Applications of incidence bounds in point covering problems*

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

1 **MADALGO, Department of Computer Science, Aarhus University,**
{peyman,berglin,ivd,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 the minimum number of 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.

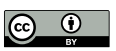
First we give a non-parameterized algorithm for both problems in $\mathcal{O}^*(2^n)$ (where the $\mathcal{O}^*(\cdot)$ notation hides polynomial factors of n) time and polynomial space, beating a previous exponential-space result. Combining this with incidence bounds similar to the famous Szemerédi-Trotter bound, we present a *Curve Cover* algorithm with running time $\mathcal{O}^*\big((Ck/\log k)^{(d-1)k}\big)$, where C is some constant. Our result improves the previous best times $\mathcal{O}^*\big((k/1.35)^k\big)$ for *Line Cover* (where $d = 2$), $\mathcal{O}^*\big(k^{dk}\big)$ 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 $\mathcal{O}^*\Big((Ck/\log^{1/10} k)^{2k}\Big)$, improving on the previous time of $\mathcal{O}^*\big((k/1.3)^{2k}\big)$.

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.xxx.yyy.p

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



Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–22

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 the minimum number of lines. It is related to *Minimum Bend Euclidean TSP* and has been studied in connection with facility location problems [8, 15]. The *Line Cover* problem is one of the few low-dimensional geometric problems that are known to be NP-complete [15]. Furthermore *Line Cover* is APX-hard, i.e., it is NP-hard to approximate within factor $(1+\varepsilon)$ for arbitrarily small ε [13]. *Line Cover* has a kernel with size polynomial in k, its solution size, so it is fixed-parameter tractable.

One generalisation 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 generalisation is to cover points with algebraic curves, e.g. circles, ellipses, parabolas, or bounded degree polynomials. These can be categorised 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 $\mathcal{O}^*\left(k^{dk}\right)$ [14]. This algorithm generalizes to generic settings, such as *Curve Cover* and *Hyperplane Cover*, retaining the same running time where d is the degree of the freedom of the curves.

The first improvement to the aforementioned generic algorithm reduced the running time to $\mathcal{O}^*\left((k/2.2)^{dk}\right)$ for the *Line Cover* problem [10]. The best algorithm for the *Hyperplane Cover* problem, including *Line Cover*, runs in $\mathcal{O}^*\left(k^{(d-1)k}/1.3^k\right)$ time [20]. 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 $\mathcal{O}(k \log k)$. [19] presents algorithms for parabola cover in time $\mathcal{O}^*\left((k/1.38)^{(d-1)k}\right)$ and conic cover in time $\mathcal{O}^*\left((k/1.15)^{(d-1)k}\right)$.

Incidence Bounds. Szemerédi and Trotter gave an asymptotic (tight) upper bound on the number of incidences (an *incidence* is a point-line pair where the point lies on the line) in the real plane in their seminal paper [18]. This has inspired a long list of similar upper bounds for incidences between points and several types of varieties in different spaces [7, 9, 16, 17].

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 $\mathcal{O}^*\left((Ck/\log k)^{(d-1)k}\right)$ and *Plane Cover* in time $\mathcal{O}^*\left((Ck/\log^{1/10} k)^{2k}\right)$, both using polynomial space. The main idea is to deploy 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. To our knowledge, we are the first to show usefulness of incidence bounds in the context of covering problems. 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.

2 Preliminaries

2.1 Definitions

We begin by formally stating the *Curve Cover* and *Hyperplane Cover* problems. Let P be a set of points in any dimension, and k, d, s be non-negative integers. A set of algebraic curves $\mathcal{C}$ have d degrees of freedom and multiplicity-type s 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 set $\mathcal{C}$ could be an infinite set and it corresponds 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 1** (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 2** (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 3** (Any-flat Hyperplane Cover Problem)**.** For $k \in \mathbb{N}$ and a set $P = \bigcup_{i=0}^{d-2} P_i$, 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 [15] and APX-hard [13], the same applies to its three generalizations as well.

2.2 Kernels

In parameterized complexity theory, a parameterized problem instance $\langle I, k \rangle$ has a *polynomial kernel* if it can be reduced in polynomial time to an instance $\langle I', k' \rangle$ such that $|I'|$ and k' are bounded by polynomial functions of k and $\langle I, k \rangle$ is a yes-instance if and only if $\langle I', k' \rangle$ is a yes-instance. Any problem with a polynomial kernel is *fixed-parameter tractable*, i.e. has an algorithm with running time on the form $f(k)n^{\mathcal{O}(1)}$ for some function f.

▶ **Lemma 1.** *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 Appendix. ◀

For *Any-flat Hyperplane Cover* a size k^d kernel is presented in [14]. 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 gives a size $k^3 + k^2$ kernel.

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

Proof. See the appendix. ◀

Our algorithms will use both properties of the kernels. Kratsch et al. [12] showed that these kernels are essentially tight under standard assumptions in computational complexity.

▶ Theorem 3 ([12]). *Line Cover* has no kernel of size $\mathcal{O}(k^{2-\varepsilon})$ 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 [18] theorem gives an upper bound on the number of point-line incidences, $I(L, P)$, in $\mathbb{R}^2$.

▶ Theorem 4 ([18]). 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 $I(L, P) = \mathcal{O}((nm)^{2/3} + n + m)$.

Structure in a point set thus implies many incidences, demonstrating the importance of incidence bounds for our covering problems.

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 pointset P in $\mathcal{O}^*(2^n)$ time and polynomial space. The technique is highly reminiscent of those presented in [3]; we give full details for completeness. The 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. We require the intersection version of the inclusion-exclusion principle.

▶ Theorem 5 (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 nonparameterized 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 there exists a curve $c \in \mathcal{C}$ that covers every $p \in Q$.

