Near-Optimal Algorithms for Point-Line Covering Problems

Jianer Chen
Department of Computer Science and Engineering, Texas A&M University, TX, USA

Qin Huang
Department of Computer Science and Engineering, Texas A&M University, TX, USA

Iyad Kanj
School of Computing, DePaul University, Chicago, IL, USA

Ge Xia
Department of Computer Science, Lafayette College, Easton, PA, USA

Abstract

We study fundamental point-line covering problems in computational geometry, in which the input is a set S of points in the plane. The first is the Rich Lines problem, which asks for the set of all lines that each covers at least λ points from S, for a given integer parameter $\lambda \geq 2$; this problem subsumes the 3-Points-on-Line problem and the Exact Fitting problem, which – the latter – asks for a line containing the maximum number of points. The second is the NP-hard problem Line Cover, which asks for a set of k lines that cover the points of S, for a given parameter $k \in \mathbb{N}$. Both problems have been extensively studied. In particular, the Rich Lines problem is a fundamental problem whose solution serves as a building block for several algorithms in computational geometry.

For Rich Lines and Exact Fitting, we present a randomized Monte Carlo algorithm that achieves a lower running time than that of Guibas et al.'s algorithm [*Computational Geometry* 1996], for a wide range of the parameter λ. We derive lower-bound results showing that, for $\lambda = \Omega(\sqrt{n \log n})$, the upper bound on the running time of this randomized algorithm matches the lower bound that we derive on the time complexity of Rich Lines in the algebraic computation trees model.

For Line Cover, we present two kernelization algorithms: a randomized Monte Carlo algorithm and a deterministic algorithm. Both algorithms improve the running time of existing kernelization algorithms for Line Cover. We derive lower-bound results showing that the running time of the randomized algorithm we present comes close to the lower bound we derive on the time complexity of kernelization algorithms for Line Cover in the algebraic computation trees model.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases line cover, rich lines, exact fitting, kernelization, randomized algorithms, complexity lower bounds, algebraic computation trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.21

Related Version *Full Version*: `https://arxiv.org/abs/2012.02363`

1 Introduction

We study fundamental problems in computational geometry pertaining to covering a set S of n points in the plane with lines. The first problem, referred to as Rich Lines, is defined as:

> Rich Lines: Given a set S of n points and an integer parameter $\lambda \geq 2$, compute the set of lines that each covers at least λ points.

A special case of Rich Lines that has received attention is the Exact Fitting problem [25], which asks for computing a line that covers the maximum number of points in S. Exact Fitting subsumes the well-known 3-Points-on-Line problem in an obvious way.

The Rich Lines problem is a fundamental problem whose solution serves as a building block for several algorithms in computational geometry [17, 16, 21, 23, 24, 31], including algorithms for the fundamental Line Cover problem, which is our other focal problem:

> Line Cover: Given a set S of n points and a parameter $k \in \mathbb{N}$, decide if there exist at most k lines that cover all points in S.

See Figure 1 for an illustration of Rich Lines, Exact Fitting, and Line Cover.

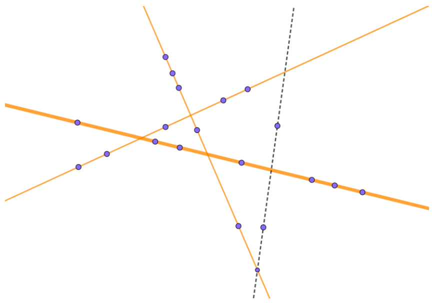

Figure 1 Illustration of an instance of Line Cover with $k = 4$, an instance of Rich Lines with $\lambda = 5$, and an instance of Exact Fitting. The set of all lines is the solution to the Line Cover instance; the set of solid orange lines is the solution to the Rich Lines instance; and the bold orange line is a solution to the Exact Fitting instance.

The Line Cover problem is NP-hard [33], and has been extensively studied in parameterized complexity [1, 7, 16, 23, 29, 31, 41], especially with respect to kernelization. Guibas et al.'s algorithm [25] for Rich Lines was used to give a simple kernelization algorithm that computes a kernel of at most k^2 points, and this upper bound on the kernel size was proved to be essentially tight by Kratsch et al. [29].

The current paper derives both upper and lower bounds on the time complexity of Rich Lines, and the time complexity of the kernelization of Line Cover. Most of the algorithmic upper-bound results we present are randomized Monte Carlo algorithms, providing guarantees on the running time of the algorithms, but may make one-sided errors with a small probability. Our work is motivated by the applications of both problems to on-line data analytics [35], where massive data processing within a guaranteed time upper bound is required (e.g., dynamic or streaming environments [3, 8, 35]). In such settings, where the data set has an enormous size, classical algorithmic techniques become infeasible, and timely pre-processing the very large input in order to reduce its size becomes essential. Therefore, we seek algorithms whose running time is nearly linear and whose space complexity is low, trading off the optimality/correctness of the algorithm with a small probability.

1.1 Related Work

Both Rich Lines and Exact Fitting were studied by Guibas et al. [25], motivated by their applications in statistical analysis (e.g., linear regressions), computer vision, pattern recognition, and computer graphics [26, 27]. Guibas et al. [25] developed an $\mathcal{O}(\min\{\frac{n^2}{\lambda}\log\frac{n}{\lambda}, n^2\})$-time deterministic algorithm for Rich Lines, and used it to solve Exact Fitting within the same time upper bound. Guibas et al.'s algorithm [25] was subsequently used in many algorithmic results [17, 16, 21, 23, 24, 31] pertaining to geometric covering problems and their applications.

The Line Cover problem has been extensively studied with respect to several computational frameworks, including approximation [7, 23] and parameterized complexity [1, 7, 16, 29, 31, 41]. The problem is known to be APX-hard [30] and is approximable within ratio $\log n$, being a special case of the set cover problem [28].

From the parameterized complexity perspective, several fixed-parameter tractable algorithms for Line Cover were developed [1, 23, 31, 41], leading to the current-best algorithm that runs in time $(c \cdot k/\log k)^k n^{\mathcal{O}(1)}$ [1], for some constant $c > 0$. Guibas et al.'s algorithm [25] was used in several works to give a kernel of size k^2 that is computable in time $\mathcal{O}(\min\{\frac{n^2}{k}\log(\frac{n}{k}), n^2\})$ [16, 29, 31]. This quadratic kernel size was shown to be essentially tight by Kratsch et al. [29], who showed that: For any $\epsilon > 0$, unless the polynomial-time hierarchy collapses to the third level, Line Cover has no kernel of size $\mathcal{O}(k^{2-\epsilon})$.

Kernelization algorithms for Line Cover have drawn attention in recent research in massive-data processing; Mnich [35] discusses how the Line Cover problem is used in such settings, where the point set represents a very large collection of observed (accurate) data, and the solution sought is a model consisting of at most k linear predictors [20].

Chitnis et al. [8] studied Line Cover in the streaming model, and Alman et al. [3] studied the problem in the dynamic model. We mention that Chitnis et al.'s streaming algorithm [8] may be used to give a Monte Carlo kernelization algorithm for Line Cover running in time $\mathcal{O}(n(\log n)^{\mathcal{O}(1)})$, and the dynamic algorithm of Alman et al. [3] may be used to give a deterministic kernelization algorithm for Line Cover running in time $\mathcal{O}(nk^2)$.

We finally note that there has been considerable work on randomized algorithms for geometric problems (see [2, 9, 10, 12], to name a few). The most relevant of which to our work is the randomized algorithm for approximating geometric set covering problems [5, 11] (see also [2]), which implies an $\mathcal{O}(\log k)$-factor approximation algorithm for the optimization version of Line Cover whose expected running time is $\mathcal{O}(nk(\log n)(\log k))$.

1.2 Results and Techniques

In this paper, we develop new tools to derive upper and lower bounds on the time complexity of Rich Lines and the kernelization time complexity of Line Cover. Our results and techniques are summarized as follows.

1.2.1 Results for Rich Lines

We present a randomized one-sided errors Monte Carlo algorithm for Rich Lines that, with probability at least $1 - \frac{3}{n^2}$, returns the correct solution set, where n is the number of points. The algorithm achieves a lower running time upper bound than Guibas et al.'s algorithm [25] for a wide range of the parameter λ, namely for $\lambda = \Omega(\log n)$, and matches its running time otherwise. For instance, when $\lambda = \Theta(\sqrt{n \log n})$, the running time of our algorithm is $\mathcal{O}(n \log n)$, whereas that of Guibas et al.'s algorithm is $\mathcal{O}(n^{3/2}\sqrt{\log n})$, yielding a $(\sqrt{n/\log n})$-factor improvement. We show that, for $\lambda = \Omega(\sqrt{n \log n})$, the upper bound of $\mathcal{O}(n \log(\frac{n}{\lambda}))$ on the running time of our randomized algorithm matches the lower bound that we derive on the time complexity of the problem in the algebraic computation trees model. The algorithm for Rich Lines implies an algorithm for Exact Fitting with the same performance guarantees – as shown by Guibas et al. [25], obtained by binary-searching for the value of λ that corresponds to the line(s) containing the maximum number of points.

