Applications of Incidence Bounds in Point Covering Problems*

Peyman Afshani[1], Edvin Berglin[2], Ingo van Duijn[3], and Jesper Sindahl Nielsen[4]

1 MADALGO, Department of Computer Science, Aarhus University, Denmark
peyman@cs.au.dk
2 MADALGO, Department of Computer Science, Aarhus University, Denmark
berglin@cs.au.dk
3 MADALGO, Department of Computer Science, Aarhus University, Denmark
ivd@cs.au.dk
4 MADALGO, Department of Computer Science, Aarhus University, Denmark
jasn@cs.au.dk

Abstract

In the *Line Cover* problem a set of n points is given and the task is to cover the points using either the minimum number of lines or at most k lines. In *Curve Cover*, a generalization of *Line Cover*, the task is to cover the points using curves with d degrees of freedom. Another generalization is the *Hyperplane Cover* problem where points in d-dimensional space are to be covered by hyperplanes. All these problems have kernels of polynomial size, where the parameter is the minimum number of lines, curves, or hyperplanes needed.

First we give a non-parameterized algorithm for both problems in $\mathcal{O}^*(2^n)$ (where the $\mathcal{O}^*(\cdot)$ notation hides polynomial factors of n) time and polynomial space, beating a previous exponential-space result. Combining this with incidence bounds similar to the famous Szemerédi-Trotter bound, we present a *Curve Cover* algorithm with running time $\mathcal{O}^*\left((Ck/\log k)^{(d-1)k}\right)$, where C is some constant. Our result improves the previous best times $\mathcal{O}^*\left((k/1.35)^k\right)$ for *Line Cover* (where $d = 2$), $\mathcal{O}^*\left(k^{dk}\right)$ for general *Curve Cover*, as well as a few other bounds for covering points by parabolas or conics. We also present an algorithm for *Hyperplane Cover* in $\mathbb{R}^3$ with running time $\mathcal{O}^*\left((Ck^2/\log^{1/5} k)^k\right)$, improving on the previous time of $\mathcal{O}^*\left((k^2/1.3)^k\right)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Point Cover, Incidence Bounds, Inclusion Exclusion, Exponential Algorithm

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.60

1 Introduction

In the *Line Cover* problem a set of points in $\mathbb{R}^2$ is given and the task is to cover them using either the minimum number of lines, or at most k lines where k is given as a parameter in the input. It is related to *Minimum Bend Euclidean TSP* and has been studied in connection with facility location problems [8, 17]. The *Line Cover* problem is one of the few low-dimensional geometric problems that are known to be NP-complete [17]. Furthermore *Line Cover* is APX-hard, i.e., it is NP-hard to approximate within factor $(1+\varepsilon)$ for arbitrarily

* Research funded by MADALGO, Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84

32nd International Symposium on Computational Geometry (SoCG 2016).
Editors: Sándor Fekete and Anna Lubiw; Article No. 60; pp. 60:1–60:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

small ε [15]. Although NP-hard, *Line Cover* is fixed-parameter tractable when parameterized by its solution size k so any solution that is "not too large" can be found quickly.

One generalization of the *Line Cover* problem is the *Hyperplane Cover* problem, where the task is to use the minimum number of hyperplanes to cover points in d-dimensional space. Another generalization is to cover points with algebraic curves, e.g. circles, ellipses, parabolas, or bounded degree polynomials. These can be categorized as covering points in an arbitrary dimension space using algebraic curves with d degrees of freedom and at most s pairwise intersections. We call this problem *Curve Cover*. The first parameterized algorithm that was presented for *Line Cover* runs in time $\mathcal{O}^*\big(k^{2k}\big)$ [16] (where $\mathcal{O}^*(.)$ hides polynomial factors). This algorithm generalizes to generic settings, such as *Curve Cover* and *Hyperplane Cover*, obtaining the running time $\mathcal{O}^*\big(k^{dk}\big)$ where d is the degree of the freedom of the curves or the dimension of the space for hyperplane cover.

The first improvement to the aforementioned generic algorithm reduced the running time to $\mathcal{O}^*\big((k/2.2)^{dk}\big)$ for the *Line Cover* problem [10]. The best algorithm for the *Hyperplane Cover* problem, including *Line Cover*, runs in $\mathcal{O}^*\big(k^{(d-1)k}/1.3^k\big)$ time [22]. A non-parameterized solution to *Line Cover* using dynamic programming has been proposed with both time and space $\mathcal{O}^*(2^n)$ [4], which is time efficient when the number of points is $\mathcal{O}(k \log k)$. Algorithms for parabola cover and conic cover appear in [21], running in time $\mathcal{O}^*\big((k/1.38)^{(d-1)k}\big)$ and $\mathcal{O}^*\big((k/1.15)^{(d-1)k}\big)$ respectively.

Incidence Bounds. Given an arrangement of n points and m lines, an *incidence* is a point-line pair where the point lies on the line. Szemerédi and Trotter gave an asymptotic (tight) upper bound of $\mathcal{O}\big((nm)^{2/3} + n + m\big)$ on the number of incidences in their seminal paper [20]. This has inspired a long list of similar upper bounds for incidences between points and several types of varieties in different spaces, e.g. [7, 9, 18, 19].

Our Results. We give a non-parameterized algorithm solving the decision versions of both *Curve Cover* and *Hyperplane Cover* in $\mathcal{O}^*(2^n)$ time and polynomial space. Furthermore we present parameterized algorithms for *Curve Cover* and *Plane Cover* (*Hyperplane Cover* in $\mathbb{R}^3$). These solve *Curve Cover* in time $\mathcal{O}^*\big((Ck/\log k)^{(d-1)k}\big)$ and *Plane Cover* in time $\mathcal{O}^*\Big((Ck^2/\log^{1/5} k)^k\Big)$, both using polynomial space. The main idea is to use Szemerédi-Trotter-type incidence bounds and using the aforementioned $\mathcal{O}^*(2^n)$ algorithm as a base case. We make heavy use of (specialized) incidence bounds and our running time is very sensitive to the maximum number of possible incidences between points and curves or hyperplanes. In general, utilization of incidence bounds for constructing algorithms is rare (see e.g. [13, 12]) and to our knowledge we are the first to do so for this type of covering problem. It is generally believed that point sets that create large number of incidences must have some "algebraic sub-structure" (see e.g. [11]) but curiously, the situation is not fully understood even in two dimensions. So, it might be possible to get better specialized incidence bounds for us in the context of covering points. Thus, we hope that this work can give further motivation to study specialized incidence bounds. This is a shortened conference version of the paper. Omitted proofs and pseudocode can be found in the full version [1].

2 Preliminaries

2.1 Definitions

We begin by briefly explaining the concept of fixed-parameter tractability before formally stating the *Curve Cover* and *Hyperplane Cover* problems.

▶ **Definition 1.** A problem is said to be *fixed-parameter tractable* if there is a parameter k to an instance I, such that I can be decided by an algorithm in time $\mathcal{O}(f(k)poly(|I|))$ for some computable function f.

The function f is allowed to be any computable function, but for NP-complete problems can be expected to be at least single exponential. The name refers to the fact that these algorithms run in polynomial time when k is (bounded by) a constant. Within this paper I will typically be a set of points and k is always a solution budget: the maximum allowed size of any solution of covering objects, but not necessarily the size of the optimal such solution.