Define $\mathcal{U}$ as the set of k-tuples of coverable sets: $\mathcal{U} = \{\langle Q_1, \ldots, Q_k\rangle \mid \forall i : Q_i \text{ is coverable}\}$. Note that there is no restriction on pairwise intersection between two coverable sets in a tuple. 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 6.** *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 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 5 and Lemma 6 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 the set $\bigcap_{i\in X} \overline{A_i}$ is the set of tuples that do not intersect any 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 $c(P \setminus X)^k$ 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 7.** *$q(X, \pi, R)$ can be computed in $\mathcal{O}(|X|)$ time and linear space.*

Proof. Let X' be the set such that $X' \cup R$ is the union of every coverable set for which R is the representative. The number of subsets of X' is the number of coverable sets with representative R, i.e. $q(X, \pi, R) = 2^{|X'|}$ for valid representatives R.

If $|R| > s+1$ or R is not a coverable set, $q(X, \pi, R) = 0$ since R is not a valid representative. If $|R| \leq s$, $q(X, \pi, R) = 1$. Finally if $|R| = s+1$, construct the set X': for all $p \in P$ with $\pi(p) > \pi(r_i)$, add p to X' if and only if there is a curve $c \in \mathcal{C}$ such that c covers $\{r_1, \ldots, r_i, p\}$. Then $q(X, \pi, R) = 2^{|X'|}$. The time complexity is $\mathcal{O}(i|X|) = \mathcal{O}(s|X|) = \mathcal{O}(|X|)$ since $i \leq s+1$ is bounded by the constant $s+1$, and the space complexity is bounded by the size of $X' \subseteq X$. ◀

▶ **Lemma 8.** *$c(X)$ may be computed in $\mathcal{O}\big(|X|^{s+2}\big)$ time and linear space.*

Proof. See the Appendix. ◀

▶ Theorem 9. 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 may 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 5, Lemma 6 and Lemma 8. ◀

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 $c(X)^k$, so we can run n simultaneous partial sums representing each possible k and return the lowest k with a positive sum. This increases the space complexity by factor n while the time complexity is unaffected.

3.2 Any-flat Hyperplane Cover

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 $\{r_1, \ldots, r_{j-1}\}$) is has lower dimension than the affine hull of Q. If so, let r_j be the first flat in Q that is not in the affine hull of $\{r_1, \ldots, r_{j-1}\}$.

▶ **Lemma 10.** *$q(X, \pi, R)$ can be computed in $\mathcal{O}(|X|)$ time and linear space.*

Proof. As in Lemma 7, for a valid representative R let X' be the set such that $X' \cup R$ is the union of all coverable sets with the representative R. Then $q(X, \pi, R) = 2^{|X'|}$. For invalid representatives, $q(X, \pi, R) = 0$.

For every $p \in X$ where $\pi(r_{j-1}) < \pi(p) < \pi(r_j)$ for some j, $p \in |X'|$ if and only if p is on the affine hull of $\{r_1, \ldots, r_{j-1}\}$. For every $p \in X$ where $\pi(r_i) < \pi(p)$, $p \in X'$ if and only if p is on the affine hull of R. ◀

There are $\mathcal{O}\left(\binom{n}{d}\right)$ representatives R with $q(X, \pi, R) > 0$ so the following two results hold; their proofs are analogous to Lemma 8 and Theorem 9.

▶ **Lemma 11.** *$c(X)$ may be computed in $\mathcal{O}\left(|X|^{d+1}\right)$ time and linear space.*

▶ Theorem 12. 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 $\mathcal{O}\left(2^n n^{d+1}\right)$ time and $\mathcal{O}(nk)$ bits of space.

4 Curve Cover

Recall that we are considering (d, s)-curves, where d and s are held to be 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 call Inclusion-Exclusion as a base case. To describe this technique and the intuition behind it, we first provide a framework based on the following theorem by Pach and Sharir.

▶ Theorem 13 ([16]). 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.

need to explain that projecting might create "different looking" curves. it can increase s but not d so the bound still holds

▶ **Definition 4.** 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 kernelization in Lemma 1, it follows that every candidate is sk-poor. The following gives a bound on the number of candidates.

▶ **Lemma 14.** *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*

$$m = \mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right).$$

Proof. See the appendix. ◀

Algorithm Sketch. We exploit the following observation: given a k-cover, some curves might cover significantly more points 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 makes the point set smaller in relation to the budget. As a result, calling Inclusion-Exclusion becomes viable. It also makes sense from a combinatorial point of view, because from Lemma 14 it follows that the search space is small for "rich" curves but large for "poor" curves.

The idea to branch on rich curves first has another important consequence. Suppose that in a point set P the richest curve ℓ covers γ points. This immediately implies that if there are strictly more than $k\gamma$ points in P, that it is impossible to cover P with only γ-poor curves. The proposed idea is to first branch on curves as rich as ℓ by selecting some subset of them. By the earlier observation, selecting the rich curves is good for reducing the point set. The other crucial observation is the following. If no such rich curves are selected, the point set is still practically reduced by k. This follows from the fact that from this point on, only $(\gamma - 1)$-poor curves are to be considered. Therefore, if more than $k(\gamma - 1)$ points remain this branch is not solvable. Thus, by discarding all γ-rich curves, the upper bound on the size of P is reduced by k.

The Algorithm. Let r be a parameter. The exact value is set in the proof of Theorem 19, 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 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. We describe the operations of the recursive algorithms in text here; pseudocode is given in the appendix.

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 onwards 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 subproblem 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 15.** *Algorithm* CurveCover *decides whether P has a k-cover of curves from $\mathcal{C}$.*

Proof. Suppose P has a k-cover $\mathcal{C}$. The proof is by induction on the recursion. Since our algorithm tries all budget partitions, assume we can freely set k_i at level i. As the induction hypothesis, assume that the current partial solution $\mathcal{C}_{i-1}$ is a subset of $\mathcal{C}$ and that $\mathcal{C}$ contains no curves that are γ_{i-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 it holds that $|P_i| \le \gamma_{i-1}K_i$, and thus the algorithm does not reject incorrectly. Furthermore, if Inclusion-Exclusion is called it accepts.

Otherwise, let $D \subseteq \mathcal{C}\setminus\mathcal{C}_{i-1}$ be the curves that are γ_i-rich when restricted to P_i and set $k_i=|D|$. One branch picks D from the set of candidates S; it therefore correctly constructs $\mathcal{C}_i=\mathcal{C}_{i-1}\cup D$, leaving $\mathcal{C}_i$ 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 its subset P_{i+1}, 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 it into a full k-cover by using Inclusion-Exclusion as an oracle.