The crux of the technical contributions leading to the randomized algorithm we present is a set of new tools we develop pertaining to point-line incidences and sampling. The aforementioned tools allow us to show that, by sampling a smaller subset of the original set of points, with high probability, we can reduce the problem of computing the set of λ-rich lines in the original set to that of computing the set of λ'-rich lines in the smaller subset, where λ' is a smaller parameter than λ.

The time lower-bound result we present is obtained via a 2-step reduction. The first employs Ben-Or's framework [4] to show a time lower bound of $\Omega(n \log(\frac{n}{\lambda}))$ in the algebraic computation trees model on a problem that we define, referred to as the Multiset Subset Distinctness problem. We then compose this reduction with a reduction from Multiset Subset Distinctness to Rich Lines, thus establishing the $\Omega(n \log(\frac{n}{\lambda}))$ lower-bound result for Rich Lines. We note that these reductions are very "sensitive", and hence need to be crafted carefully, as the lower-bound results apply for *every* value of n and λ.

1.2.2 Results for Line Cover

We derive a lower bound on the time complexity of kernelization algorithms for Line Cover in the algebraic computation trees model and show that the running time of any such algorithm must be at least $cn \log k$ for some constant $c > 0$; more specifically, one cannot asymptotically improve either of the two factors n or $\log k$ in this term. This result particularly rules out the possibility of a kernelization algorithm that runs in $\mathcal{O}(n)$ time (i.e., in linear time). We derive this lower bound by combining a lower-bound result by Grantson and Levcopoulos [23] on the time complexity of Line Cover with a result that we prove in this paper connecting the time complexity of Line Cover to its kernelization time complexity. In fact, it is not difficult to develop a kernelization algorithm for Line Cover that runs in time $\mathcal{O}(n \log k + g(k))$ for some computable function $g(k)$, and computes a kernel of size k^2. This can be done by processing the input in "batches" of size roughly k^2 each; this is implied by the algorithm in [23], which runs in time $\Omega(n \log k + k^4 \log k)$, and approximates the optimization version of Line Cover. Therefore, we focus on developing kernelization algorithms where the function $g(k)$ in their running time is as small as possible. Since we can assume that $n \geq k^2$ (otherwise, the instance is already kernelized), we may assume that $g(k) = \Omega(k^2 \log k)$. Therefore, we endeavor to develop a kernelization algorithm for which the function $g(k)$ – in its running time – is as close as possible to $\mathcal{O}(k^2 \log k)$, and hence, a kernelization algorithm whose running time is as close as possible to $\mathcal{O}(n \log k)$. In addition, reducing the function $g(k)$ serves well our purpose of obtaining near-linear-time kernelization algorithms for Line Cover due to their potential applications [20, 35].

We present two kernelization algorithms for Line Cover. The first is a randomized one-sided errors Monte Carlo algorithm that runs in time $\mathcal{O}(n \log k + k^2 (\log^2 k)(\log\log k)^2)$ and space $\mathcal{O}(k^2 \log^2 k)$ and, with probability at least $1 - \frac{2}{k^3}$, computes a kernel of size at most k^2. The second is a deterministic algorithm that computes a kernel of size at most k^2 in time $\mathcal{O}(n \log k + k^3 (\log^3 k)(\sqrt{\log\log k}))$. Both algorithms improve the running time of existing kernelization algorithms for Line Cover [3, 8, 16, 23, 29, 31]. Moreover, the running time of the randomized algorithm comes very close to the derived lower bound on the time complexity of kernelization algorithms for Line Cover, as it only differs from it by a factor of $\log k(\log\log k)^2$ in the term that depends only on k.

The key tool leading to the improved kernelization algorithms is partitioning the "saturation range" of the saturated lines (i.e., the lines that each contains at least $k + 1$ points and must be in the solution) in the batch of points under consideration into intervals, thus defining a spectrum of saturation levels. Then the algorithm for Rich Lines (either the randomized or Guibas et al.'s algorithm [25]) is invoked starting with the highest saturation threshold, and iteratively decreasing the threshold until either: the saturated lines computed cover "enough" points of the batch under consideration, or the total number of saturated lines computed is "large enough" thus making "enough progress" towards computing the line cover. This scheme enables us to amortize the running time of the algorithm that computes the saturated lines, creating a win/win situation and improving the overall running time.

2 Preliminaries

We assume familiarity with basic geometry, probability, and parameterized complexity and refer to the following standard textbooks on some of these subjects [13, 14, 18, 34, 36]. For a positive integer i, we write $[i]$ for $\{1, 2, \dots, i\}$. We write "*w.h.p.*" as an abbreviation for "with high probability", and we write "*u.a.r.*" as an abbreviation for "uniformly at random".

Probability. The *union bound* states that, for any probabilistic events $E_1, E_2, \dots, E_j$, we have: $\Pr\left(\bigcup_{i=1}^{j} E_i\right) \le \sum_{i=1}^{j} \Pr(E_i)$. For any discrete random variables $X_1, \dots, X_n$ with finite expectations, it is well known that: $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$, where $E[X]$ denotes the expectation of X.

The following lemma, for the sum of a random sample without replacement from a finite set, can be viewed as an application of Chernoff's bounds – customized to our needs – to negatively correlated random variables:

▶ **Lemma 1.** *Let $\mathcal{C} = \{x_1, \dots, x_N\}$, where $x_i \in \{0, 1\}$ for $i \in [N]$. Let $X_1, X_2, \dots, X_j$ denote a random sample without replacement from $\mathcal{C}$. Let $X = \sum_{i=1}^{j} X_i$, $\mu = E[X]$, and μ_1, μ_2 be any two values such that $\mu_1 \le \mu \le \mu_2$. Then, (A) for any $\delta > 0$, we have $\Pr(X \ge (1+\delta)\mu_2) \le \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu_2}$; and (B) for any $0 < \delta < 1$, we have $\Pr(X \le (1-\delta)\mu_1) \le \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu_1}$.*

Point-Line Incidences. Let S be a set of points. A line l *covers* a point $p \in S$ if l passes through p (i.e., contains p). A set L of lines *covers* S if every point in S is covered by at least one line in L. A line l is *induced* by S if l covers at least 2 points of S, and a set L of lines is *induced* by S if every line in L is induced by S. For a set L of lines, we define $I(L, S)$ as $I(L, S) = |\{(q, l) \mid q \in S \cap l, l \in L\}|$; that is, $I(L, S)$ is the number of incidences between L and S. For a line l, let $I(l, S) = |\{(q, l) \mid q \in S \cap l\}|$.

The following theorems upper bound $I(L, S)$ and the complexity of computing it:

▶ **Theorem 2** ([37]). *$I(L, S) \le \frac{5}{2}(mn)^{2/3} + m + n$, where $n = |S|$ and $m = |L|$.*

The theorem below follows from Theorem 3.1 in [32] after a slight modification, as was also observed by [15]:

▶ **Theorem 3** ([15, 32]). *Let S be a set of n points and L a set of m lines in the plane. The set of incidences between S and L, and hence $I(L, S)$, can be computed in (deterministic) time $\mathcal{O}(n \log m + m \log n + (mn)^{2/3} 2^{\mathcal{O}(\log^*(n+m))})$. Moreover, within the same running time, we can compute for each line $l \in L$ the set of points in S that are contained in l.*

Let P be a subset of S, and let $x \in \mathbb{N}$. We say that a line l is *x-rich* for P if l covers at least x points from P; when P is clear from the context, we will simply say that l is x-rich.

The following extends Theorem 2 in [40], which applies only when the constant $c < 1$:

▶ **Theorem 4.** *Let S be a set of n points, let $c > 0$ be a constant, and let k be an integer such that $2 \le k \le c\sqrt{n}$. Let L be the set of k-rich lines for S. Then $|L| < \max\{\frac{40n^2}{k^3}, \frac{40c^2n^2}{k^3}\}$.*

Parameterized Complexity. A *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed alphabet. Each instance of Q is a pair (I, κ), where $\kappa \in \mathbb{N}$ is called the *parameter*. A parameterized problem Q is *fixed-parameter tractable* (FPT), if there is an algorithm, called an FPT-*algorithm*, that decides whether an input (I, κ) is a member of Q in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function and $|I|$ is the input instance size. The class FPT denotes the class of all fixed-parameter tractable parameterized problems. A parameterized problem is *kernelizable* if there exists a polynomial-time reduction that maps an instance (I, κ) of the problem to another instance (I', κ') such that (1) $|I'| \leq f(\kappa)$ and $\kappa' \leq f(\kappa)$, where f is a computable function, and (2) (I, κ) is a yes-instance of the problem if and only if (I', κ') is. The instance (I', κ') is called the *kernel* of I.

3 A Randomized Algorithm for Rich Lines

In this section, we present a randomized algorithm for Rich Lines that achieves a better running time than Guibas et al.'s algorithm [25] for a wide range of the parameter λ (in the problem definition). We will show in Section 5 that for $\lambda = \Omega(\sqrt{n \log n})$, the upper bound of $\mathcal{O}(n \log(\frac{n}{\lambda}))$ on the running time of our randomized algorithm for Rich Lines matches the lower bound on its time complexity that we derive in the algebraic computation trees model.