Let P be a set of n points in any dimension, and d, s be non-negative integers.

▶ **Definition 2.** A set of algebraic curves $\mathcal{C}$ are called (d, s)-curves if (i) any pair of curves from $\mathcal{C}$ intersect in at most s points and (ii) for any d points there are at most s curves in $\mathcal{C}$ through them. The parameter d is the *degrees of freedom* and s is the *multiplicity-type*.

The set $\mathcal{C}$ could be an infinite set corresponding to a family of curves, and it is often defined implicitly. We assume two geometric predicates: First, we assume that given two curves $c_1, c_2 \in \mathcal{C}$, we can find their intersecting points in polynomial time. Second, we assume that given any set of up to $s + 1$ points, in polynomial time, we can find a curve that passes through the points or decide that no such curve exists. These two predicates are satisfied in the real RAM model of computation for many families of algebraic curves and can be approximated reasonably well in practice.

We say that a curve *covers* a point, or that a point is *covered* by a curve, if the point lies on the curve. A set of curves $H \subset \mathcal{C}$ *covers* a set of points P if every point in P is covered by a curve in H, furthermore, H is a k-cover if $|H| \leq k$.

▶ **Definition 3** (Curve Cover Problem)**.** Given a family of (d, s)-curves $\mathcal{C}$, a set of points P, and an integer k, does there exist a subset of $\mathcal{C}$ that is a k-cover of P?

Now let P be a set of points in $\mathbb{R}^d$. A hyperplane covers a point if the point lies on the hyperplane. A set H of hyperplanes covers a set of points if every point is covered by some hyperplane; H is a k-cover if $|H| \leq k$. In $\mathbb{R}^d$, a j-flat is a j-dimensional affine subset of the space, e.g., 0-flats are points, 1-flats are lines and $(d-1)$-flats are called hyperplanes.

▶ **Definition 4** (Hyperplane Cover Problem)**.** Given an integer k and a set P of points in $\mathbb{R}^d$, does there exist a set of hyperplanes that is a k-cover of P?

For $d = 3$ we call the problem *Plane Cover*. To make our parameterized *Plane Cover* algorithm work, we need to introduce a third generalization: a version of *Hyperplane Cover* where the input contains any type of flats. A hyperplane covers a j-flat for $j \leq d-2$ if the flat lies on the hyperplane; further notation follows naturally from the above.

▶ **Definition 5** (Any-flat Hyperplane Cover Problem)**.** For $k \in \mathbb{N}$ and a tuple $P = \langle P_0, \ldots, P_{d-2} \rangle$, where P_i is a set of i-flats in $\mathbb{R}^d$, does there exist a set of hyperplanes that is a k-cover of P?

We stress that our non-parameterized algorithm in Section 3 solves *Any-flat Hyperplane Cover* while the parameterized algorithm in Section 5 solves *Plane Cover*. *Line Cover* is a special case of both *Curve Cover* and *Hyperplane Cover*. Since *Line Cover* is known to be both NP-hard [17] and APX-hard [15], the same applies to its three generalizations as well.

2.2 Kernels

Central to parameterized complexity theory is the concept of *polynomial kernels*. A parameterized problem has a polynomial kernel if an instance $\langle P, k \rangle$ in polynomial time can

be reduced to an instance $\langle P', k' \rangle$ where $|P'|$ and k' are bounded by $k^{\mathcal{O}(1)}$ and $\langle P, k \rangle$ is a yes-instance if and only if $\langle P', k' \rangle$ is a yes-instance. Problems with polynomial kernels are immediately fixed-parameter tractable: simply use brute force on reduced instance.

▶ **Lemma 6.** *For a family $\mathcal{C}$ of (d, s)-curves, Curve Cover has a size sk^2 kernel where no curve in $\mathcal{C}$ covers more than sk points.*

Proof. See the full version of the paper. ◀

For *Any-flat Hyperplane Cover* a size k^d kernel is presented in [16]. It uses a *grouping* operation, removing points and replacing them with higher dimension flats, which is not acceptable for a *Hyperplane Cover* input. We present an alternative, slightly weaker hyperplane kernel containing only points; in $\mathbb{R}^3$ it contains at most $k^3 + k^2$ points.

▶ **Lemma 7.** *Hyperplane Cover in $\mathbb{R}^d, d \geq 2$, has a size $k^2(\sum_{i=0}^{d-2} k^i) = \mathcal{O}\big(k^d\big)$ kernel where for $j \leq d-2$ any j-flat covers at most $\sum_{i=0}^{j} k^i = \mathcal{O}\big(k^j\big)$ points and any hyperplane covers at most $k \sum_{i=0}^{d-2} = \mathcal{O}\big(k^{d-2}\big)$ points.*

Proof. See the full version of the paper. ◀

Our algorithms will use both properties of the kernels. Kratsch et al. [14] showed that these kernels are essentially tight under standard assumptions in computational complexity.

▶ **Theorem 8** (Kratsch et al. [14])**.** *Line Cover has no kernel of size $\mathcal{O}\big(k^{2-\varepsilon}\big)$ unless coNP $\subseteq$ NP/poly.*

2.3 Incidence bounds

Consider the *Line Cover* problem. Obviously, if the input of n points are in general position, then we need $n/2$ lines to cover them. Thus, if $k \ll \frac{n}{2}$, we expect the points to contain "some structure" if they are to be covered by k lines. Such "structures" are very relevant to the study of incidences. For a set P of points and a set L of lines, the classical Szemerédi-Trotter [20] theorem gives an upper bound on the number of point-line incidences, $I(L, P)$, in $\mathbb{R}^2$.

▶ **Theorem 9** (Szemerédi and Trotter [20])**.** *For a set P of n points and a set L of m lines in the plane, let $I(L, P) = |\{(p, \ell) \mid p \in P \cap \ell, \ell \in L\}|$. Then $I(L, P) = \mathcal{O}\big((nm)^{2/3} + n + m\big)$.*

The linear terms in the theorem arise from the cases when there are very few lines compared to points (or vice versa). In the setting of *Line Cover* these cases are not interesting since they are easy to solve. The remaining term is therefore the interesting one. Since it it large, it implies there are many ways of placing a line such that it covers many points; this demonstrates the importance of incidence bounds for covering problems. We introduce specific incidence bounds for curves and hyperplanes in their relevant sections.

3 Inclusion-exclusion algorithm

This section outlines an algorithm Inclusion-Exclusion that for both problems decides the size of the minimum cover, or the existence of a k-cover, of a point set P in $\mathcal{O}^*(2^n)$ time and polynomial space. Our algorithm improves over the one from [4] for *Line Cover* which finds the cardinality of the smallest cover of P with the same time bound but exponential space. The technique is an adaptation of the one presented in [3]; their paper immediately gives either $\mathcal{O}^*(3^n)$-time polynomial-space or $\mathcal{O}^*(2^n)$-time $\mathcal{O}^*(2^n)$-space algorithms for our problems. We give full details of the technique for completeness; to do so, we require the intersection version of the inclusion-exclusion principle.