Running time. To analyse 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 of a tree is the number of leaf nodes times the running time of Inclusion-Exclusion. 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). It turns out the most involved part is proving an upper bound on the number of leaves of $\mathcal{T}_j$.

▶ **Lemma 16.** *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*

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

We leave the proof for Lemma 16 for later.

▶ **Lemma 17.** *The time complexity of a complete subtree $\mathcal{T}_j$ is $\mathcal{O}^*\left((c_4 k/\log k)^{(d-1)K_j}\right)$, where $c_4 = c_4(d, s)$ is a constant that depends on the family $\mathcal{C}$.*

Proof. Inclusion-Exclusion at depth r in time

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

By Lemma 16, the number of leaves in $\mathcal{T}_j$ is

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

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

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

Suppose $k - k_r = o(k)$. We leave the full proof for the Appendix. In the other case $(k - k_r) \geq c_3 k$ for some $c_3 > 0$ and the running time solves to

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

for some $c_4 = c_2/c_3$. ◀

▶ **Lemma 18.** *Let L_j be a leaf of $\mathcal{T}$ that calls* Inclusion-Exclusion*. The running time of $\mathcal{T}_j$ dominates that of L_j.*

Proof. By Lemma 17, 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 $\mathcal{O}^*\left(2^{\frac{d-1}{2} K_j \log k}\right) = \mathcal{O}^*\left((k^{1/2})^{(d-1)K_j}\right)$ time, which is bounded by that for $\mathcal{T}_j$. ◀

▶ Theorem 19. The time complexity of CurveCover is $\mathcal{O}^*\left((Ck/\log k)^{(d-1)k}\right)$ for a constant $C = C(d, s)$ that depends on the family $\mathcal{C}$.

Proof. Fix a budget partition $\langle k_1, \ldots, k_r \rangle$. By Lemma 18, 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, which by the "stars and bars" theorem are only $\binom{k+r-1}{k}$, a quasipolynomial 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 20.** *The polynomial time dependency of* CurveCover *is* $\mathcal{O}\big((k\log k)^{2+s}\big)$ *and its space complexity is* $\mathcal{O}\big(k^4\log^2 k\big)$ *bits.*

Proof. See the Appendix. ◀

What is left is to prove Lemma 16. The whole setup of the algorithm is to be able to use the incidence bound from Lemma 14 to bound the number of candidates at recursion level i. With a bound on the number of candidates we can bound the branching at level i as follows.

▶ **Lemma 21.** *For some constant* $c_1 = c_1(d,s)$ *and* $\alpha = 2^{d-1}$*. The branching factor of an internal node of* $\mathcal{T}$ *at level* i *is bounded by* $\left(\frac{\alpha^i c_1 k}{k_i}\right)^{k_i}$*.*

Proof sketch. The result almost immediately follows from applying Lemma 14 and bounding the binomial. See appendix for full proof. ◀

For the worst case analysis, we need to know for which budget partition the product of branching factors is maximised. We therefore prove the following.

▶ **Lemma 22.** *Let* $k_1,\dots,k_t$ *be nonnegative integers with a fixed sum, and* $\alpha > 1$ *and* β *constant real numbers. It holds that:*

$$\Pi = \prod_{i=1}^{t}\left(\frac{\alpha^i\beta}{k_i}\right)^{k_i} \le \left(\frac{\beta}{k_0}\right)^{\sum_{i=1}^{t}k_i} \quad \textit{where } k_0 = \frac{\alpha-1}{\alpha^t-\alpha}\sum_{i=1}^{t}k_i$$

Proof. See appendix. ◀

The product Π here is essentially the bound on T from Lemma 16 that we are looking for. Before we can apply it however, we need to solve for k_0. Note that k_0 depends on r (the deepest recursion level of the algorithm) and the sum $\sum_{i=1}^{r}k_i$ (i.e. the budget used in the recursive part). To determine the deepest recursion level, recall that the algorithm keeps recursing until either too few or too many points remain. That means that we can derive the maximal recursion depth by solving what the recursion depth is where both those bounds are equal (i.e. no more branching can occur). The algorithm switches no later than depth r, where at most $\frac{(d-1)}{2}K_r\log k$ points remain. Conversely, if the instance was not immediately rejected, at most $K_t sk/2^{r-1}$ points remain. By solving for r and using the expression for k_0 from Lemma 22 we can prove the following.

▶ **Lemma 23.** *Let* r *be the deepest level of recursion in* CC-Recursive*,* k_0 *is bounded by:*

$$k_0 = \mathcal{O}\left(\frac{(k-k_r)((d-1)\log k)^{d-1}}{(2sk)^{d-1}}\right)$$

Proof. See appendix. ◀

We now have enough machinery to prove Lemma 16.

Proof of Lemma 16. Let $\alpha = 2^{d-1}$. Lemma 21 gives a bound of $\left(\frac{\alpha^i c_1 k}{k_i}\right)^{k_i}$ on the branching of an internal node at recursion level i. Taking the product of branching factors on the recursion levels j through r gives a bound on T_j. We can directly apply the bound from Lemma 22 to bound this product and therefore bound T_j.

$$T_j \le \prod_{i=j}^{r-1}\left(\frac{\alpha^i c_1 k}{k_i}\right)^{k_i} \le \left(\frac{c_1 k}{k_0}\right)^{\sum_{i=j}^{r-1}k_i} = \left(\frac{c_1 k}{k_0}\right)^{K_j - k_r}$$

Now substitute k_0 from Lemma 23 and collect any constants in $c_2 = c_2(d, s)$ to get

$$T \leq \left(\frac{c_1 k (2sk)^{d-1}}{(k-k_r)((d-1)\log k)^{d-1}}\right)^{K_j - k_r} = \left(\frac{c_2 k^d}{(k-k_r)\log^{d-1} k}\right)^{K_j - k_r}$$

◀

5 Hyperplane Cover

One of the generalisation of *Line Cover* was discussed in the previous section. In this section we discuss its other generalizaton *Hyperplane Cover*, and give an algorithm for the threedimensional 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 [1, 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 [1, 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 14 (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 specialised bound appears in [7], where the authors study the number of incidences between points and hyperplanes with a certain *saturation*.