We first present an intuitive low-rigor explanation of the randomized algorithm and the techniques entailed. The crux of the technical results in this section lies in Lemma 6. This lemma shows that, by sampling a smaller subset S' of S whose size depends on λ, w.h.p. we can reduce the problem of computing the set of λ-rich lines for S to that of computing the set of λ'-rich lines for S', where $\lambda' < \lambda$.

The algorithm exploits the above technical results as follows. Given an instance (S, λ) of Rich Lines, the algorithm samples a subset $S' \subseteq S$ whose size depends on λ. Depending on the value of λ, the algorithm defines a threshold value λ', and computes the set L' of λ'-rich lines for S'. As we show in this section, w.h.p. L' contains all the λ-rich lines for S, and hence, by sifting through the lines in L', the algorithm computes the solution to (S, λ).

▶ **Lemma 5.** *Let $\lambda \geq 2\sqrt{n}$. The number of λ-rich lines for S is at most $\frac{2n}{\lambda}$.*

Proof. Let $L = \{l_1, l_2, ..., l_m\}$ be the set of λ-rich lines for P. Denote by z_i, for $i = 1, 2, \ldots, m$, the number of points covered by l_i. The set L covers at least $\sum_{i=1}^{m} z'_i$ points, where $z'_i = \max\{z_i - i + 1, 0\}$. This is true since l_1 covers at least z_1 points, l_2 covers at least $z_2 - 1$ new points (excluding at most 1 point on l_1), and in general, l_i covers at least $z_i - i + 1$ new points (excluding at most $i - 1$ points covered by $\{l_1, ..., l_{i-1}\}$) if $i \leq z_i$ or 0 otherwise. Suppose, to get a contradiction, that $m > \frac{2n}{\lambda}$, and consider the first $m' = \lceil \frac{2n}{\lambda} \rceil$ lines. Then $\sum_{i=1}^{m} z'_i \geq \sum_{i=1}^{m'} z'_i = \sum_{i=1}^{m'} (z_i - i + 1)$ since $i \leq m' < z_i$. Hence, $\sum_{i=1}^{m'} z'_i \geq \lambda \cdot m' - \frac{(m'-1)m'}{2}$ $\geq 2n - \frac{(\frac{2n}{\lambda}+1)\frac{2n}{\lambda}}{2} = 2n - \frac{n}{\lambda} - \frac{2n^2}{\lambda^2}$. Since $\lambda \geq 2\sqrt{n}$, $\sum_{i=1}^{m'} z'_i \geq 2n - \frac{\sqrt{n}}{2} - \frac{n}{2} > n$ (assuming w.l.o.g. that $n > 1$), which is a contradiction. Therefore, we have $m \leq \frac{2n}{\lambda}$. ◀

Throughout this section, S denotes a set of $n \geq 3$ points. Let $S'(m)$ be a set formed by sampling without replacement $m \leq n$ points from S uniformly and independently at random.

▶ **Lemma 6.** *Let λ be an integer satisfying $140 \ln^{3/2} n \leq \lambda \leq n$. Let $S'(m)$ be as defined above where $m = \lceil \frac{140 n \ln n}{\lambda} \rceil$. Let L_1 be the set of λ-rich lines for S, and let L_3 be the set of $(98 \ln n)$-rich lines for $S'(m)$. Then, with probability at least $1 - \frac{2}{n^2}$, we have: (1) $L_1 \subseteq L_3$, and (2) if $\lambda \geq 5\sqrt{n}$, then $|L_3| \leq \frac{5n}{\lambda}$; if $\lambda < 5\sqrt{n}$, then $|L_3| < \frac{2500 n^2}{\lambda^3}$.*

Proof. Let L_2 be the set of lines induced by S containing less than $\frac{2\lambda}{5}$ points each. Without loss of generality, suppose that $L_1 = \{l_1, \ldots, l_d\}$ and $L_2 = \{l_{d+1}, \ldots, l_{d'}\}$, and note that $d \leq d' \leq \binom{n}{2}$. For $i \in [d']$, let x_i be the number of points in S covered by l_i, and let X'_i be the random variable, where X'_i is the number of the points in $S'(m)$ on l_i.

For $i \in [d]$, we have $E[X'_i] = \frac{x_i}{n} \cdot m \geq 140 \ln n$ since $x_i \geq \lambda$. Applying part (B) of the Chernoff bounds in Lemma 1 with $\mu_1 = 140 \ln n$, we have $\Pr(X'_i \leq (1 - 3/10) \cdot 140 \ln n) \leq \left(\frac{e^{-3/10}}{(1-3/10)^{1-3/10}}\right)^{140 \ln n} \leq \frac{1}{n^4}$, where the last inequality can be easily verified by a simple analysis. Let E_i, for $i \in [d]$, denote the event that $X'_i \leq (1 - 3/10) \cdot 140 \ln n$. Applying the union bound, we have $\Pr(\bigcap_{i=1}^{d} \overline{E_i}) = 1 - \Pr(\bigcup_{i=1}^{d} E_i) \geq 1 - d \cdot \frac{1}{n^4} \geq 1 - \frac{1}{n^2}$. Let $\mathcal{E} = \bigcap_{i=1}^{d} \overline{E_i}$. The probability that every line $l_i \in L_1$ contains at least $98 \ln n$ points of $S'(m)$ is at least $1 - \frac{1}{n^2}$. That is to say, with probability at least $1 - \frac{1}{n^2}$, we have $L_1 \subseteq L_3$.

For $i = d+1, \ldots, d'$, $E[X'_i] = \frac{x_i}{n} \cdot m < \frac{x_i}{n}(\frac{140 n \ln n}{\lambda} + 1) \leq 56 \ln n + \frac{2\lambda}{5n} < 57 \ln n$, since $x_i \leq \frac{2\lambda}{5}$ and $140 \ln^{3/2} n \leq \lambda \leq n$. Applying part (A) of the Chernoff bounds in Lemma 1 with $\mu_2 = 57 \ln n$, we get $\Pr(X'_i \geq (1 + \frac{13}{19}) \cdot 57 \ln n) \leq \left(\frac{e^{13/19}}{(1+13/19)^{1+13/19}}\right)^{57 \ln n} \leq \frac{1}{n^4}$, where the last inequality can be easily verified by a simple analysis. Consequently, via the union bound, the probability that every line $l_i \in L_2$ contains less than $96 \ln n$ sampled points is at least $1 - (d' - d) \cdot \frac{1}{n^4} \geq 1 - \frac{1}{n^2}$. It follows that, with probability at least $1 - \frac{1}{n^2}$, we have $L_3 \cap L_2 = \emptyset$.

Altogether, with probability at least $1 - \frac{2}{n^2}$, $L_1 \subseteq L_3$ and $L_2 \cap L_3 = \emptyset$. Recall that each line in L_3 covers at least $\frac{2\lambda}{5}$ points of S. If $\frac{2\lambda}{5} \geq 2\sqrt{n}$, i.e., $\lambda \geq 5\sqrt{n}$, we have $|L_3| \leq \frac{2n}{2\lambda/5} = \frac{5n}{\lambda}$ by Lemma 5. If $\frac{2\lambda}{5} < 2\sqrt{n}$, i.e. $\lambda < 5\sqrt{n}$, by Theorem 4 (with $c \leq 2$), we have $|L_3| < \frac{40 \cdot 2^2 n^2}{(2\lambda/5)^3} = \frac{2500 n^2}{\lambda^3}$. It follows that, with probability at least $1 - \frac{2}{n^2}$, parts (1) and (2) of the lemma hold. ◀

Algorithm 1 : Alg-RichLines(S, λ) – A randomized algorithm for computing all λ-rich lines.

Input: a set of points S and $\lambda \in \mathbb{N}$.
Output: The set L of λ-rich lines for S.

1: **if** $\lambda < \ln n$ **then** apply Guibas et al.'s algorithm [25] to compute L and return L;
2: sample $x = \lceil \frac{10 n^2 \ln n}{\lambda^2} \rceil$ pairs of points $(p_1, q_1), \ldots, (p_x, q_x)$ u.a.r. from $\binom{S}{2}$;
3: let l_i be the line formed by (p_i, q_i), for $i \in [x]$; let Q_1 be the multi-set $\{l_1, l_2, \ldots, l_x\}$, and let Q_2 be the set of distinct lines in Q_1;
4: **if** $\lambda \leq 140 \ln^{3/2} n$ **then** let $L = \{l \in Q_2 \mid I(l, S) \geq \lambda\}$; return L;
5: let $m = \lceil \frac{140 n \ln n}{\lambda} \rceil$, $y = 98 \ln n$;
6: sample m points u.a.r. from S without replacement to obtain $S'(m)$;
7: **if** $\lambda < 5\sqrt{n}$ **then**
8: let $z = \frac{2500 n^2}{\lambda^3}$;
9: **else** let $z = \frac{5n}{\lambda}$;
10: let $L' = \{l \in Q_2 \mid I(l, S'(m)) \geq y\}$;
11: **if** $|L'| \leq z$ **then** let $L = \{l \in L' | I(l, S) \geq \lambda\}$; return L;
12: **else** return $\emptyset$;

Refer to **Alg-RichLines** for the terminologies used in the subsequent discussions.