▶ **Theorem 10** (Folklore). *Let $A_1, \ldots, A_n$ be a number of subsets of a universe $\mathcal{U}$. Using the notation that $\overline{A} = \mathcal{U} \setminus A$ and $\bigcap_{i\in\emptyset} \overline{A_i} = \mathcal{U}$, we have:*

$$\left| \bigcap_{i\in\{1,\ldots,n\}} A_i \right| = \sum_{X\subseteq\{1,\ldots,n\}} (-1)^{|X|} \left| \bigcap_{i\in X} \overline{A_i} \right|.$$

3.1 Curve Cover

Let P be the input set of points and $\mathcal{C}$ be the family of (d, s)-curves under consideration. Although we are creating a non-parameterized algorithm, we nevertheless assume that we have access to the solution parameter k. This assumption will be removed later. We say a set Q is a *coverable set in* P (or is *coverable* in P) if $Q \subseteq P$ and Q has a 1-cover.

Let a *tuple* (in P) be a k-tuple $\langle Q_1, \ldots, Q_k\rangle$ such that $\forall i : Q_i$ is coverable in P. Note that there is no restriction on pairwise intersection between two coverable sets in a tuple. Define $\mathcal{U}$ as the set of all tuples. For $p \in P$, let $A_p = \{\langle Q_1, \ldots, Q_k\rangle \mid p \in \bigcup_i Q_i\} \subseteq \mathcal{U}$ be the set of all tuples where at least one coverable set contains p.

▶ **Lemma 11.** *P has a k-cover if and only if $\left|\bigcap_{p\in P} A_p\right| \geq 1$.*

Proof. Take a tuple in $\bigcap_{p\in P} A_p$. For each coverable set Q in the tuple, place a curve that covers Q. Since the tuple was in the intersection, every point is in some coverable set, so every point is covered by a placed curve. Hence we have a k-cover.

Take a k-cover $\mathcal{C}$ and from each curve $c \in \mathcal{C}$ construct a coverable set of the points covered by c. Form a tuple out of these sets and observe that the tuple is in the intersection $\bigcap_{p\in P} A_p$, hence its cardinality is at least 1. ◀

Note that several tuples may correspond to the same k-cover, so this technique cannot be used for the counting version of the problem. Theorem 10 and Lemma 11 reduce the problem of deciding the existence of k-covers to computing a quantity $\left|\bigcap_{i\in X} \overline{A_i}\right|$. The key observation is that $\overline{A_p}$ is the set of tuples where no coverable set contains p and $\bigcap_{i\in X} \overline{A_i}$ is the set of tuples that contain no point in X, i.e. the set of tuples in $P \setminus X$. The remainder of this section shows how to compute the size of this set in polynomial time. Let $c(X) = |\{Q \mid Q \subseteq X, Q \text{ is coverable in } X\}|$ be the number of coverable sets in a point set X. A tuple in $P \setminus X$ is k coverable sets drawn from a size $c(P \setminus X)$ pool (with replacement), hence there are $c(P \setminus X)^k$ such tuples. To compute $c(X)$ we introduce the notion of *representatives*. Let π be an arbitrary ordering of P. The representative $R = \{r_1, \ldots, r_i\}$ of a coverable set Q is the $\min(|Q|, s + 1)$ first points in Q as determined by the order π. Note that for any coverable set Q, it holds that $R \subseteq Q$. Let $q(X, \pi, R)$ be the number of coverable sets that have the representative R.

▶ **Lemma 12.** *$q(X, \pi, R)$ can be computed in $\mathcal{O}(|X|)$ time and $\mathcal{O}(\log |X|)$ space.*

Proof. If R is not a valid representative, $q(X, \pi, R) = 0$. If $|R| \leq s$, $q(X, \pi, R) = 1$. If $|R| = s + 1$, let U be the union of every coverable set with representative R, and $X' = U \setminus R$. The number of subsets of X' is the number of coverable sets with representative R, i.e. $q(X, \pi, R) = 2^{|X'|}$. For any $p \in P$ with $\pi(p) > \pi(r_i)$, $p \in X'$ if and only if there is a curve $c \in \mathcal{C}$ such that c covers $\{r_1, \ldots, r_i, p\}$. Since $i \leq s + 1$ the time complexity is $\mathcal{O}(|X|)$. The space complexity is logarithmic since we need only maintain $|X'|$ rather than X'. ◀

▶ **Lemma 13.** *$c(X)$ can be computed in $\mathcal{O}\big(|X|^{s+2}\big)$ time and $\mathcal{O}(|X|)$ space.*

Proof. Fix an ordering π. As every coverable set in X has exactly one representative under π, we get that $c(X) = \sum_R q(X, \pi, R)$. There there are only $\mathcal{O}\Big(\binom{|X|}{s+1}\Big) = \mathcal{O}\big(|X|^{s+1}\big)$ choices of R for which $q(X, \pi, R) > 0$, and by Lemma 12 each term of the sum is computable in $\mathcal{O}(|X|)$ time and logarithmic space. The space complexity is therefore dominated by the space to store π which is linear. ◀

▶ **Theorem 14.** *There exists a k-cover of curves from $\mathcal{C}$ for P if and only if*

$$\left|\bigcap_{p\in P} A_p\right| = \sum_{X\subseteq P} (-1)^{|X|} \left|\bigcap_{p\in X} \overline{A_p}\right| = \sum_{X\subseteq P} (-1)^{|X|} c(P \setminus X)^k \geq 1.$$

This comparison can be performed in $\mathcal{O}\big(2^n n^{s+2}\big)$ time and $\mathcal{O}(nk)$ bits of space.

Proof. Since $c(P \setminus X)^k \leq 2^{nk}$ for any X it can be stored in nk bits. The absolute value of the partial sum can be kept smaller than 2^{nk} by choosing an appropriate next X. The rest follows from Theorem 10, Lemma 11 and Lemma 13. ◀

Finally, we remove the assumption that we have the parameter k. Any input requires at most n curves. Since k is only used to compute $c(X)^k$ we can try $k = 1, 2, \ldots, n$ and return the first k with a positive sum. This increases the time by an $\mathcal{O}(n)$ factor. Alternatively, we can run n simultaneous sums, since the parameter k is only accessed when computing $c(X)^k$. This increases the space by factor $\mathcal{O}(n)$ and the time by a lower-order additive term.

3.2 Any-flat Hyperplane Cover

Here we treat all flats in the instance $\langle P_0, \ldots, P_{d-2}\rangle$ as atomic objects and P as a union $\bigcup_{i=0}^{d-2} P_i$. This algorithm is very similar to that of Section 3.1, so we only describe their differences. A set of flats $Q \subseteq P$ is a coverable set in P if there exists a hyperplane that covers every $p \in Q$. The representative of $\emptyset$ is $\emptyset$, and the representative of a non-empty coverable set Q is a set $R = \{r_1, \ldots, r_i\}$. Let r_1 be the first flat in Q and for $j \geq 2$, r_j is defined if the affine hull of $\{r_1, \ldots, r_{j-1}\}$ has lower dimension than the affine hull of Q. If so, let r_j be the first flat in Q that is not covered by the affine hull of $\{r_1, \ldots, r_{j-1}\}$.