▶ **Definition 5.** 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 24 (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. An issue is that it is not easy to verify if a hyperplane is σ-saturated. In the same paper as Theorem 24, the authors give another bound based on a more managable property called *degeneracy*.

▶ **Definition 6.** 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 25 (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 25. This algorithm is similar to the algorithm for *Curve Cover*, and it is assumed that the reader be sufficiently familiar with CC-Recursive before reading this section.

Recall that by Lemma 2, *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 7.** 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 the line.

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, 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 L_i be the set of all lines in $\mathcal{L}$ that were added at depth f or earlier. Remove L_i from $\mathcal{L}$. For each way of placing $|L_i|$ planes $\mathcal{H}$ such that every plane contains one line in L_i and at least one point in P, let $P' = P \setminus \mathcal{H}$ be 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 correctness, we assume that the algorithm is non-deterministic and guesses all the planes and rich lines "correctly". At any stage, P might contain some ghost points that should not be covered using the budget remaining K_i – the budget to cover them was already paid when removing the points on their corresponding degenerate lines. The check that P is small enough needs to account for the maximum number of ghost points that can remain in the instance, or the algorithm might erroneously reject.

▶ **Lemma 26.** *The number of ghost points at depth i of* PC-Recursive *is at most $|\mathcal{L}|\gamma_{i-1}$.*

Proof. Consider a degenerate line $L \in \mathcal{L}$ guessed at some recursion depth $j < i$. Line L was on a δ_j-degenerate plane, i.e. it covered at least $\gamma_j\delta_j$ points and has at most $2\gamma_j^{4/5}$ ghost points. Since L was not removed from $\mathcal{L}$ at depth $i-1$, we get $j > f_{i-1} = \left\lceil \frac{5((i-1)-1)-2\log k}{4} \right\rceil$. Some simple algebra gives $\frac{k^2}{2^{i-2}} \geq \left(\frac{k^2}{2^j}\right)^{4/5}$, i.e. $\gamma_{i-1} \geq 2\gamma_j^{4/5}$. Hence any line in $\mathcal{L}$ has left at most γ_{i-1} ghost points in the instance, and the sum of ghost points is at most $|\mathcal{L}|\gamma_{i-1}$. ◀

The next three lemmas show that the manipulations of $\mathcal{L}$ and the input to the base case algorithm indeed yield the correct result. We make a new definition to state them.

▶ **Definition 8.** For a set of points P and a set of lines $\mathcal{L}$, we say that $\langle P, \mathcal{L}\rangle$ has a k-cover $\mathcal{C}$ if there exists a set of $|\mathcal{L}|$ planes $\mathcal{H}$ such that every $h \in \mathcal{H}$ intersects a distinct $\ell \in \mathcal{L}$, and $P \setminus (P \cap \mathcal{H})$ has a k-cover $\mathcal{C}$. We call $\mathcal{H}$ the *accompanying set* of $\mathcal{C}$.

▶ **Lemma 27.** *Let $k > 0$ and $P \neq \emptyset$. $\langle P, \mathcal{L}\rangle$ has a k-cover $\mathcal{C}$ if and only if for every nonempty $h \in \mathcal{C}$ and for every nonempty line ℓ on h it holds that $\langle P \setminus (P \cap \ell), \mathcal{L} \cup \{\ell\}\rangle$ has a $(k-1)$-cover.*

Proof. Let $\mathcal{C}$ be a k-cover of $\langle P, \mathcal{L}\rangle$ with accompanying set $\mathcal{H}$. Take any plane $h \in \mathcal{C}$ such that $|h \cap P| > 0$ and any line ℓ on h such that $|\ell \cap P| > 0$. Then $\mathcal{C} \setminus \{h\}$ is a $(k-1)$-cover of $\langle P \setminus (P \cap \ell), \mathcal{L} \cup \{\ell\}\rangle$ with the accompanying set $\mathcal{H} \cup \{h\}$.

Let ℓ be any line such that $\langle P \setminus (P \cap \ell), \mathcal{L} \cup \{\ell\}\rangle$ has a $(k-1)$-cover $\mathcal{C}$ with accompanying set $\mathcal{H}$. Then by definition there exists a $h \in \mathcal{H}$ that covers ℓ, and $\mathcal{H} \setminus \{h\}$ covers $\mathcal{L}$ since

every plane in $\mathcal{H}$ intersects a distinct $\mathcal{L} \cup \{\ell\}$. Let $P_h = P \setminus (P \cap (\mathcal{H} \setminus \{h\}))$. Since $\mathcal{C} \cup \{h\}$ covers P_h, it is a k-cover of P with accompanying set $\mathcal{H} \setminus \{h\}$. ◀

▶ **Lemma 28.** *$\langle P, \mathcal{L}\rangle$ has a k-cover $\mathcal{C}$ with accompanying set $\mathcal{H}$ if and only if for any $\ell \in \mathcal{L}$ where $h \in \mathcal{H}$ covers ℓ, $\langle P \setminus (P \cap h), \mathcal{L} \setminus \{\ell\}\rangle$ has a k-cover with accompanying set $\mathcal{H} \setminus \{h\}$.*

Proof. $\mathcal{C} \cup \{h\}$ is a $(k+1)$-cover of $\langle P, \mathcal{L} \setminus \{\ell\}\rangle$. Therefore, by removing the points covered by h from P, $P_h = P \setminus (P \cap h)$, $\mathcal{C}$ is a k-cover of P_h. Since h covers ℓ, $\mathcal{H} \setminus \{h\}$ covers $\mathcal{L} \setminus \{\ell\}$ and is an accompanying set to the k-cover of P_h. ◀

▶ **Lemma 29.** *$\langle P, \mathcal{L}\rangle$ has a k-cover if and only if $P \cup \mathcal{L}$ has a $(k + |\mathcal{L}|)$-cover $\mathcal{C}$ and $\mathcal{L}$ has no $(|\mathcal{L}| - 1)$-cover.*

Proof. Let $\mathcal{C}$ be a k-cover of $\langle P, \mathcal{L}\rangle$ with accompanying set $\mathcal{H}$. Then $\mathcal{C} \cup \mathcal{H}$ is a $(k + |\mathcal{L}|)$-cover of P. By definition $\mathcal{H}$ also covers $\mathcal{L}$, so $\mathcal{C} \cup \mathcal{H}$ is a $(k + |\mathcal{L}|)$-cover of $P \cup \mathcal{L}$.