▶ **Lemma 7.** *Let $\lambda \in \mathbb{N}$. Let L_1 be the set of λ-rich for S. Then, with probability at least $1 - \frac{3}{n^2}$,* **Alg-RichLines**(S, λ) *returns a set $L = L_1$.*

Proof. If $\lambda < \ln n$ then $L = L_1$ with probability 1 by Step 1, as Guibas et al.'s algorithm [25] computes L_1 deterministically.

Now consider the case that $\lambda \geq \ln n$. Let l be an arbitrary line in L_1. Step 2 samples x pairs of points that determine x lines. In a single sampling, the probability ρ that l is sampled is $\binom{\lambda}{2}/\binom{n}{2} \geq \frac{\lambda^2}{2n^2}$ because l covers at least λ points of S. Thus, $\Pr(l \notin Q_1) = (1-\rho)^x \leq e^{-\rho x} \leq \frac{1}{n^5}$. Since $|L_1| < n^2$, applying the union bound, we get $Pr(L_1 \subseteq Q_1) \geq 1 - \frac{1}{n^3}$. Hence, we have $\Pr(L_1 \subseteq Q_2) \geq 1 - \frac{1}{n^3}$ since Q_2 is obtained from Q_1 by removing repeated lines. Let L_3 be the set of y-rich lines for $S'(m)$. If $\lambda < 140 \ln^{3/2} n$, then since $L_1 \subseteq Q_2$ with probability at least $1 - \frac{1}{n^3}$, the algorithm returns in Step 4 a set L that, with probability at least $1 - \frac{1}{n^3}$, is equal to L_1 .

Finally, if $140 \ln^{3/2} n \leq \lambda$, then by Lemma 6, $L_1 \subseteq L_3$ and $|L_3| \leq z$ with probability at least $1 - \frac{2}{n^2}$. Since $L' = L_3 \cap Q_2$, $|L'| \leq z$ holds with probability at least $1 - \frac{2}{n^2}$. Since $\Pr(L_1 \subseteq Q_2) \geq 1 - \frac{1}{n^3}$, we have $L_1 \subseteq (L_3 \cap Q_2)$ and $|L'| \leq z$ with probability at least $1 - \frac{3}{n^2}$. Thus, $\Pr(L_1 \subseteq L') \geq 1 - \frac{3}{n^2}$. Therefore, the algorithm returns in Step 11 a set L equal to L_1 with probability at least $1 - \frac{3}{n^2}$. ◀

The correctness of **Alg-RichLines**, stated in the following theorem, follows from Lemma 7.

▶ **Theorem 8.** *Let S be a set of n points and $\lambda \in \mathbb{N}$. With probability at least $1 - \frac{3}{n^2}$,* **Alg-RichLines**(S, λ) *solves the* Rich Lines *problem in time:*

(1) $\mathcal{O}(n^2)$ *if $\lambda < \ln n$; and*
(2) $\mathcal{O}(n \log \frac{n}{\lambda} + \frac{n^2 \log n \log \frac{n}{\lambda}}{\lambda^2})$ *otherwise.*

Proof. By Lemma 7, with probability at least $1 - \frac{3}{n^2}$, **Alg-RichLines**(P, λ) correctly returns the set of λ-rich lines for P. We discuss next the running time of the algorithm.

Case 1: $\lambda < \ln n$. In this case the running time of the algorithm is that of Guibas et al.'s algorithm [25], which is $\mathcal{O}(n^2)$.

It is easy to see that Step 2 takes $\mathcal{O}(x) = \mathcal{O}(\frac{n^2 \ln n}{\lambda^2})$ time. Step 3 can be implemented by sorting the slopes of the x lines, which takes $\mathcal{O}(x \ln x) = O(\frac{n^2 \ln n}{\lambda^2} \cdot (\ln \frac{n}{\lambda} + \ln \ln n))$ time. Step 6 takes time $\mathcal{O}(m) = \mathcal{O}(\frac{140 n \ln n}{\lambda}) = \mathcal{O}(n)$ since $n \geq \lambda \geq 140 \ln^{3/2} n$. Steps 5, 7, 8, 9 and 12 take constant time. Note that all the above running times (for Steps 2, 3, 5, 6, 7, 8, 9, 12) are dominated by the running time listed in item (2) of the theorem.

We discuss the running time of Step 4 in Case 2 below, and that of Step 10 and Step 11 in both Cases 3 and 4. Note that, to determine the set of rich lines for S in Steps 4, 10 and 11, we apply Theorem 3 to compute the number of points in S (or $S'(m)$) on each of the lines in question, thus determining the set of rich lines for S (or $S'(m)$).

Case 2: $\ln n \leq \lambda \leq 140 \ln^{3/2} n$. In this case, Step 4 takes time $T_1 = \mathcal{O}(x \log n + n \log x + (nx)^{2/3} 2^{\mathcal{O}(\log^*(x+n))})$ by Theorem 3. Substituting $x = \lceil \frac{10 n^2 \ln n}{\lambda^2} \rceil$, we obtain $T_1 = \mathcal{O}(\frac{n^2 \ln^2 n}{\lambda^2} + n \ln n + \frac{n^2 \ln^{2/3} n}{\lambda^{4/3}} 2^{\mathcal{O}(\log^* n)}) = \mathcal{O}(\frac{n^2 \ln^2 n}{\lambda^2})$, which is dominated by the running time listed in item (2) of the theorem.

We discuss the running time of Step 10 in both Case 3 and 4. Note that $|S'(m)| = m = \lceil \frac{140 n \ln n}{\lambda} \rceil$ and $|Q_2| \leq x = \mathcal{O}(\frac{n^2 \ln n}{\lambda^2})$. By Theorem 3, Step 10 takes time:

$$\begin{aligned} T_2 &= \mathcal{O}(m \log x + x \log m + (xm)^{2/3} 2^{\mathcal{O}(\log^*(x+m))}) \\ &= \mathcal{O}(\frac{n^2 \ln n}{\lambda^2}(\ln(\frac{n}{\lambda}) + \ln \ln n) + \frac{n^2}{\lambda^2} \ln^{4/3}(n) 2^{\mathcal{O}(\log^* n)}). \end{aligned}$$

If $\lambda \leq n^{2/3}$, we have $\frac{n^2}{\lambda^2} \ln^{4/3}(n) 2^{\mathcal{O}(\log^* n)} = \mathcal{O}(\frac{n^2 \log n \log \frac{n}{\lambda}}{\lambda^2})$, and if $n^{2/3} < \lambda \leq n$, we have $\frac{n^2}{\lambda^2} \ln^{4/3}(n) 2^{\mathcal{O}(\log^* n)} = \mathcal{O}(n \log \frac{n}{\lambda})$. Altogether, $\frac{n^2}{\lambda^2} \ln^{4/3}(n) 2^{\mathcal{O}(\log^* n)} = \mathcal{O}(n \log \frac{n}{\lambda} + \frac{n^2 \log n \log \frac{n}{\lambda}}{\lambda^2})$. Therefore, $T_2 = \mathcal{O}(n \log \frac{n}{\lambda} + \frac{n^2 \log n \log \frac{n}{\lambda}}{\lambda^2})$, which is dominated by the running time listed in item (2) of the theorem.

Case 3: $140\ln^{3/2} n < \lambda < 5\sqrt{n}$. Step 11 takes time $T_3 = \mathcal{O}(n\log|L'| + |L'|\log n + (n|L'|)^{2/3}2^{\mathcal{O}(\log^*(n+|L'|))})$ by Theorem 3. Since $|L'| \le z = \frac{2500n^2}{\lambda^3}$, we have $T_3 = \mathcal{O}(n\ln n + \frac{n^2\log n}{\lambda^3} + \frac{n^2}{\lambda^2}2^{\mathcal{O}(\log^* n)}) = \mathcal{O}(n\log\frac{n}{\lambda} + \frac{n^2\log n\log\frac{n}{\lambda}}{\lambda^2})$. Thus, the total running time in this case is $\mathcal{O}(n\log\frac{n}{\lambda} + \frac{n^2\log n\log\frac{n}{\lambda}}{\lambda^2})$.

Case 4: $\lambda \ge 5\sqrt{n}$. Step 11 takes time $T_4 = \mathcal{O}(n\log|L'| + |L'|\log n + (n|L'|)^{2/3}\cdot 2^{\mathcal{O}(\log^*(n+|L'|))})$ by Theorem 3. Since $|L'| \le z = \mathcal{O}(\frac{n}{\lambda})$, $T_4 = \mathcal{O}(n\log\frac{n}{\lambda} + \frac{n}{\lambda}\log n + \frac{n^{4/3}}{\lambda^{2/3}}2^{\mathcal{O}(\log^* n)}) = \mathcal{O}(n\log\frac{n}{\lambda} + \frac{n^{4/3}}{\lambda^{2/3}}2^{\mathcal{O}(\log^* n)}) = \mathcal{O}(n\log\frac{n}{\lambda})$. The last equality holds because $\lambda \ge 5\sqrt{n}$. Consequently, the total running time in this case is $\mathcal{O}(n\log\frac{n}{\lambda} + \frac{n^2\log n\log\frac{n}{\lambda}}{\lambda^2})$. ◀

Guibas et al.'s algorithm [25] solves the Rich Lines and the Exact Fitting problems in the plane in time $\mathcal{O}(\min\{\frac{n^2}{\lambda}\log\frac{n}{\lambda}, n^2\})$. Theorem 8 is an improvement over Guibas et al.'s algorithm [25] for both problems for all values of $\lambda \ge \ln n$, and for $\lambda < \ln n$ it obviously has a matching running time. In particular, for $\ln n \le \lambda \le 140\ln^{3/2} n$, the improvement could be in the order of $\frac{1}{\sqrt{\log n}}$ (i.e., the running time of **Alg-RichLines** is a $\frac{1}{\sqrt{\log n}}$-fraction of that in [25]); for $140\ln^{3/2} n < \lambda < 5\sqrt{n}$, the improvement could be in the order of $\frac{\log n}{\sqrt{n}}$; and for $\lambda \ge 5\sqrt{n}$, the improvement could be in the order of $\sqrt{\frac{\log n}{n}}$.