▶ **Lemma 15.** *$q(X, \pi, R)$ can be computed in $\mathcal{O}(|X|)$ time and $\mathcal{O}(\log |X|)$ space.*

Proof. If R is not a valid representative, $q(X, \pi, R) = 0$. Otherwise, let U be the union all coverable sets with the representative R, and $X' = U \setminus S$. For every $p \in X \setminus R$, let j be the highest index such that $\pi(r_j) < \pi(p)$. Then $p \in X'$ if and only if p is on the affine hull of $\{r_1, \ldots, r_j\}$. ◀

There are $\mathcal{O}\big(\binom{n}{d}\big)$ representatives R with $q(X, \pi, R) > 0$ so the following two results hold; their proofs are analogous to Lemma 13 and Theorem 14.

▶ **Lemma 16.** *$c(X)$ may be computed in $\mathcal{O}\big(|X|^{d+1}\big)$ time and $\mathcal{O}(|X|)$ space.*

▶ **Theorem 17.** *There exists a hyperplane k-cover for P if and only if*

$$\left|\bigcap_{p\in P} A_p\right| = \sum_{X\subseteq P} (-1)^{|X|} \left|\bigcap_{p\in X} \overline{A_p}\right| = \sum_{X\subseteq P} (-1)^{|X|} c(P \setminus X)^k \geq 1.$$

This comparison may be performed in $\mathcal{O}\big(2^n n^{d+1}\big)$ time and $\mathcal{O}(nk)$ bits of space.

4 Curve Cover

Recall that we are considering (d, s)-curves, where d and s are constants. Since we have a kernel of up to sk^2 points, Inclusion-Exclusion used on its own runs in time $\mathcal{O}^*\left(2^{sk^2}\right)$ which is too slow to give an improvement. We improve this by first using a technique that reduces the number of points in the input, and then using Inclusion-Exclusion. To describe this technique and the intuition behind it, we first provide a framework based on the following theorem by Pach and Sharir.

▶ **Theorem 18** (Pach and Sharir [18]). *Let P be a set of n points and L a set of m (d, s)-curves in the plane. The number of point-curve incidences between P and L is*

$$I(P, L) = \mathcal{O}\left(n^{d/(2d-1)} m^{(2d-2)/(2d-1)} + n + m\right).$$

Note that the above holds for curves in arbitrary dimension. This can be seen by projecting the points and curves onto a random plane, which will keep the projection of distinct points, and prevent the curves from projecting to overlapping curves.

▶ **Definition 19.** Let a *candidate* be any curve in $\mathcal{C}$ that covers at least 1 point in P. Define its *richness* with respect to P as the number of points it covers. A candidate is γ-rich if its richness is at least γ, and γ-poor if its richness is at most γ.

Recall that from the kernel in Lemma 6, every candidate is sk-poor. The following gives a bound on the number of γ-rich candidates and is an immediate consequence of Theorem 18.

▶ **Lemma 20** (Folklore). *Let P be a set of n points in some finite dimension space $\mathbb{R}^x$. The number of γ-rich candidates in P is $\mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right)$.*

Intuition for algorithm. We exploit the following observation: given a k-cover $\mathcal{C}$, some curves in $\mathcal{C}$ might be significantly richer than others. The main idea of our technique is to try to select (i.e. branch on) these rich curves first. Since they cover "many" points, removing these decreases the ratio $|P|/k$ and calling Inclusion-Exclusion eventually becomes viable. The idea to branch on rich curves first has another important consequence. Suppose we know that no candidate in $\mathcal{C}$ covers more than γ points in P. This immediately implies that if there are strictly more than $k\gamma$ points in P, it is impossible to cover P. Therefore we have $|P|/k \leq \gamma$. Now look at the set of $\frac{\gamma}{2}$-rich candidates and decide for each whether to include it in the cover or not. By the earlier observation, including such a candidate is good for reducing the ratio $|P|/k$. But excluding such a candidate has essentially the same effect, because that candidate will not be considered again (remove it from $\mathcal{C}$). Any remaining candidates in $\mathcal{C}$ now cover at most $\frac{\gamma}{2}$ points; we must have $|P|/k \leq \frac{\gamma}{2}$ (or the instance is not solvable) and have strengthened the bound on the ratio. Regardless of which choice we make, we make progress towards being able to call the base case.

This strategy also makes sense from a combinatorial point of view, because from Lemma 20 it follows that the search space is small for rich curves. Switching to Inclusion-Exclusion early enough lets us bypass the potentially very large search space of poor candidates.

The Algorithm. Let r be a parameter. The exact value is set in the proof of Theorem 25, for now it is enough that $r = \Theta(\log k)$. For a budget k let $\langle k_1, \ldots, k_r \rangle$ with $\sum_j k_j = k$ be a *budget partition*. We describe a main recursive algorithm CC-Recursive (see full paper for pseudocode) that takes 4 arguments: the point set P, the class of curves $\mathcal{C}$, a budget partition

$\langle k_1, \ldots, k_r \rangle$, and a recursion level i. For convenience we define $\gamma_i = sk/2^i$. A simple top-level procedure CurveCover tries all budget partitions and calls the recursive algorithm with that partition at recursion level 1.

At every recursion depth i, let $K_i = \sum_{j=i}^{r} k_j$ be the remaining budget and P_i the remaining point set. That means earlier levels have created a partial solution $\mathcal{C}_{i-1}$ of $k - K_i$ curves covering the points $P \setminus P_i$. The recursive algorithm will try to cover the remaining points using γ_{i-1}-poor curves. Specifically, at depth i let S be the set of candidates from $\mathcal{C}$ that are γ_i-rich and γ_{i-1}-poor. Since from depth i and onward it has a remaining budget of K_i and cannot pick candidates that are $(\gamma_{i-1} + 1)$-rich, the algorithm rejects if strictly more than $K_i \gamma_{i-1}$ remain. If fewer than $\frac{(d-1)}{2} K_i \log k$ points remain, the sub problem is solved with inclusion-exclusion.

If neither a reject (due to too many points) or a base-case call to inclusion-exclusion has occurred, the algorithm will branch. It does so in $\binom{|S|}{k_i}$ ways by simply trying all ways of choosing k_i candidates from S. For each such choice, all points in P covered by the chosen candidates are removed and the algorithm recurses to depth $i + 1$. If all those branches fail, the instance is rejected.

4.1 Analysis

▶ **Lemma 21.** *Algorithm* CurveCover *decides whether P has a k-cover of curves from $\mathcal{C}$.*

Proof. Regard CurveCover as being non-deterministic. Suppose P has a k-cover $\mathcal{C}$. The proof is by induction on the recursion. Assume as the induction hypothesis that the current partial solution $\mathcal{C}_{i-1}$ is a subset of $\mathcal{C}$ and that $\mathcal{C}$ contains no curves that are $(\gamma_{i-1} + 1)$-rich when restricted to P_i. The assumption is trivially true for $i = 1$ as $\mathcal{C}_0 = \emptyset$.

By the induction hypothesis, $\mathcal{C} \setminus \mathcal{C}_{i-1}$ is a K_i-cover for P_i using only γ_{i-1}-poor curves. Therefore $|P_i| \leq \gamma_{i-1} K_i$ and the algorithm does not reject incorrectly. Furthermore, if Inclusion-Exclusion is called it accepts since we are in the case that a solution exists.