Suppose $\mathcal{L}$ has no $(|\mathcal{L}| - 1)$-cover and let $\mathcal{C}$ be a $(k + |\mathcal{L}|)$-cover of $P \cup \mathcal{L}$. Otherwise, let $\mathcal{H} \subset \mathcal{C}$ be size $|\mathcal{L}|$ subset that covers $\mathcal{L}$. Then $\mathcal{C} \setminus \mathcal{H}$ is a k-cover of $P \setminus (P \cap \mathcal{H})$. ◀

Note that $\langle P \cup \mathcal{L}, k + |\mathcal{L}|\rangle$ is an *Any-flat Hyperplane Cover* instance, which is why we need our base-case algorithm to solve that extension to *Hyperplane Cover*.

▶ **Lemma 30.** *Algorithm* PlaneCover *decides whether P has a k-cover of planes.*

Proof. On the first level of recursion, PC-Recursive receives a $\mathcal{L}_1 = \emptyset$ – clearly P has a k-cover $\mathcal{C}$ if and only if $\langle P, \emptyset\rangle$ has the same k-cover. Suppose the k-cover contains a too-degenerate plane h with degenerate line ℓ such that $h \setminus (P \cap \ell)$ is also a too-degenerate plane: this could potentially mean that $\mathcal{L}$ has a $(|\mathcal{L}| - 1)$-cover. But as degenerate lines in $\mathcal{L}$ are extended to planes before considering whether its ghost points form a too-degenerate plane, we avoid this problem. As such, Lemma 27 and Lemma 28 assure solvability is maintained whenever the algorithm puts a new line in $\mathcal{L}$ or takes one out, andLemma 29 guarantees that Inclusion-Exclusion gives the correct answer at any time it is called.

Hence we only need to show that if a cover exists, the algorithm at some point calls Inclusion-Exclusion on a yes-instance. By viewing the algorithm as non-deterministic, it always builds a partial solution $\mathcal{C}_i \subseteq \mathcal{C}$ so that $\mathcal{C} \setminus \mathcal{C}_i$ contains no γ_{i-1}-rich planes restriced on P_i. Because of this and Lemma 26 the check that $|P| \leq (K_i + |\mathcal{L}|)\gamma_{i-1}$ never fails. By non-determinism it chooses the correct h_i not-too-degenerate planes and the ℓ_i degenerate lines to create $\mathcal{C}_i \subseteq \mathcal{C}$ such that $\mathcal{C} \setminus \mathcal{C}_i$ contains no γ_i-rich planes restricted on P_{i+1}. The algorithm non-deterministically recurses until the P is small enough. ◀

▶ Theorem 31. PlaneCover runs in $\mathcal{O}\Big((k \log k)^4 (Ck^2/\log^{1/5} k)^k\Big) = \mathcal{O}^*\Big((Ck^2/\log^{1/5} k)^k\Big)$ time for some constant C, and polynomial space.

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 $\mathcal{O}(n^d/\gamma^3)$ 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 for $0 \leq j \leq d-2$. 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. P. K. Agarwal and B. Aronov. Counting facets and incidences. *Discrete & Computational Geometry*, 7(1):359–369, 1992.
2. A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
3. C. Cao. Study on two optimization problems: Line cover and maximum genus embedding. Master's thesis, Texas A&M University, 2012.
4. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Publishing Company, Incorporated, 1st edition, 2012.
5. H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity of many cells in arrangements of planes and related problems. *Discrete & Computational Geometry*, 5(1):197–216, 1990.
6. G. Elekes and C. D. Tóth. Incidences of not-too-degenerate hyperplanes. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 16–21. ACM, 2005.
7. V. Estivill-Castro, A. Heednacram, and F. Suraweera. FPT-algorithms for minimum-bends tours. *International Journal of Computational Geometry & Applications*, 21(02):189–213, 2011.
8. J. Fox, J. Pach, A. Sheffer, A. Suk, and J. Zahl. A semi-algebraic version of Zarankiewicz's problem. *arXiv preprint arXiv:1407.5705*, 2014.
9. M. Grantson and C. Levcopoulos. *Covering a set of points with a minimum number of lines*. Springer, 2006.
10. B. J. Green and T. Tao. On sets defining few ordinary lines. *Discrete & Computational Geometry*, 50(2):409–468, 2013.
11. S. Kratsch, G. Philip, and S. 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.
12. V. A. Kumar, S. Arya, and H. Ramesh. Hardness of set cover with intersection 1. In *Automata, Languages and Programming*, pages 624–635. Springer, 2000.
13. S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.
14. N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.
15. J. Pach and M. Sharir. On the number of incidences between points and curves. *Combinatorics, Probability and Computing*, 7(01):121–127, 1998.
16. J. Solymosi and T. Tao. An incidence theorem in higher dimensions. *Discrete & Computational Geometry*, 48(2):255–280, 2012.
17. E. Szemerédi and W. T. Trotter Jr. Extremal problems in discrete geometry. *Combinatorica*, 3(3-4):381–392, 1983.
18. P. Tiwari. On covering points with conics and strips in the plane. Master's thesis, Texas A&M University, 2012.
19. J. Wang, W. Li, and J. Chen. A parameterized algorithm for the hyperplane-cover problem. *Theoretical Computer Science*, 411(44):4005–4009, 2010.