4 Kernelization Algorithms for Line Cover

In this section, we present a randomized Monte Carlo kernelization algorithm for Line Cover that employs **Alg-RichLines** developed in the previous section. We also show how the tools developed in this section can be used to obtain a deterministic kernelization algorithm for Line Cover that employs Guibas et al.'s algorithm [25]. Both algorithms improve the running time of existing kernelization algorithms for Line Cover. Moreover, we will show in Section 5 that the running time of our randomized algorithm comes close to the lower bound that we derive on the time complexity of kernelization algorithms for Line Cover in the algebraic computation trees model. The majority of this section is dedicated to proving the following theorem:

▶ **Theorem 9.** *There is a Monte Carlo randomized algorithm,* **Alg-Kernel***, that given an instance (S, k) of* Line Cover*, in time $\mathcal{O}(n\log k + k^2(\log^2 k)(\log\log k)^2)$, returns an instance (S', k') such that $|S'| \le k^2$, and such that with probability at least $1 - \frac{2}{k^3}$, (S', k') is a kernel of (S, k). More specifically: (1) if (S, k) is a yes-instance of* Line Cover*, then with probability at least $1 - \frac{2}{k^3}$, (S', k') is a yes-instance of* Line Cover*; and (2) if (S, k) is a no-instance of* Line Cover *then (S', k') is a no-instance of* Line Cover*. The space complexity of this algorithm is $\mathcal{O}(k^2\log^2 k)$.*

Let (S, k) be an instance of Line Cover. We say that a line l is *saturated* w.r.t. S if it is $(k+1)$-rich for S; denote by the *saturation* of a saturated line l the number of points on l. A line l is *unsaturated* w.r.t. S if it is not saturated. We start by giving an intuitive explanation of the results leading to the kernelization algorithm **Alg-Kernel**.

The kernelization algorithm processes the set S of points in "batches" of roughly $2k^2$ uncovered points each, and for each batch S', computes the saturated lines induced by S' and adds them to the (partial) solution. Since processing each batch should result in computing at least one saturated line – assuming a yes-instance, the above process iterates at most k times. The main task becomes to compute the saturated lines induced by a batch efficiently. One straightforward idea is to invoke **Alg-RichLines** directly with $\lambda = k + 1$, which, w.h.p., computes all the saturated lines in S'. The drawback is that **Alg-RichLines** takes time $\mathcal{O}(k^2 \log^2 k)$ per batch, and may result in a single saturated line, and hence in an overall running time of $\mathcal{O}(n \log k + k^3 \log^2 k)$ for the kernelization algorithm.

The main technical contributions of this section lie in devising a more efficient implementation of the above kernelization scheme. The improved scheme rests on two key observations: (1) the running time of **Alg-RichLines** decreases as the saturation threshold (i.e., λ) of the saturated lines sought increases; and (2) assuming that a subset of the batch S' needs to be covered only by saturated lines, then for any $\lambda < \lambda'$, it requires more (saturated) lines of saturation λ to cover S' than lines of saturation λ'.

Based on the above observations, we design an algorithm **Alg-SaturatedLines** that intuitively works as follows. We first partition the saturation range into intervals, thus defining a spectrum of saturation levels. Then **Alg-SaturatedLines** calls **Alg-RichLines** starting with the highest saturation threshold (i.e., starting with a value of λ defining the highest saturation interval in the spectrum), and iteratively decreasing the saturation threshold until either: (1) the saturated lines computed cover "enough" points of the batch S', or (2) the total number of saturated lines computed for the batch S' is "large enough", thus making enough progress towards computing the k lines in the line cover of (S, k).

The above scheme enables us to amortize the running time of **Alg-SaturatedLines** over the number of saturated lines it computes. The main kernelization algorithm, **Alg-Kernel**, then calls **Alg-SaturatedLines** on each batch of $2k^2$ uncovered points. As we show in the analysis, the above scheme enables a win/win situation, yielding an overall running time of $\mathcal{O}(n \log k + k^2 (\log^2 k)(\log \log k)^2)$.

We also show that, if instead of using the randomized Monte Carlo algorithm **Alg-RichLines** to compute the saturated lines we use the deterministic algorithm of Guibas et al. [25], the above scheme yields a deterministic kernelization algorithm for LINE COVER that runs in time $\mathcal{O}(n \log k + k^3 (\log^3 k)\sqrt{\log \log k})$ and computes a kernel of size at most k^2.

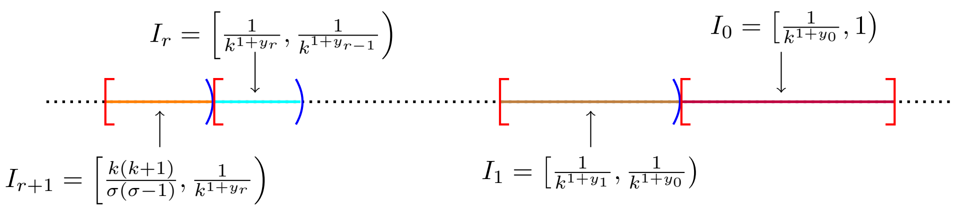

Figure 2 Illustration for the definition of the intervals $I_0, \ldots, I_{r+1}$.

We now give an intuitive low-rigor description of the technical results leading to the kernelization algorithm. Lemma 10 is a combinatorial result showing that either the saturated lines belonging to the highest interval in the saturation spectrum cover enough points of the batch S', or there is a saturation interval in the spectrum containing a "large enough" number of lines. Lemma 10 is then employed by Lemma 11 to show that, w.h.p., **Alg-SaturatedLines** returns a set of saturated lines that either covers enough points of the batch S', or contains a "large enough" number of saturated lines. We employ Lemma 10 and use amortized analysis to upper bound the running time of **Alg-SaturatedLines** w.r.t. the number of saturated lines computed by this algorithm, which we subsequently use to upper bound the running time of **Alg-SaturatedLines** in Lemma 13. Finally, Theorem 9 employs the above results to prove the correctness of **Alg-Kernel** and upper bounds its time and space complexity. We now proceed to the details.

In what follows let $\sigma = 2k^2$, and let $S' \subseteq S$ be a subset of points such that $|S'| = \sigma$. We want to identify a subset of saturated lines w.r.t. S'. We define the following notations. Let $\epsilon = \frac{\ln\ln\ln k}{\ln k}$. For $i \in \mathbb{N}$, let $y_i = 1 - \frac{\ln\ln k}{\ln k} - \frac{\ln\ln\ln k}{\ln k} + i\epsilon$. Let r be the minimum integer such that $k^{y_r} \geq \frac{k}{(\ln\ln k)^2}$, and note that $r = \mathcal{O}(\ln\ln k)$. Note that we have $y_0 < y_1 < \cdots < y_r$.

We define a sequence of intervals $I_0, \ldots, I_{r+1}$ as follows: $I_0 = [\frac{1}{k^{1+y_0}}, 1] = [\frac{\ln k(\ln\ln k)}{k^2}, 1]$, $I_i = [\frac{1}{k^{1+y_i}}, \frac{1}{k^{1+y_{i-1}}})$, for $i = 1, 2, \ldots, r$, and $I_{r+1} = [\frac{(k+1)k}{\sigma(\sigma-1)}, \frac{1}{k^{1+y_r}})$. Observe that the intervals $I_0, \ldots, I_{r+1}$ are mutually disjoint, and partition the interval $[\frac{(k+1)k}{\sigma(\sigma-1)}, 1]$. It is easy to verify that the lengths of the intervals $I_1, \ldots, I_r$ are decreasing. See Figure 2 for illustration.

Suppose that there are h saturated lines $l_1, \ldots, l_h$ w.r.t. S'. Denote by s_i the number of points in S' covered by l_i, for $i \in [h]$. Let $\rho_i = \frac{s_i(s_i-1)}{\sigma(\sigma-1)}$, and note that ρ_i belongs to one of the intervals $I_0, \ldots, I_{r+1}$. We partition the h saturated lines into at most $r + 2$ groups, $H'_0, \ldots, H'_{r+1}$, where H'_i, for $i = 0, \ldots, r+1$, consists of every saturated line l_j, $j \in [h]$, such that $\rho_j \in I_i$. Clearly, it follows that $H'_0, \ldots, H'_{r+1}$ is indeed a partitioning of $\{l_1, \ldots, l_h\}$.