Otherwise, let $D \subseteq \mathcal{C} \setminus \mathcal{C}_{i-1}$ be the curves that are γ_i-rich when restricted to P_i. The algorithm non-deterministically picks D from the set of candidates S and constructs $\mathcal{C}_i = \mathcal{C}_{i-1} \cup D$. This leaves $\mathcal{C}_i$ to be a subset of $\mathcal{C}$. Additionally, $\mathcal{C}_i$ contains all γ_i-rich curves in $\mathcal{C}$ restricted to P_i and hence to $P_{i+1} \subseteq P_i$, upholding the induction hypothesis.

Suppose the algorithm accepts the instance $\langle P, k \rangle$. It can only accept if some call to Inclusion-Exclusion accepts. Let $\mathcal{C}_r$ be the set of curves selected by the recursive part such that Inclusion-Exclusion accepted the instance $\langle P \setminus \mathcal{C}_r, k - |\mathcal{C}_r| \rangle$. Let $\mathcal{C}_{ie}$ be any $(k - |\mathcal{C}_r|)$-cover of $P \setminus \mathcal{C}_r$. Then $\mathcal{C}_r \cup \mathcal{C}_{ie}$ is a k-cover of P. ◀

By the nature of the inclusion-exclusion algorithm, CurveCover detects the existence of a k-cover rather than producing one. But since CC-Recursive produces a partial cover during its execution, it is straight-forward to extend that into a full k-cover by using Inclusion-Exclusion as an oracle.

Running time. To analyze the running time of the algorithm we see the execution of CC-Recursive as a search tree $\mathcal{T}$. Each leaf of the tree is either an immediate reject or a call to Inclusion-Exclusion. Since the latter is obviously most costly to run, we must assume for a worst case analysis that every leaf node calls the base case algorithm. The running time is the number of leaf nodes in the search tree times the running time of Inclusion-Exclusion. Since the algorithm performs exponential work in these leaf nodes but not in inner nodes, it is insufficient to reason about the *size* of the tree. Therefore we will speak of the "running time of a subtree", which simply means the running time of the

recursive call that corresponds to the root of that subtree. We show that in the worst case, $\mathcal{T}$ is a complete tree $\mathcal{T}_1$ of depth r. That is, $\mathcal{T}_1$ has no leaf nodes at depths less than r.

Let $\mathcal{T}_j$ be a complete subtree of $\mathcal{T}_1$ rooted at depth j. To prove that $\mathcal{T}_1$ is the worst case for $\mathcal{T}$ we prove two things. First we first prove an upper bound on the running time for arbitrary $\mathcal{T}_j$. Then we prove that the running time of $\mathcal{T}_1$ can only improve if an arbitrary subtree is replaced by a leaf (i.e. a call to Inclusion-Exclusion). The most involved part is proving an upper bound on the number of leaves of $\mathcal{T}_j$.

▶ **Lemma 22.** *Let L be the number of leaves in $\mathcal{T}_j$. Then for some constant $c_2 = c_2(d, s)$, L is bounded by*

$$L \leq \left(\frac{c_2 k^d}{(k - k_r) \log^{d-1} k} \right)^{K_j - k_r}.$$

The proof is long and tedious and left for the appendix of the full version of the paper. To give an idea of how Lemma 22 is proved, we sketch a simplified worst case analysis for *Line Cover*. The analysis can be generalized to *Curve Cover* and gives (up to a constant in the base of the exponent) the same running time as the real worst case.

Analysis sketch. The branching of $\mathcal{T}_1$ at recursion level i depends on the budget k_i that is being used. That means that the structure of the whole tree depends on the complete budget partition. From Lemma 20 it follows that the lower the richness the more candidates there are. Since the richness halves after every recursive call, one could conjecture that the worst case budget partition would put as much budget in the end. It could e.g. look like $\langle 0, 0, \ldots, 0, k_{r-1}, k_r \rangle$, where $k - k_r = k_{r-1} > k_r$. That is, only in the penultimate and last recursion level is there any budget to spend. At the deepest level of recursion, the richness considered is strictly less than $\frac{\log k}{2}$ (because with this richness the base case algorithm is efficient). Therefore, at the penultimate recursion level the richness is $\log k$. At this level there are $k \log k$ points left and we can apply Lemma 20 to bound the number of $\log k$ rich lines. This yields a bound of $\frac{k^2}{\log k}$ on the number of candidates. From these we pick $k - k_r$ lines, giving a branching of roughly $\left(\frac{k^2}{(k-k_r)\log k}\right)^{k-k_r}$ (where roughly means up to a constant in the base of the exponent).

It turns out that the worst case budget partition is in fact $\langle k_0 2^1, k_0 2^2, ..., k_0 2^{r-1}, k_r \rangle$ for some k_0. However, to understand where the division by $\log^{d-1} k$ comes from in the expression of Lemma 22, it is sufficient to understand the above analysis sketch. With Lemma 22 in place, we can prove the following bound on the running time of $\mathcal{T}_j$.

▶ **Lemma 23.** *The time complexity of a complete subtree $\mathcal{T}_j$ is $\mathcal{O}^*\big((c_4 k/\log k)^{(d-1)K_j}\big)$, where $c_4 = c_4(d, s)$ is a constant that depends on the family $\mathcal{C}$.*

Proof. By Lemma 22, the number of leaves in $\mathcal{T}_j$ is $L \leq \left(\frac{c_2 k^d}{(k-k_r)\log^{d-1} k}\right)^{K_j - k_r}$. Observe that at depth r, Inclusion-Exclusion runs in time $\mathcal{O}^*\left(2^{\frac{d-1}{2} k_r \log k}\right) = \mathcal{O}^*\left(k^{\frac{d-1}{2} k_r}\right)$. Since an inner node performs polynomial time work and the leaves perform exponential time work, this immediately implies that the running time for $\mathcal{T}_j$ is

$$\mathcal{O}^*\left(\left(\frac{c_2 k^d}{(k - k_r) \log^{d-1} k} \right)^{K_j - k_r} \cdot k^{\frac{d-1}{2} k_r} \right).$$

Suppose $k - k_r = o(k)$. Then it holds that $K_j - k_r = o(K_h)$ since $k \geq K_j \geq k_r$. We get

$$\mathcal{O}^*\left(\left(\frac{c_2 k^d}{(k-k_r)\log^{d-1} k}\right)^{o(K_j)} \cdot k^{\frac{d-1}{2}(K_j - o(K_j))}\right) = \mathcal{O}^*\left(2^{o(dK_j \log k) + \frac{d-1}{2}(K_j \log k - o(k \log k))}\right).$$

With some simple algebra one gets that the exponent is bounded by $(d-1)K_j(\log k - \log\log k)$, giving the desired time bound $\mathcal{O}^*\left(2^{(d-1)K_j(\log k - \log\log k)}\right) = \mathcal{O}^*\left((k/\log k)^{(d-1)K_j}\right)$.

