\tau}_i$  times. We do it “virtually” by using multiplicities on the constraints. Now, we are back to solving linear programming with unweighted violations. Solving such a problem can be expensive, but a  $(1 + \varepsilon)$ -approximation can be done efficiently [AH08]. For the sake of completeness, we outline the idea behind the algorithm of Aronov and Har-Peled. Let us assume that we guess that the optimal solution violates  $k$  constraints. Then, if we pick each constraint with probability  $O(\varepsilon^{-2}(\log n)/k)$  then with high probability, in the sample, the optimal solution would have only  $O(\varepsilon^{-2} \log n)$  constraints it violates, and this estimate is reliable. As such, we can now apply standard linear programming with violations to this sample. If the estimate  $k$  is too large, we shrink it by a factor of two, and repeat the above algorithm.

In our case, since we have only  $n$  real constraints, each call to the (exact) linear programming with violations solver takes  $O(nk^{d+1})$  time [Mat95], where  $k = O(\varepsilon^{-2} \log n)$ . Putting everything together, the resulting algorithm has running time  $O(nk^{d+1} \log n)$ . ■

## Acknowledgments

The authors would like to thank Chandra Chekuri for insightful discussions, and suggesting the problem of covering by expansion. The authors also thank Ken Clarkson, Jeff Erickson,

and Mihai Patrascu on useful discussion on the problems studied in this paper. The authors also thank the anonymous referees for useful comments on an earlier version of this paper.

## References

- [AEMN06] C. Ambühl, T. Erlebach, M. Mihalák, and M. Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *The 10th Intl. Work. Approx. Algs. Combin. Opt. Problems*, pages 3–14, 2006.
- [AH08] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [Aro98] S. Arora. Polynomial time approximation schemes for euclidean TSP and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, Sep 1998.
- [BG95] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14:263–279, 1995.
- [Cla93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3th Workshop Algorithms Data Struct.*, volume 709 of *Lect. Notes in Comp. Sci.*, pages 246–252. Springer-Verlag, 1993.
- [CV07] K. L. Clarkson and K. R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete Comput. Geom.*, 37(1):43–58, 2007.
- [EJS05] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005.
- [Eri99] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28:1198–1214, 1999.
- [ERS05] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the vc-dimension is small. *Inform. Process. Lett.*, 95(2):358–362, 2005.
- [Fei98] U. Feige. A threshold of  $\ln n$  for approximating set cover. *J. Assoc. Comput. Mach.*, 45:634–652, 1998.
- [FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.
- [GO95] A. Gajentaan and M. H. Overmars. On a class of  $O(n^2)$  problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.
- [Har08] S. Har-Peled. Geometric approximation algorithms. Class notes. Online at <http://uiuc.edu/~sariel/teach/notes/aprx/>, 2008.

- [HL08] S. Har-Peled and M. Lee. Weighted geometric set cover problems revisited. [http://www.uiuc.edu/~sariel/papers/08/expand\\_cover/](http://www.uiuc.edu/~sariel/papers/08/expand_cover/), 2008.
- [HM85] D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *J. Assoc. Comput. Mach.*, 32:130–136, 1985.
- [Hoc96] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1996.
- [LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. Assoc. Comput. Mach.*, 41(5):960–981, 1994.
- [Mat95] J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14:365–384, 1995.
- [Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31:114–127, 1984.
- [MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.