Consider **Alg-SaturatedLines** for computing the saturated lines w.r.t. S':

Algorithm 2 : **Alg-SaturatedLines**(S', k, r) – A randomized algorithm for computing saturated lines w.r.t. S'.

Input: S', where $|S'| = \sigma = 2k^2$; $k \in \mathbb{N}$; and integer r as defined before
Output: A set of points S'' and a set of saturated lines L'

for $(i = 0; i \leq r+1; i++)$ **do**
  if $i \leq r$ **then** let $L' =$ **Alg-RichLines**$(S', \sigma k^{-(1+y_i)/2})$;
  else let $L' =$ **Alg-RichLines**$(S', k+1)$;
  compute the set $S'' \subseteq S'$ not covered by L';
  if $i = 0$ and L' covers at least $k^2/3$ points **then** return (S'', L');
  else if $i \leq r$ and $|L'| \geq \frac{1}{12r}k^{(1+y_{i-1})/2}$ **then** return (S'', L');
  else if $i = r+1$ and $|L'| \geq \frac{1}{12}k^{(1+y_r)/2}$ **then** return (S'', L');
return $(S', \emptyset)$;

Now we are ready to present the kernelization algorithm, **Alg-Kernel**, for Line Cover.

The kernelization algorithm works by computing w.h.p. the set H of saturated lines in S and removing all points covered by these lines. Observe that, any set of more than k^2 points that can be covered by at most k lines must contain at least one saturated line. During the execution of the algorithm, the set S', which will eventually contain the kernel, contains a subset of points in S. We start by initializing S' to the empty set, and order the points in S arbitrarily. We repeatedly add the next point in S (w.r.t. the defined order) to S' until either $|S'| = 2k^2$, or no points are left in S. Afterwards, the algorithm distinguishes two cases.

If $|S'| = 2k^2$, the algorithm calls **Alg-SaturatedLines** to compute a subset of the saturated lines w.r.t. S'. **Alg-SaturatedLines** may not compute all the saturated lines in S', and rather acts as a "filtering algorithm". This algorithm either computes a subset of saturated lines that cover at least $k^2/3$ many points in S' "efficiently", that is more efficiently than **Alg-RichLines**, which w.h.p. computes all the saturated lines in S'; or computes a "large" set of saturated lines (a little bit less efficiently than **Alg-RichLines**), thus decreasing the parameter k significantly (and hence the overall execution of the algorithm).

If $k^2 < |S'| < 2k^2$, no more points are left in S to consider. **Alg-RichLines** is called at most once to compute w.h.p. all the remaining saturated lines w.r.t. S' to return the kernel.

We now proceed to prove the correctness and analyzing the complexity of **Alg-Kernel**.

▶ **Lemma 10.** *Given a set S' of σ points and a parameter k, if S' can be covered by at most k lines then one of the following conditions must hold:*

(1) H'_0 *covers at least* $\frac{\sigma-k^2}{3}$ *points;*

(2) $|H'_i| \geq (\frac{\sigma-k^2}{6r\sigma}) \cdot k^{(1+y_{i-1})/2}$ *for some* $i \in [r]$*; or*

(3) $|H'_{r+1}| \geq (\frac{\sigma-k^2}{6\sigma}) \cdot k^{(1+y_r)/2}$.

Algorithm 3 : Alg-Kernel(S,k)–A randomized kernelization algorithm for Line Cover.

Input: $S = \{q_1, \ldots, q_n\}$; $k \in \mathbb{N}$.
Output: an instance (S', k') of Line Cover.

```
if k ≤ 15 then return the instance (S', k') described in Lemma 12;
H = ∅; S' = ∅; i = 1;
construct a search structure Γ_H for the lines in H and set Γ_H = ∅;
while |H| ≤ k do
    while |S'| < 2k^2 and i ≤ n do
        if q_i is not covered by H then add q_i to S' and set i = i + 1;
    if |S'| = 2k^2 then
        let (S', L') = Alg-SaturatedLines(S', k, r);
        if L' = ∅ then return a (trivial) no-instance (S', k');
        H = H ∪ L'; update Γ_H for H;
    else
        if |S'| > k^2 then
            L' = Alg-RichLines(S', k + 1); H = H ∪ L';
            update Γ_H for H; remove the points from S' covered by L';
            if |H| > k or |S'| > k^2 then return a (trivial) no-instance (S', k');
        return (S', k − |H|);
return a (trivial) no-instance (S', k');
```

▶ **Lemma 11.** *Given a set S' of points and a parameter $k \geq 16$, let L' be the set of lines returned by* **Alg-SaturatedLines**(S', k, r)*. If S' can be covered with at most k lines, then with probability at least $1 - \frac{1}{k^4}$ one of the following holds:*

(1) L' *covers at least* $\frac{k^2}{3}$ *points;*

(2) $|L'| \geq \frac{1}{12r} k^{(1+y_{i-1})/2}$ *for some* $i \in [r]$*; or*

(3) $|L'| \geq \frac{1}{12} k^{(1+y_r)/2}$.

One technicality ensues from the definition of the saturation intervals. Since this definition entails using the term $\ln\ln\ln k$, $\ln\ln\ln k$ must be positive, and hence $k \geq 16 > e^e$. This forces a separate treatment of instances in which $k \leq 15$. Since $k = \mathcal{O}(1)$, we could opt to use a brute-force algorithm in this case, or an FPT-algorithm, but those would result in a polynomial running time of a higher degree than what is desired for our purpose. Instead, we provide an efficient linear-time algorithm for this special case in the following lemma:

▶ **Lemma 12.** *Given an instance (S, k) of* Line Cover*, where $|S| = n$ and $k \leq 15$, there is an algorithm that computes in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space a kernel (S', k') for (S, k) such that $|S'| \leq k^2$.*

▶ **Lemma 13.** *Given an instance (S, k) of* Line Cover*, where $|S| = n$,* **Alg-Kernel** *runs in time $\mathcal{O}(n \log k + k^2(\log^2 k)(\log\log k)^2)$ and space $\mathcal{O}(k^2 \log^2 k)$.*

Proof of Theorem 9 (stated at the beginning of this section). The time and space complexity of the algorithm follow from Lemmas 12 and 13. We prove its correctness next. The correctness of Step 1 was proved separately in Lemma 12, so we may assume that $k \geq 16$.

Suppose that (S, k) is a no-instance of LINE COVER. Observe that whenever the algorithm includes a subset L' of lines into the solution H (in Steps 10 and 13) (and updates S'), then the lines in L' are saturated lines, and hence, must be part of *every* solution to the instance (S, k). Therefore, either the algorithm returns an instance in Step 16 that must be a no-instance by the above observation, or returns a (trivial) no-instance in Step 9, 15, or 17. It follows from above that if (S, k) is a no-instance of LINE COVER, then **Alg-Kernel** returns a no-instance (S', k'). This proves part (2) of the theorem.

Suppose now that (S, k) is a yes-instance of LINE COVER, and hence, that S can be covered by at most k lines. By Step 9, if $L' = \emptyset$, then the algorithm will stop. Thus, Steps 7–10 will be executed at most $k + 1$ times. Consider a single execution of Steps 7–10. By Lemma 11, if S' can be covered with at most k saturated lines, then, with probability at least $1 - \frac{1}{k^4}$, **Alg-SaturatedLines**(S', k, r) returns a non-empty set L'. That is to say, **Alg-SaturatedLines**(S', k, r) fails with probability at most $\frac{1}{k^4}$. By the union bound, **Alg-Kernel**(S, k) fails during the execution of Steps 7–10 with probability at most $\frac{k+1}{k^4}$. At Step 13, by Theorem 8, with probability at least $1 - \frac{3}{|S'|^2} > 1 - \frac{3}{k^4}$, **Alg-RichLines**$(S', k)$ finds all the saturated lines in S'. After that, we have $|S'| \leq k^2$. By the union bound, with probability at least $1 - \frac{k+1}{k^4} - \frac{3}{k^4} > 1 - \frac{2}{k^3}$ (since $k \geq 16$), **Alg-Kernel**(S, k) returns a kernel (S', k') of (S, k) satisfying $|S'| \leq k^2$. This proves part (1) of the theorem. ◀

We conclude this section by giving a deterministic kernelization algorithm for LINE COVER. Recall that **Alg-RichLines** is a randomized algorithm for computing all λ-rich lines and that Guibas et al.'s algorithm [25] is a deterministic algorithm for the same purpose. We can replace **Alg-RichLines** with Guibas et al.'s algorithm [25] in the algorithms **Alg-SaturatedLines** and **Alg-Kernel** to obtain a deterministic kernelization algorithm from **Alg-Kernel** after this replacement. We can optimize the running time of this deterministic algorithm by fine-tuning the lengths of the defined intervals $I_0, \ldots, I_{r+1}$.

▶ **Theorem 14.** *There is a deterministic kernelization algorithm for* LINE COVER *that, given an instance (S, k) of* LINE COVER*, where $|S| = n$, the algorithm runs in time $\mathcal{O}(n \log k + k^3 (\log^3 k)\sqrt{\log \log k})$ and computes a kernel (S', k') such that $|S'| \leq k^2$.*

5 Lower Bounds

In this section, we establish time-complexity lower-bound results for LINE COVER and RICH LINES in the algebraic computation trees model [6]. The algebraic computation trees model is a more powerful model than the real-RAM model [19], which is the model of computation that is most commonly used to analyze geometric algorithms [38]. The lower-bound results we derive in the algebraic computation trees model apply to the real RAM model as well; for more details see [19].