A Algorithms

Algorithm 1 Recursive Curve Cover

procedure CC-RECURSIVE($P, \langle k_1, \ldots, k_r \rangle, i$)
  if $|P| > K_i sk/2^{i-1}$ **then**
    return no
  if $|P| < K_i \log k$ **then**
    return INCLUSION-EXCLUSION(P, k')
  let S be the set of γ_i-rich-γ_{i-1}-poor candidates
  for all S' s.t. $S' \subseteq S, |S| = k_i$ **do**
    if CC-RECURSIVE($P \setminus (\bigcup S'), \langle k_1, \ldots, k_r \rangle, i+1$) **then**
      return yes
  return no

Algorithm 2 Recursive Plane Cover

procedure PC-RECURSIVE($P, \mathcal{L}, \langle h_1, \ell_1, \ldots, h_r, \ell_r \rangle, i$)
  if $|P| > (K_i + |\mathcal{L}|)\gamma_{i-1}$ **then**
    return no
  if $|P| < K_i \log k$ **then**
    return INCLUSION-EXCLUSION($P \cup \mathcal{L}, K_i + |\mathcal{L}|$)
  let A be the set of lines in $\mathcal{L}$ added at depth $\left\lceil \frac{5(i-1)-2\log k}{4} \right\rceil$ or earlier
  if $|A| \geq 1$ **then**
    for all sets of $|A|$ planes $\mathcal{H}$ s.t. each $h \in \mathcal{H}$ covers a $\ell \in A$ and a $p \in P$ **do**
      if PC-RECURSIVE($P \setminus (\bigcup \mathcal{H}), \mathcal{L} \setminus A, \langle h_1, \ell_1, \ldots, h_r, \ell_r \rangle, i$) **then**
        return yes
    return no
  let H be the set of not-too-degenerate planes
  let L be the set of $\gamma_i\delta_i$-rich γ_{i-1}-poor lines
  for all $\langle H', L' \rangle$ s.t. $H' \subseteq H, |H'| = h_i$ and $L' \subseteq L, |L'| = \ell_i$ **do**
    let P' be the set of points that are in P but not on any $h \in H'$ or on any $\ell \in L'$
    if PC-RECURSIVE($P', \mathcal{L} \cup L', \langle h_1, \ell_1, \ldots, h_r, \ell_r \rangle, i+1$) **then**
      return yes
  return no

B Omitted proofs

▶ **Lemma 1.** *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. Suppose some curve $c \in \mathcal{C}$ covers at least $sk+1$ points in P. These points cannot be covered by k other curves $c_1, \ldots, c_k$ as the pairwise intersection of c_i with c contains at most s points. Therefore every k-cover must include c; remove the points that it covers and decrement k. We can repeat that for every curve that covers $sk+1$ points, until every curve in $\mathcal{C}$ covers at most sk of the remaining points. Thus, if the number of remaining points is more than sk^2, the instance has no k-cover and can be immediately rejected. Otherwise, we are left with an instance of sk^2 points. ◀

▶ **Lemma 8.** *$c(X)$ may be computed in $\mathcal{O}\big(|X|^{s+2}\big)$ time and linear 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 7 each term of the sum is computable in $\mathcal{O}(|X|)$ time and linear space. ◀

▶ **Lemma 14.** *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*

$$m = \mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right).$$

Proof. If m curves pass through γ or more points, this generates at least $m\gamma$ incidences. By Theorem 13 we get

$$m\gamma = \mathcal{O}\Big(n^{d/(2d-1)} m^{(2d-2)/(2d-1)} + n + m\Big).$$

We deal with the three terms in the $\mathcal{O}(\cdot)$ separately. If $m\gamma = \mathcal{O}\big(n^{d/(2d-1)} m^{(2d-2)/(2d-1)}\big)$, the expression simplifies to $m = \mathcal{O}\big(n^d \gamma^{-(2d-2)}\big)$. If $m\gamma = \mathcal{O}(n)$ we have that $m = \mathcal{O}(n/\gamma)$. If $m\gamma = \mathcal{O}(m)$ then γ is a constant. Since at most s curves pass through the same d points, m is bounded by the total number of distinct curves $s\binom{n}{d} = \mathcal{O}\big(n^d\big)$. Therefore, this case is covered by the first term, giving the total bound $m = \mathcal{O}\Big(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\Big)$. ◀

▶ **Lemma 2.** *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. This kernel boils down to creating a maximum intersection $s = \sum_{i=0}^{d-2} k^i$ between two hyperplanes. Then we have a kernel of size sk^2 where no hyperplane covers more than sk points, in exactly the same way as Lemma **??**. For a point set P in $\mathbb{R}^d$, call a t-flat *heavy* if it covers at least $\sum_{i=0}^{t} k^t$ points in P. P is t-ready if every heavy t-flat covers exactly $\sum_{i=0}^{t} k^t$ points. When P is $(d-2)$-ready, we have the desired intersection bound s and are done. Clearly any point set is 0-ready, so we only need to show how to modify a t-ready pointset into one that is equivalent and $(t+1)$-ready.

Let P be t-ready and suppose it contains a heavy $(t+1)$-flat f. Since by assumption two $(t+1)$-flats intersect in at most $\sum_{i=0}^{t} k_i$ points, any hyperplane cover of P must have a

hyperplane covering f. This property is maintained by removing arbitrary points R on f so that f covers exactly $\sum_{i=0}^{t+1} k_i$ points. Consider that some of the points R were on another heavy flat f_1, which as a consequence is no longer heavy. This could mean that $P \setminus R$ has a cover where P did not, and must be rectified. We do this by re-adding enough points to f_1 so that it too covers exactly $\sum_{i=0}^{t+1} k_i$. Put these new points in general position on f_1 to avoid creating new heavy $(t+1)$-flats or increasing the number of points on any heavy t-flat. Once all heavy $(t+1)$-flats are reduced to $\sum_{i=0}^{t+1} k_i$ points in this manner, the new pointset is $(t+1)$-ready and a yes-instance if and only if P is. ◀

▶ **Lemma 20.** *The polynomial time dependency of* CurveCover *is* $\mathcal{O}\big((k \log k)^{2+s}\big)$ *and its space complexity is* $\mathcal{O}\big(k^4 \log^2 k\big)$ *bits.*

Proof. The height of the tree is $r = \mathcal{O}(\log k)$. Inner nodes have polynomial time and space, while leaves have exponential time and polynomial space. Hence the polynomial time dependency is exactly the polynomial time dependency of the leaves. Inclusion-Exclusion runs in $\mathcal{O}\big(n^{s+2} 2^n\big)$ and $n = \mathcal{O}(k \log k)$ points remain when it is called; the polynomial dependency is $\mathcal{O}\big((k \log k)^{s+2}\big)$.