If $k - k_r \neq o(k)$, then $k - k_r \geq c_3 k$ for some constant $c_3 > 0$. The running time solves to:

$$\mathcal{O}^*\left(\left(\frac{c_2 k^{d-1}}{c_3 \log^{d-1} k}\right)^{K_j - k_r} \cdot k^{\frac{d-1}{2}k_r}\right) = \mathcal{O}^*\left(\left(\frac{c_4 k}{\log k}\right)^{(d-1)K_j}\right)$$

where $c_4 = (c_2/c_3)^{1/(d-1)}$. ◀

▶ **Lemma 24.** *Let L_j be a depth $j < r$ leaf of $\mathcal{T}$ that calls* Inclusion-Exclusion*. Then the running time of $\mathcal{T}_j$ dominates that of L_j.*

Proof. By Lemma 23, the time complexity of $\mathcal{T}_j$ is $\mathcal{O}^*\left((c_4 k/\log k)^{(d-1)K_j}\right)$. At depth j the algorithm has K_j remaining budget to spend. Since the algorithm called Inclusion-Exclusion at this depth, at most $\frac{d-1}{2}K_j \log k$ points remained and the call takes $\mathcal{O}^*\left(2^{\frac{d-1}{2}K_j \log k}\right) = \mathcal{O}^*\left(k^{\frac{d-1}{2}K_j}\right)$ time, which is bounded by that for $\mathcal{T}_j$. ◀

▶ **Theorem 25.** CurveCover *decides Curve Cover in time $\mathcal{O}^*\left((Ck/\log k)^{(d-1)k}\right)$ where $C = C(d,s)$ is a constant that depends on the family $\mathcal{C}$.*

Proof. Fix a budget partition $\langle k_1, \ldots, k_r \rangle$. By Lemma 24, calling Inclusion-Exclusion at a depth $j < r$ does not increase the running time of the algorithm. Therefore the time complexity of CC-Recursive is $\mathcal{O}^*\left((c_4 k/\log k)^{(d-1)K_1}\right) = (c_4 k/\log k)^{(d-1)k}$.

CurveCover runs CC-Recursive over all possible budget partitions, of which by the "stars and bars" theorem are only $\binom{k+r-1}{k}$, a quasi-polynomial in k. Therefore by letting $C = c_4 + \varepsilon$ for any $\varepsilon > 0$, the time complexity of CurveCover is $\mathcal{O}^*\left((Ck/\log k)^{(d-1)k}\right)$. ◀

▶ **Lemma 26.** *The polynomial time dependency of* CurveCover *is $\mathcal{O}\left((k \log k)^{2+s}\right)$ and its space complexity is $\mathcal{O}\left(k^4 \log^2 k\right)$ bits.*

Proof. See the full version of the paper. ◀

5 Hyperplane Cover

One generalization of *Line Cover* was discussed in the previous section. In this section we discuss its other generalization *Hyperplane Cover*, and give an algorithm for the three dimensional case. We would like to follow the same basic attack plan of using incidence bounds but here we face significant challenges and we need non-trivial changes in our approach. One major challenge is the nature of incidences in higher dimensions. For example, the asymptotically maximum number of incidences between a set of points and hyperplanes in d-dimensions is obtained by placing half of the points on one two-dimensional plane (see [2, 5]) which clearly makes it an easy instance for our algorithm (due to kernelization). Thus, in essence, we need to use specialized incidence bounds that disallow such configurations of points; unfortunately, such bounds are more difficult to prove than ordinary incidence bounds (and as it turns out, also more difficult to use).

5.1 Point-Hyperplane incidence bounds in higher dimensions

The most general bound for point-hyperplane incidences from [2, 6] yields a bound of $\Theta\left(\frac{n^d}{\gamma^3} + \frac{n^{d-1}}{\gamma}\right)$ on the number of γ-rich hyperplanes in d dimensions similar to Lemma 20 (where the left term is again the significant one). Our method requires that the exponent is greater in the denominator than in the numerator, so this bound is not usable beyond $\mathbb{R}^2$. As stated before, the constructions that make the upper bound tight are easy cases for our algorithm; they contain very low dimensional flats that have many points on them. A specialized bound appears in [7], where the authors study the number of incidences between points and hyperplanes with a certain *saturation*.

▶ **Definition 27.** Consider a point set P and a hyperplane h in $\mathbb{R}^d$. We say that h is σ-saturated, $\sigma > 0$, if $H \cap P$ spans at least $\sigma \cdot |H \cap P|^{d-1}$ distinct $(d-2)$-flats of F.

For example in three dimensions, a $(1 - \frac{1}{n})$-saturated plane contains no three collinear points. The main theorem of [7] can be stated as follows.

▶ **Theorem 28** (Elekes and Tóth [7]). *Let $d \geq 2$ be the dimension and $\sigma > 0$ a real number. There is a constant $C_1(d, \sigma)$ with the following property. For every set P of n points in $\mathbb{R}^d$, the number of γ-rich σ-saturated hyperplanes is at most:*

$$\mathcal{O}\left(C_1(d,\sigma)\left(\frac{n^d}{\gamma^{d+1}} + \frac{n^{d-1}}{\gamma^{d-1}}\right)\right).$$

The interesting term in this bound has a greater exponent in the denominator, as required. Unfortunatley it is difficult to verify if a hyperplane is σ-saturated. In the same paper, the authors give another bound based on a more manageable property called *degeneracy*.

▶ **Definition 29.** Given a point set P and a hyperplane h in $\mathbb{R}^d$, we say that h is δ-degenerate, $0 < \delta \leq 1$, if $H \cap P$ is non-empty and at most $\delta \cdot |H \cap P|$ points of $H \cap P$ lie in any $(d-2)$-flat.

For example in $\mathbb{R}^3$, any 1-degenerate plane might have all its points lying on a single line, and a plane with degeneracy strictly less than 1 must have at least 3 points not on the same line. As such it is an easy property to test.

▶ **Theorem 30** (Elekes and Tóth [7]). *For any set of n points in $\mathbb{R}^3$, the number of γ-rich δ-degenerate planes is at most*

$$\mathcal{O}\left(\frac{1}{(1-\delta)^4}\left(\frac{n^3}{\gamma^4} + \frac{n^2}{\gamma^2}\right)\right).$$

This bound is usable and relies on an easily-tested property, but unfortunately only applies to the $\mathbb{R}^3$ setting.

5.2 Algorithm for *Plane Cover*

In this section we present our algorithm PC-Recursive that solves *Plane Cover* using the bound from Theorem 30. This algorithm is similar to that for *Curve Cover*, and it is assumed that the reader is sufficiently familiar with CC-Recursive before reading this section.

Recall that by Lemma 7, *Plane Cover* has a kernel of size $k^3 + k^2$ where no plane contains more than $k(k+1) \leq 2k^2$ points and no two planes pairwise intersect in more than $k+1$ points. For convenience we define $\gamma_0 = k^2 + k$ and $\gamma_i = k^2/2^i$ for $i > 0$. We inherit the basic structure of the CC-Recursive algorithm, such that every recursion level considers γ_i-rich-γ_{i-1}-poor candidates. Additionally, any candidate considered must be *not-too-degenerate*:

▶ **Definition 31.** Let $\delta_i = 1 - \gamma_i^{-1/5}$. A γ_i-rich-γ_{i-1}-poor plane is called *not-too-degenerate* if it is δ_i-degenerate, and *too-degenerate* otherwise.