5.1 LINE COVER

In order to derive lower bounds on the time complexity of kernelization algorithms for LINE COVER, we combine a lower-bound result by Grantson and Levcopoulos [23] on the time complexity of LINE COVER with a result that we prove below connecting the time complexity for solving LINE COVER to its kernelization time complexity. We remark that, since LINE

Cover is NP-hard [33] when the parameter k is unbounded, Grantson and Levcopoulos' [23] time complexity lower-bound result for Line Cover is interesting only when k is "small" relative to the input size, and should be read this way.

▶ **Theorem 15** (Grantson and Levcopoulos [23]). *There exists a constant $c > 0$ such that, for every positive $n, k \in \mathbb{N}$ satisfying $k = \mathcal{O}(\sqrt{n})$,* Line Cover *requires time at least $c \cdot n \log k$ in the algebraic computation trees model.*

We now exploit a folklore connection between kernelization and FPT [13, 14] to translate the above time-complexity lower-bound result into a kernelization time-complexity lower-bound result.

▶ **Theorem 16.** *Let Q be a parameterized problem in* NP. *For any proper complexity function h, Q has a kernelization algorithm of running time $\mathcal{O}(h(|x|, k))$, where (x, k) is the input instance to Q, if and only if Q can be solved in time $\mathcal{O}(h(|x|, k) + g(k))$ for some proper complexity function $g(k)$.*

The corollary below follows from Theorem 15 and Theorem 16 above:

▶ **Corollary 17.** *There exists a constant $c > 0$ such that the running time of any kernelization algorithm for* Line Cover *in the algebraic computation trees model is at least $cn \log k$.*

▶ Remark 18. The above corollary implies that one cannot asymptotically improve on either of the two factors n or $\log k$ in the term $n \log k$. This rules out, for instance, the possibility of a kernelization algorithm that runs in (linear) $O(n)$ time or in $\mathcal{O}(n \log \log k)$ time.

5.2 Rich Lines

In this subsection, we derive lower-bound results on the time complexity of Rich Lines in the algebraic computation trees model using Ben-Or's framework [4]. Consider the variant of the Element Distinctness problem [4]:

> Multiset Subset Distinctness
> Given a multi-set $A = \{a_1, a_2, \ldots, a_n\}$ and a positive integer λ, decide whether A can be partitioned into n/λ multi-subsets $A_1, A_2, \ldots, A_{n/\lambda}$, such that each subset A_i, where $i \in [n/\lambda]$, contains exactly λ identical elements, and no two (distinct) multi-subsets contain identical elements.

▶ **Theorem 19.** *There exists a constant $c > 0$ such that, for every positive $n, \lambda \in \mathbb{N}$ such that λ divides n,* Multiset Subset Distinctness *requires time at least $c \cdot n \log(\frac{n}{\lambda})$ in the algebraic computation trees model.*

Proof. For any fixed n and λ, the instance (A, λ), where $A = (a_1, \ldots, a_n)$, is represented as the point $(a_1, \ldots, a_n, \lambda)$ in the $(n+1)$-dimensional Euclidean space R^{n+1}. Denote by W_λ^{n+1} the set of points in R^{n+1} that corresponds to the set of yes-instances of Multiset Subset Distinctness. By Ben-Or's results [4, §4], it suffices to show that the number of connected components of W_λ^{n+1} is at least $\binom{n}{\lambda, \lambda, \ldots, \lambda} = \Theta(\frac{\sqrt{2\pi n}(n/e)^n}{(\sqrt{2\pi\lambda}(\lambda/e)^\lambda)^{n/\lambda}})$ [22, §9.6], as this would show that the depth of any algebraic computation tree for Multiset Subset Distinctness is at least $\Omega(\log \binom{n}{\lambda, \lambda, \ldots, \lambda}) = \Omega(n \log(\frac{n}{\lambda}))$.

Each yes-instance (A, λ) of Multiset Subset Distinctness corresponds to a mapping f from $[n] \to [n/\lambda]$ such that $f(i) < f(j)$ if and only if $a_i < a_j$, and such that $f(i) = f(j)$ if and only if $a_i = a_j$, and such that for each $j \in [n/\lambda]$: $|\{i \in [n] \mid f(i) = j\}| = \lambda$. It is easy to see that the number of such functions f is $\binom{n}{\lambda,\lambda,\ldots,\lambda}$. For each such function f, let W_f be the set of yes-instances corresponding to f, and let $\mathcal{W}$ be the set of all subsets W_f. It is easy to verify that the sets W_f in $\mathcal{W}$ partition W_λ^{n+1}, and that W_f is a connected region/subset in $\mathbb{R}^{n+1}$, as it is the intersection of hyperplanes with a convex set/region.

We prove that, for any two different functions f and f', W_f and $W_{f'}$ belong to two different connected component of W_λ^{n+1}. Assume to the contrary that a point $p \in W_f$ and a point $p' \in W_{f'}$ are in the same connected component of the set W_λ^{n+1}. Then there is a path Π in W_λ^{n+1} from p to p'. This path Π can be given in the parametric form as:

$$\Pi : \pi(t) = (a_1(t), a_2(t), \ldots, a_n(t), \lambda), 0 \leq t \leq 1,$$

where $\pi(0) = p$, $\pi(1) = p'$, and each $a_i(t)$, $i \in [n]$ is a continuous function of t. For an interval $I \subseteq [0,1]$, denote by $\pi(I) = \{\pi(t) \mid t \in I\}$.

Suppose first that, for each $t \in [0,1]$, there is an open interval I_t containing t such that all points in $\pi(I_t)$ are in the same subset of $\mathcal{W}$. Then by the Heine-Borel Theorem [39], we can find a finite set of open intervals covering $[0,1]$ such that for each such open interval I_t, all points in $\pi(I_t)$ are in the same subset of $\mathcal{W}$. This implies that all points on the path Π are in the same subset of $\mathcal{W}$, contradicting the fact that the subsets W_f and $W_{f'}$ are disjoint.

Suppose now that there exists a $t_0 \in [0,1]$, where $\pi(t_0)$ is in some W_{f_1}, such that for every open interval I containing t_0, $\pi(I)$ contains a point not in W_{f_1}. Since $\mathcal{W}$ is finite, we can construct a sequence $(t)_i$ in $[0,1]$ converging to t_0, and such that, for each i, $\pi(t_i)$ belongs to the *same* set $W_{f_2} \in \mathcal{W}$, where $f_1 \neq f_2$. Since $f_1 \neq f_2$, there exist indices z_1 and z_2 such that $z_1 \neq z_2$, $f_1(z_1) < f_1(z_2)$ and $f_2(z_1) > f_2(z_2)$. Consider the sequence of points

$$\pi(t_r) = (a_1(t_r), a_2(t_r), \ldots, a_n(t_r), \lambda), \text{ for } r \geq 1.$$

Since $\pi(t_r)$ approaches $\pi(t_0)$ as $t_r \to t_0$, we must have

$$|a_{z_1}(t_r) - a_{z_1}(t_0)| + |a_{z_2}(t_r) - a_{z_2}(t_0)| \to 0, \tag{1}$$

as $t_r \to t_0$. Recall that $f_1(z_1) < f_1(z_2)$ and $f_2(z_1) > f_2(z_2)$, and hence, $a_{z_1}(t_0) < a_{z_2}(t_0)$ and $a_{z_1}(t_r) > a_{z_2}(t_r)$. It follows that:

$$\begin{aligned}
&|a_{z_1}(t_r) - a_{z_1}(t_0)| + |a_{z_2}(t_r) - a_{z_2}(t_0)| && (2)\\
&\geq |(a_{z_1}(t_r) - a_{z_1}(t_0)) - (a_{z_2}(t_r) - a_{z_2}(t_0))| && (3)\\
&\geq |(a_{z_2}(t_0) - a_{z_1}(t_0)) + (a_{z_1}(t_r) - a_{z_2}(t_r))| && (4)\\
&\geq |(a_{z_2}(t_0) - a_{z_1}(t_0))|. && (5)
\end{aligned}$$

Observing that $a_{z_1}(t_0)$ and $a_{z_2}(t_0)$ are fixed, inequality (5) contradicts (1). This completes the proof. ◀

Now, we can prove a time lower bound $\Omega(n \log \frac{n}{\lambda})$ for the Rich Lines problem via a reduction from Multiset Subset Distinctness problem.

▶ **Theorem 20.** *There exists a constant $c_0 > 0$ such that, for every positive $n, \lambda \in \mathbb{N}$,* Rich Lines *requires time at least $c_0 \cdot n \log(\frac{n}{\lambda})$ in the algebraic computation trees model.*

Proof. We prove the theorem via a Turing-reduction $\mathcal{T}$ from the Multiset Subset Distinctness problem. The theorem would then follow from Theorem 19. We first present the reduction.