Inclusion-Exclusion requires only $\mathcal{O}(nk) = \mathcal{O}\big(k^2 \log k\big)$ bits of storage, while an inner node stores its set of candidates S. A trivial bound on the siez of any S is $\binom{sk^2}{2}$ elements, which can be stored in $\mathcal{O}\big(k^4 \log k\big)$ bits. Since $r = \Theta(\log k)$, we use no more than $\mathcal{O}\big(k^4 \log^2 k\big)$ bits to store them. ◀

▶ **Lemma 21.** *For some constant* $c_1 = c_1(d, s)$ *and* $\alpha = 2^{d-1}$*. The branching factor of an internal node of* $\mathcal{T}$ *at level* i *is bounded by* $\left(\frac{\alpha^i c_1 k}{k_i}\right)^{k_i}$*.*

Proof. Let the budget partition $\langle k_1, \ldots, k_r \rangle$ be fixed and consider recursion level i. At this point at most $K_i \gamma_{i-1} \leq sk^2/2^{i-1}$ points remain and all candidate curves in S are γ_i-rich. By Lemma 14, $|S|$ is bounded by one of the following:

$$\mathcal{O}\left(\frac{(K_i\gamma_i)^d}{\gamma_i^{2d-1}}\right) = \mathcal{O}\left(\left(\frac{sk^2}{2^{i-1}}\right)^d \left(\frac{sk}{2^i}\right)^{-(2d-1)}\right) = \mathcal{O}\big(\alpha^i s^{1-d} 2^d k\big)$$

$$\mathcal{O}\left(\frac{(K_i\gamma_i)}{\gamma_i^{2d-1}}\right) = \mathcal{O}\left(\left(\frac{sk^2}{2^{i-1}}\right)\left(\frac{sk}{2^i}\right)^{-1}\right) = \mathcal{O}(k)$$

Let c_1 be the smallest constant (dependent on the constants s and d) such that $\alpha^i \frac{c_1}{e} k$ is always greater than the implicit functions of both bounds. At level i, the algorithm will branch on all possible ways of picking k_i curves out of $|S| \leq \alpha^i \frac{c_1}{e} k$ candidates. We can bound this by

$$\binom{\alpha^i \frac{c_1}{e} k}{k_i} \leq \left(\frac{e\alpha^i \frac{c_1}{e} k}{k_i}\right)^{k_i} = \left(\frac{\alpha^i c_1 k}{k_i}\right)^{k_i}$$

◀

▶ **Lemma 17.** *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 of case $(k - k_r) = o(K_j)$**.** It was established that the running time for $\mathcal{T}_j$ is

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

Since we are in the case where $(k - k_r) = o(K_j)$ we get:

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

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 $\mathcal{O}^*\big(2^{(d-1)K_j(\log k - \log\log k)}\big) = \mathcal{O}^*\big((k/\log k)^{(d-1)K_j}\big)$. ◀

▶ **Lemma 22.** *Let $k_1, \ldots, k_t$ be nonnegative integers with a fixed sum, and $\alpha > 1$ and β constant real numbers. It holds that:*

$$\Pi = \prod_{i=1}^{t} \left(\frac{\alpha^i \beta}{k_i}\right)^{k_i} \leq \left(\frac{\beta}{k_0}\right)^{\sum_{i=1}^{t} k_i} \quad \textit{where } k_0 = \frac{\alpha - 1}{\alpha^t - \alpha} \sum_{i=1}^{t} k_i$$

Proof. Let $\sum_{i=1}^{t} k_i = k$ and assume that $k_1, ..., k_t$ maximise Π; we explicitly compute their value in terms α, r and k. Consider k_i and k_{i+1} that sum to κ, and the function $f : [0, \kappa] \mapsto \mathbb{R}$, $f(x) = \left(\frac{n}{x}\right)^x \left(\frac{\alpha n}{\kappa - x}\right)^{\kappa - x}$. Because k_i and k_{i+1} maximise Π, the function f is maximal at $f(k_i)$. To find the maximum we take the logarithm of the function, differentiate, and equate to 0.

$$\begin{aligned}
\log f(x) &= x(\log n - \log x) + (\kappa - x)(\log \alpha n - \log(\kappa - x)) \\
(\log f(a))' &= (\log n - \log x) - 1 - (\log \alpha n - \log(\kappa - x)) + 1 \\
&= \log \frac{\kappa - x}{x\alpha} = 0
\end{aligned}$$

From this we derive that $f(x)$ is maximal when $\kappa - x = x\alpha$, and thus that $\alpha k_i = k_{i+1}$. Since the above argument holds for all $i \geq 1$, only k_1 can be freely set and all other k_i are of the form $\alpha^{i-1} k_1$. Define $k_0 = \alpha k_1$, so that for all $i \geq 1$ we have $k_i = \alpha^i k_0$. The expression for k_0 as it appears in the lemma can be derived from $\sum_{i=1}^{r} \alpha^i k_0 = k$ by finding the correct geometric series. Filling in the computed values for k_i in the definition of Π we get:

$$\Pi = \prod_{i=1}^{t} \left(\frac{\alpha^i \beta}{\alpha^i k_0}\right)^{\alpha^i k_0} = \prod_{i=1}^{t} \left(\frac{\beta}{k_0}\right)^{\alpha^i k_0} = \left(\frac{\beta}{k_0}\right)^{\sum_{i=1}^{t} k_i}$$

◀

▶ **Lemma 23.** *Let r be the deepest level of recursion in* CC-Recursive*, k_0 is bounded by:*

$$k_0 = \mathcal{O}\left(\frac{(k - k_r)((d-1)\log k)^{d-1}}{(2sk)^{d-1}}\right)$$

Proof. The algorithm does not recurse if either the number of points left is less than $\frac{(d-1)}{2} k_r \log k$, or more than $k_r sk/2^{r-1}$. This means that it cannot recurse if

$$\frac{(d-1)}{2} k_r \log k > k_r sk/2^{r-1}$$

Setting these quantities equal and solving for r will thus give an upper bound on the recursion depth of any branch.