It is of no consequence that the definition does not cover all candidates considered on depth 1. The main extension of PC-Recursive compared to CC-Recursive is to first use a different technique to deal with too-degenerate candidates, which then allows normal branching on the not-too-degenerate ones. The key observation is that any too-degenerate candidate has at least $\gamma_i\delta_i = \gamma_i - \gamma_i^{4/5}$ points on a line and at most $\gamma_{i-1}(1-\delta_i) = 2\gamma_i^{4/5}$ points not on it.

Suppose a k-cover contains some too-degenerate plane h. By correctly guessing its very rich line ℓ and removing the points on the line, the algorithm makes decent progress in terms of shrinking the instance. The points on h but not ℓ will remain in the instance even though the budget for covering them has been paid. These are called the *ghost points* of h (or of ℓ), and ℓ is called a *degenerate line*. The ghost points must be removed by extending the line ℓ into a full plane. But the ghost points are few enough that the algorithm can delay this action until a later recursion level. Specifically, for a line ℓ guessed at depth i, we extend ℓ into a plane at the first recursion depth which considers $2\gamma_i^{4/5}$-poor candidates, i.e. the depth j such that $\gamma_{j-1} \geq 2\gamma_i^{4/5} \geq \gamma_j$.

Therefore the algorithm keeps a separate structure $\mathcal{L}$ of lines that have been guessed to be degenerate lines on some planes in the solution. Augment $\mathcal{L}$ to remember the recursion depth that a line was added to it. At any recursion depth, the algorithm will deal with old-enough lines in $\mathcal{L}$, then guess a new set of degenerate lines to add to $\mathcal{L}$ before finally branching on not-too-degenerate planes.

The algorithm Let $r = \Theta(\log k)$ as before. Let $\langle h_1, \ell_1, \ldots, h_r, \ell_r\rangle$ with $\sum_{i=1}^{r} h_i + \ell_i = k$ be a budget partition. The recursive algorithm PC-Recursive takes 4 arguments: the point set P, a set of lines $\mathcal{L}$, the budget partition, and a recursion level i. A top level algorithm PlaneCover tries all budget partitions and calls PC-Recursive accordingly.

Let the current recursion depth be i, and let $K_i = \sum_{j=i}^{r} h_j + \ell_j$ be the remaining budget. The sub-budget h_i will be spent on not-too-degenerate planes, and ℓ_i on degenerate lines. Let $\mathcal{L}$ be an augmented set of lines as described above. This means that earlier levels have already created a partial solution of $k - (K_i + |\mathcal{L}|)$ planes, and a set $\mathcal{L}$ of lines that still need to be covered by a plane. If strictly more than $(K_i + |\mathcal{L}|)\gamma_{i-1}$ points remain, the algorithm rejects. If at most $K_i \log k$ points remain, the algorithm switches to Inclusion-Exclusion passing on the instance $\langle P \cup \mathcal{L}, K_i + |\mathcal{L}|\rangle$.

Let $f = \left\lceil \frac{5(i-1)-2\log k}{4} \right\rceil$. Let A be the set of all lines in $\mathcal{L}$ that were added at depth f or earlier. Remove A from $\mathcal{L}$. For each way of placing $|A|$ planes $\mathcal{H}$ such that every plane contains one line in A and at least one point in P, let $P' = P \setminus (P \cap \mathcal{H})$ be the point set not covered by these planes. For a P', let H be the set of not-too-degenerate planes and L the set of degenerate lines too-degenerate candidates.

For every P' and every way of choosing h_i planes from H and ℓ_i lines from L, branch depth $i+1$ by removing the covered points from P and adding the chosen lines of L to $\mathcal{L}$.

5.3 Analysis

Correctness. To prove that the algorithm is correct, we follow a similar strategy as for CurveCover. We build on the notion that the algorithm is building up a partial solution of planes. Removing the points covered by the partial solution yields a "residual problem" just as in CurveCover. A partial solution is *correct* if it is a subset of some k-cover. Correctness

of the algorithm follows from proving that a k-cover exists if and only if one branch maintains a correct partial solution until it reaches Inclusion-Exclusion.

The difference here is that the residual problem is an instance of *Any-flat Plane Cover* and not *Plane Cover*. Therefore, we simply consider the original problem to be an instance of *Any-flat Plane Cover*, namely $R_1 = \langle P, \emptyset \rangle$. We say that $\mathcal{C}$ covers $\langle P, \mathcal{L} \rangle$ if $\mathcal{C}$ covers both P and $\mathcal{L}$. What needs to be established is that there is a correct way to replace points with lines (Observation 32) and, conversely, that there is a correct way to extend a line in $\mathcal{L}_i$ (Observation 33). The proofs for these are elementary and we omit them. Given these two facts, we can easily show that the algorithm will call Inclusion-Exclusion on appropriate instances.

▶ **Observation 32.** *Let ℓ be a line and $\mathcal{C}$ a set of planes such that some plane $h \in \mathcal{C}$ covers ℓ. Then $\mathcal{C}$ is a cover for $\langle P, \mathcal{L} \rangle$ if and only if $\mathcal{C}$ is a cover for $\langle P \setminus \ell, \mathcal{L} \cup \{\ell\} \rangle$.*

▶ **Observation 33.** *Let ℓ be a line, $\mathcal{L} \ni \ell$ be a set of lines, and $\mathcal{C}$ be a set of planes such that some $h \in \mathcal{C}$ covers ℓ but not any other line $\ell' \in \mathcal{L}$. Then $\mathcal{C}$ is a cover for $\langle P, \mathcal{L} \rangle$ if and only if $\mathcal{C} \setminus \{h\}$ is a cover of $\langle P \setminus h, \mathcal{L} \setminus \{\ell\} \rangle$.*

The conditions for Observation 33 might seem overly restrictive. But as the following lemma shows, that situation arises when $\mathcal{L}$ contains only correctly guessed degenerate lines.

▶ **Lemma 34.** *Let h be a too-degenerate plane with degenerate line ℓ such that $h \setminus \ell$ is a too-degenerate plane with degenerate line ℓ'. Then at no point during the execution of* PC-Recursive *will $\mathcal{L}$ contain ℓ and ℓ'.*

Proof sketch. The candidate $h \setminus \ell$ is too poor to be considered before the recursion depth where ℓ gets removed from L and extended to h. Full proof in full version of the paper. ◀

▶ **Lemma 35.** *If $\mathcal{L}$ contains only the degenerate lines of some too-degenerate planes in a k-cover, the number of ghost points at depth i is at most $|\mathcal{L}|\gamma_{i-1}$.*

Proof. See full version of the paper. ◀

▶ **Lemma 36.** *Algorithm* PlaneCover *decides whether P has a k-cover of planes.*

Proof. View the algorithm as being non-deterministic. Suppose P has a k-cover. Observation 32, Observation 33 and Lemma 34 guarantee that there is a correct path, and Lemma 35 guarantees that the point set is not erroneously rejected. Therefore the algorithm will send a yes-instance to Inclusion-Exclusion and accept.

Suppose P has no k-cover. If the conditions for Observation 33 are not satisfied, removing ℓ from $\mathcal{L}$ and pairing it up with points but not with another $\ell' \in \mathcal{L}$ can only reduce the number of solutions. Therefore the algorithm detects no cover and rejects. ◀

We can now state our main theorem for Plane-Cover.