Given an instance $(A, \lambda) = (a_1, a_2, \ldots, a_n, \lambda)$ of Multiset Subset Distinctness, we construct the instance (P, λ) of Rich Lines, where $P = \{(a_i, i) \mid a_i \in A\}$. Note that (P, λ) can be constructed in $\mathcal{O}(n)$ time. Observe that (A, λ) is a yes-instance of Multiset Subset

Distinctness if and only if there are n/λ vertical lines that each covers exactly λ points of P. We can solve (P, λ) to find the set L of lines induced by P that each covers at least λ points. Then, we compute the subset V of vertical lines in L and accept (A, λ) if and only if $|V| = n/\lambda$. Let $t(n, \lambda)$ be the time needed to perform this reduction $\mathcal{T}$.

Now to prove the theorem, we proceed by contradiction. Suppose that no such constant c_0 exists, and let c be the universal constant in Theorem 19. Then, for every constant $c' > 0$, there exist $n, \lambda \in \mathbb{N}$ such that, for *all* input instances of size n and parameter λ, Rich Lines can be solved in time less than $c' \cdot n \log(\frac{n}{\lambda})$. We observe that, under this assumption, the number of lines in the solution to each of these instances must be less than $c' \cdot n \log(\frac{n}{\lambda})$, otherwise, the running time for solving the instance would necessarily exceed $c' \cdot n \log(\frac{n}{\lambda})$. It is not difficult to see that we can choose a constant $c' > 0$ and $n, \lambda \in \mathbb{N}$ such that for the specific function $t(n, \lambda)$, where $t(n, \lambda)$ is running time of the reduction $\mathcal{T}$ given above, we have $t(n, \lambda) + c' \cdot n \log(\frac{n}{\lambda}) < c \cdot n \log(\frac{n}{\lambda})$. Let n, λ be the values chosen accordingly.

Assume first that λ divides n, and we explain below how the proof can be modified to lift this assumption. Given an instance $(A, \lambda) = (a_1, a_2, \ldots, a_n, \lambda)$ of Multiset Subset Distinctness, where A has n elements, we reduce (A, λ) via reduction $\mathcal{T}$ to an instance (P, λ) of Rich Lines and solve (P, λ) to obtain a solution to (A, λ) in time less than $cn \log(\frac{n}{\lambda})$, contradicting Theorem 19.

In the case where λ does not divide n, let $n = r \cdot \lambda + s$, where $s < \lambda$, and let $n' = r \cdot \lambda$. Observe that the lower bound for Multiset Subset Distinctness established in Theorem 19 holds for the values n', λ (since λ divides n'). Given an instance $(A', \lambda) = (a_1, a_2, \ldots, a_{n'}, \lambda)$ of Multiset Subset Distinctness, we construct the instance (P, λ) of Rich Lines, where $P = P' \cup S$, and $P' = \{(a_i, i) \mid a_i \in A'\}$. The set S contains precisely $s < \lambda$ points and is constructed as follows. We find the smallest element $a_{min} \in A'$, and choose a number $x < a_{min}$. Define $S = \{(x, j) \mid j \in [s]\}$. It is easy to verify that (A', λ) is a yes-instance of Multiset Subset Distinctness if and only if the number of vertical lines, each containing at least λ points of P, is n'/λ. Hence, we can decide (A', λ) as explained in the first case above. Note that all the steps involved in the construction of (P, λ), including the computation of the number x, can be carried out in linear time. Since the constant c' can be chosen to be arbitrary small, it is not difficult to see that we can choose c' and the values n, λ such that the running time of the above reduction is less than $c \cdot n' \log(\frac{n'}{\lambda})$, again contradicting Theorem 19. Note also that all the operations involved in the above reduction can be equivalently modeled in the algebraic computation trees model [19]. This completes the proof. ◀

6 Conclusion

Several interesting questions ensue from our work. First, many of the previous algorithms for Rich Lines and Line Cover can be lifted to higher dimensions (e.g., see [25, 31, 41]). We believe that it is possible to lift the results in this paper to higher dimensions as well (where the covering objects are hyperplanes). Second, most of the algorithms we presented are randomized Monte Carlo algorithms. It is interesting to investigate if these algorithms can be derandomized without trading off their performance guarantees by much. Finally, it is interesting to see if the sampling and optimization techniques developed in this paper can be applied to other related problems in computational geometry. We leave all the above questions as directions for future research.

References

1. P. Afshani, E. Berglin, I. van Duijn, and J. Nielsen. Applications of incidence bounds in point covering problems. In *Proc. 32nd International Symposium on Computational Geometry (SoCG 2016),* Article No. 60, pages 1–15, 2016.
2. P. K. Agarwal and S. Sen. Randomized algorithms for geometric optimization problems. In *Handbook of Randomized Computation*, pages 151–201. Kluwer Academic Press, 2001.
3. J. Alman, M. Mnich, and V. V. Williams. Dynamic parameterized problems and algorithms. In *Proc. 44th International Colloquium on Automata, Languages and Programming (ICALP 2017),* Article No. 41, pages 1–16, 2017.
4. M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Symposium on Theory of Computing (STOC 1983)*, pages 80–86, 1983.
5. H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14:263–279, 1995.
6. P. Bürgisser, M. Clausen, and M. Shokrollahi. *Algebraic Complexity Theory*. Springer, Berlin, 1997.
7. C. Cao. *Study on two optimization problems: Line cover and maximum genus embedding*. PhD thesis, Texas A&M University, 2012.
8. R. Chitnis, G. Cormode, H. Esfandiari, M. Hajiaghayi, and M. Monemizadeh. New streaming algorithms for parameterized maximal matching and beyond. In *Proc. 27th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA 2015)*, pages 56–58, 2015.
9. K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.
10. K. L. Clarkson. Randomized geometric algorithms. In *Computing in Euclidean Geometry*, volume 1, pages 117–162. World Scientific, 1992.
11. K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct*, pages 246–252, 1993.
12. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.
13. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
14. R. Downey and M. Fellows. *Fundamentals of Parameterized Complexity*. Springer, New York, 2013.
15. J. Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16:389–418, 1996.
16. V. Estivill-Castro, A. Heednacram, and F. Suraweera. Reduction rules deliver efficient FPT-algorithms for covering points with lines. *Journal of Experimental Algorithmics*, 14:1–7, 2010.
17. V. Estivill-Castro, A. Heednacram, and F. Suraweera. FPT-algorithms for minimum-bends tours. *International Journal of Computational Geometry & Applications*, 21(2):189–213, 2011.
18. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2010.
19. H. Fournier and A. Vigneron. A tight lower bound for computing the diameter of a 3D convex polytope. *Algorithmica*, 49:245–257, 2007.
20. D. A. Freedman. *Statistical Model: Theory and Practice*. Cambridge University Press, 2009.
21. V. Froese, I. A. Kanj, A. Nichterlein, and R. Niedermeier. Finding points in general position. *International Journal of Computational Geometry and Applications*, 27(4):277–296, 2017.
22. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics: A foundation for computer science*. Addison-Wesley, Reading, MA, 1989.
23. M. Grantson and C. Levcopoulos. Covering a set of points with a minimum number of lines. In *Proc. 6th Italian Conference on Algorithms and Complexity (CIAC 2006)*, pages 6–17. Lecture Notes in Computer Science 3998, 2006.
24. J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *Geoinformatica*, 11(2):195–215, 2007.

25 L. J. Guibas, M. H. Overmars, and J. M. Robert. The exact fitting problem in higher dimensions. *Computational geometry*, 6:215–230, 1996.

26 M. Houle, H. Imai, K. Imai, J. Robert, and P. Yamamoto. Orthogonal weighted linear L_1 and L_∞ approximation and applications. *Discrete Applied Mathematics*, 43(3):217–232, 1993.

27 M. Houle and T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

28 D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

29 S. Kratsch, G. Philip, and S. Ray. Point line cover: the easy kernel is essentially tight. *ACM Transactions on Algorithms*, 12:3, 2016.

30 V. A. Kumar, S. Arya, and H. H. Ramesh. Hardness of set cover with intersection 1. In *Proc. 27th International Colloquium on Automata, Languages, and Programming (ICALP 2000)*, pages 624–635, 2000.

31 S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

32 J. Matousek. Range searching with efficient hierarchical cutting. *Discrete Computational Geometry*, 10:157–182, 1993.

33 N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1(5):194–197, 1982.

34 M. Mitzenmacher and E. Upfal. *Probability and Computing.* Cambridge University Press, 2nd edition, 2017.

35 M. Mnich. Big data algorithms beyond machine learning. *Künstliche Intelligencz*, 32:9–17, 2018.

36 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

37 J. Pach, R. Radoicic, G. Tardos, and G. Toth. Improving the crossing lemma by finding more crossings in sparse graphs. *Discrete & Computational Geometry*, 36(4):527–552, 2006.

38 F. Preparata and I. Shamos. *Computational Geometry: An Introduction.* 2nd edn. Texts and Monographs in Computer Science. Springer, New York, 1985.

39 W. Rudin. *Principles of Mathematical Analysis.* McGraw-Hill, 3rd edition, 1976.

40 E. Szemerédi and W. Trotter. Extremal problems in discrete geometry. *Combinatorica*, 3(3-4):381–392, 1983.

41 J. Wang, W. Li, and J. Chen. A parameterized algorithm for the hyperplane-cover problem. *Theoretical Computer Science*, 411:4005–4009, 2010.