$$\begin{aligned}
\frac{(d-1)}{2} k_r \log k &= k_r sk/2^{r-1} \\
2^r &= \frac{4sk}{(d-1)\log k} \\
r &= \log \frac{k}{\log k} + \log \frac{4s}{d-1}
\end{aligned}$$

Thus at most the budgets $k_1, ..., k_{r-1}$ summing to $k - k_r$ can be used by the recursive part of the algorithm. Plugging these values into Lemma 22 yields:

$$k_0 = \frac{k - k_r}{\sum_{i=1}^{r-1} \alpha^i} = \frac{(k - k_r)(\alpha - 1)}{\alpha^r - \alpha}$$

We now expand $(2^{d-1})^r$

$$(2^{d-1})^r = (2^{d-1})^{\log \frac{k}{\log k} + \log \frac{4s}{d-1}} = \left(\frac{4sk}{(d-1)\log k}\right)^{d-1} = 2^{d-1}\left(\frac{2sk}{(d-1)\log k}\right)^{d-1}$$

We substitute $\alpha^r = (2^{d-1})^r$ in the expression for k_0 to get:

$$k_0 = \frac{(k - k_r)(2^{d-1} - 1)}{2^{d-1}\left(\frac{2sk}{(d-1)\log k}\right)^{d-1} - 2^{d-1}} = \Theta\left(\frac{(k - k_r)}{\left(\frac{2sk}{(d-1)\log k}\right)^{d-1} - 1}\right) = \Omega\left(\frac{(k - k_r)}{\left(\frac{2sk}{(d-1)\log k}\right)^{d-1}}\right)$$

By simplifying the last expression the proof is complete. ◀

▶ Theorem 31. PlaneCover runs in $\mathcal{O}\Big((k \log k)^4 (Ck^2/\log^{1/5} k)^k\Big) = \mathcal{O}^*\Big((Ck^2/\log^{1/5} k)^k\Big)$ time for some constant C, and polynomial space.

Proof. Set $\alpha = 2^{1/5}$ and $\beta = c_1 k^{13/5}$ for some constant c_1. It holds that $\sum_{i=1}^r h_i + \sum_{i=1}^r \ell_i = \sum_{i=1}^r k_i = k - K_r$, so for convenience we define ε such that $\varepsilon(k - K_r) = \sum_{i=1}^r h_i$.

By the bound we have that the number of not-too-degenerate planes at level i is:

$$\left(\frac{1}{1-\delta_i}\right)^4 \frac{(k\gamma_{i-1})^3}{(\gamma_i)^4} = \mathcal{O}\left(\frac{k^3}{\gamma_i^{1/5}}\right) = \alpha^i \beta$$

And by the same argument as for curves, the total branching for picking planes can be bounded by $\left(\frac{\alpha^i \beta}{h_i}\right)^{h_i}$.

$$\prod_{i=1}^r \binom{\alpha^i \beta}{h_i} \leq \prod_{i=1}^r \left(\frac{\alpha^i \beta}{\alpha^i h_0}\right)^{\alpha^i h_0} = \prod_{i=1}^r \left(\frac{\beta}{h_0}\right)^{\alpha^i h_0} = \left(\frac{\beta}{h_0}\right)^{\sum_{i=1}^r h_i} = \left(\frac{\beta}{h_0}\right)^{\varepsilon(k-K_r)}$$

The number of γ_{i+1}-rich lines at level i is $\frac{(k\gamma_{i-1})^2}{\gamma_{i+1}^3} = \frac{2k^2}{\gamma_i}$. From these we pick ℓ_i lines, giving a branching of $(\frac{2k^2}{\gamma_i \ell_i})^{\ell_i}$. The number of points at level j where $\gamma_j = \gamma_i^{4/5}$ is $k\gamma_i^{4/5}$, thus matching ℓ_i lines with this many points yields a branching of $(k\gamma_i^{4/5})^{\ell_i}$. Combining this with the branching factor above gives

$$(k\gamma_i^{4/5})^{\ell_i} \cdot (\frac{k^2}{\gamma_i \ell_i})_i^{\ell} = \left(\frac{\alpha^i \beta}{\ell_i}\right)^{\ell_i}$$

By the same technique as the curves and planes, the total branching on lines can thus be bounded by

$$\left(\frac{\beta}{\ell_0}\right)^{\sum_{i=1}^r \ell_i} = \left(\frac{\beta}{\ell_0}\right)^{(1-\varepsilon)(k-K_r)}$$

The numbers h_0 and ℓ_0 can be lower bounded by $\Omega\Big(\frac{\varepsilon(k-K_r)\log^{1/5} k}{k^{2/5}}\Big)$ and $\Omega\Big(\frac{(1-\varepsilon)(k-K_r)\log^{1/5} k}{k^{2/5}}\Big)$. Let c_2 be the constant that contains the implicit constant in these lower bounds, together with the constant c_1. The total branching can now be bounded as follows:

$$\left(\frac{\beta}{h_0}\right)^{\varepsilon(k-K_r)} \cdot \left(\frac{\beta}{\ell_0}\right)^{(1-\varepsilon)(k-K_r)} =$$
$$\left(\frac{c_2k^3}{\varepsilon(k-K_r)\log^{1/5} k}\right)^{\varepsilon(k-K_r)} \cdot \left(\frac{c_2k^3}{(1-\varepsilon)(k-K_r)\log^{1/5} k}\right)^{(1-\varepsilon)(k-K_r)} =$$
$$\left(\frac{c_2k^3}{\varepsilon^{\varepsilon}(1-\varepsilon)^{1-\varepsilon}(k-K_r)\log^{1/5} k}\right)^{(k-K_r)} \leq \left(\frac{2c_2k^3}{(k-K_r)\log^{1/5} k}\right)^{(k-K_r)}$$

The Inclusion-Exclusion part runs in $2^{\frac{1}{2}K_r \log k} = \sqrt{k}^{K_r}$. By the same arguments as before, we can bound the total running time as desired.

◀