▶ **Theorem 37.** PlaneCover *decides Plane Cover in $\mathcal{O}\Big((Ck^2/\log^{1/5} k)^k\Big)$ time for some constant C.*

Proof. See the full version of the paper. ◀

To give an idea of how to prove the above theorem, we give a sketch of the analysis that reflects the core of the real analysis. As before, we assume a (slightly incorrect) worst case for the budget partition where all the budget is assigned to the two deepest recursion levels. This gives a bound analogous to that in Lemma 22. After achieving this bound, the same arguments as for CurveCover can be applied to achieve the bound from Theorem 37.

Analysis sketch. The branching of the analysis is twofold. First there is the branching done on picking not-too-degenerate planes. Secondly, we have the branching on too-degenerate planes. This branching is actually a combination of picking the rich lines in too-degenerate planes, and the branching done by covering these lines with planes later on.

We sketch a proof for two extreme cases: either (**i**) $\forall i, k_i = h_i$ or (**ii**) $\forall i, \ell_i = k_i$. For both cases the branching can be bounded by $\left(\frac{k^3}{(k-k_r)\log^{1/5} k}\right)^{k-k_r}$ (compare to Lemma 22). The full proof for Theorem 37 shows that if the budget is distributed between these cases, then taking the product of the worst case running times of both cases is roughly the same as what we present here. As it turns out, the first case is (up to the incidence theorem used) identical to the curve case.

For case (**ii**) we again assume a (slightly incorrect) worst case budget partition where $k_{r-1} + k_r = k$ and $k_{r-1} \geq k_r$. By the same arguments as in the analysis sketch of Section 4 we have the following two parameters at recursion level $r-1$: the number of of points remaining is $n = k \log k$ and the richness γ_{r-1} is $\log k$. The algorithm picks γ_{i+1}-rich lines at level i, and these lines are matched with points at later level j where $\gamma_j = \gamma_i^{4/5}$. The cost for branching at level j is charged to level i, so that we can more easily analyze the total branching on lines selected at level i. With the budget partition as stated above, we can now bound the branching done at level $r-1$. By the Szemerédi-Trotter theorem, there are at most $\frac{n^2}{(\log k)^3} = \frac{k^2}{\log k}$ candidates, from which we select $k - k_r$ lines. This yields a total branching of $\binom{\frac{k^2}{\log k}}{k-k_r}$, which is roughly $\left(\frac{k^2}{(k-k_r)\log k}\right)^{k-k_r}$. We then need to mach these $k - k_r$ lines with $k \log^{4/5} k$ points, yielding a further branching of $(k \log^{4/5} k)^{k-k_r}$. Taking the product of both these branching factors gives $\left(\frac{k^2}{(k-k_r)\log k}\right)^{k-k_r} \cdot (k \log^{4/5} k)^{k-k_r} = \left(\frac{k^3}{(k-k_r)\log^{1/5} k}\right)^{k-k_r}$.

▶ **Lemma 38.** *The polynomial time dependency of* PlaneCover *is* $\mathcal{O}\big(k^4 \log^4 k\big)$ *and its space complexity is* $\mathcal{O}\big(k^6 \log^2 k\big)$ *bits.*

Proof. See the full version of the paper. ◀

6 Discussion

We have presented a general algorithm that improves upon previous best algorithms for all variations of *Curve Cover* as well as for the *Hyperplane Cover* problem in $\mathbb{R}^3$. Given good incidence bounds it should not be difficult to apply this algorithm to more geometric covering problems. However, such bounds are difficult to obtain in higher dimensions and for *Hyperplane Cover* the bound $\mathcal{O}\big(n^d/\gamma^3\big)$ is tight when no constraints are placed on the input, but it is too weak to be used even in $\mathbb{R}^3$. The bound by Elekes and Tóth works when the hyperplanes are well saturated, but the convenient relationship between saturation and degeneracy on hyperplanes does not extend past the $\mathbb{R}^3$ setting. Our hyperplane kernel guarantees a bound on the number of points on any j-flat. This overcomes the worst-case constructions for known incidence bounds, which involve placing very many points on the same line. An incidence bound for a kernelized point set might provide the needed foundation for similar *Hyperplane Cover* algorithms in higher dimensions.

References

1 Peyman Afshani, Edvin Berglin, Ingo van Duijn, and Jesper Sindahl Nielsen. Applications of incidence bounds in point covering problems. *arXiv:1603.07282*, 2016.

2 Pankaj K Agarwal and Boris Aronov. Counting facets and incidences. *Discrete & Computational Geometry*, 7(1):359–369, 1992.

3 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

4 Cheng Cao. Study on two optimization problems: Line cover and maximum genus embedding. Master's thesis, Texas A&M University, 2012.

5 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Publishing Company, Incorporated, 1st edition, 2012.

6 Herbert Edelsbrunner, Leonidas Guibas, and Micha Sharir. The complexity of many cells in arrangements of planes and related problems. *Discrete & Computational Geometry*, 5(1):197–216, 1990.

7 György Elekes and Csaba D Tóth. Incidences of not-too-degenerate hyperplanes. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 16–21. ACM, 2005.

8 Vladimir Estivill-Castro, Apichat Heednacram, and Francis Suraweera. FPT-algorithms for minimum-bends tours. *International Journal of Computational Geometry & Applications*, 21(02):189–213, 2011.

9 Jacob Fox, János Pach, Adam Sheffer, Andrew Suk, and Joshua Zahl. A semi-algebraic version of Zarankiewicz's problem. *arXiv preprint arXiv:1407.5705*, 2014.

10 Magdalene Grantson and Christos Levcopoulos. *Covering a set of points with a minimum number of lines*. Springer, 2006.

11 Ben Joseph Green and Terence Tao. On sets defining few ordinary lines. *Discrete & Computational Geometry*, 50(2):409–468, 2013.

12 Leonidas J Guibas, Mark H Overmars, and Jean-Marc Robert. The exact fitting problem in higher dimensions. *Computational geometry*, 6(4):215–230, 1996.

13 LJ Guibas, Mark Overmars, and Jean-Marc Robert. The exact fitting problem for points. In *Proc. 3rd Canadian Conference on Computational Geometry*, pages 171–174, 1991.

14 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: The easy kernel is essentially tight. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1596–1606. SIAM, 2014.

15 VS Anil Kumar, Sunil Arya, and Hariharan Ramesh. Hardness of set cover with intersection 1. In *Automata, Languages and Programming*, pages 624–635. Springer, 2000.

16 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

17 Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.

18 János Pach and Micha Sharir. On the number of incidences between points and curves. *Combinatorics, Probability and Computing*, 7(01):121–127, 1998.

19 József Solymosi and Terence Tao. An incidence theorem in higher dimensions. *Discrete & Computational Geometry*, 48(2):255–280, 2012.

20 Endre Szemerédi and William T Trotter Jr. Extremal problems in discrete geometry. *Combinatorica*, 3(3-4):381–392, 1983.

21 Praveen Tiwari. On covering points with conics and strips in the plane. Master's thesis, Texas A&M University, 2012.

22 Jianxin Wang, Wenjun Li, and Jianer Chen. A parameterized algorithm for the hyperplane-cover problem. *Theoretical Computer Science*, 411(44):4005–4009, 2010.