Instance-Optimal Geometric Algorithms

PEYMAN AFSHANI, MADALGO, University of Aarhus
JÉRÉMY BARBAY, DCC, Universidad de Chile
TIMOTHY M. CHAN, University of Waterloo

We prove the existence of an algorithm A for computing 2D or 3D convex hulls that is optimal for *every point set* in the following sense: for every sequence σ of n points and for every algorithm A' in a certain class $\mathcal{A}$, the running time of A on input σ is at most a constant factor times the running time of A' on the worst possible permutation of σ for A'. In fact, we can establish a stronger property: for every sequence σ of points and every algorithm A', the running time of A on σ is at most a constant factor times the average running time of A' over all permutations of σ. We call algorithms satisfying these properties *instance optimal* in the *order-oblivious* and *random-order* setting. Such instance-optimal algorithms simultaneously subsume output-sensitive algorithms and distribution-dependent average-case algorithms, and all algorithms that do not take advantage of the order of the input or that assume the input are given in a random order.

The class $\mathcal{A}$ under consideration consists of all algorithms in a decision tree model where the tests involve only *multilinear* functions with a constant number of arguments. To establish an instance-specific lower bound, we deviate from traditional Ben-Or-style proofs and adopt a new adversary argument. For 2D convex hulls, we prove that a version of the well-known algorithm by Kirkpatrick and Seidel [1986] or Chan, Snoeyink, and Yap [1995] already attains this lower bound. For 3D convex hulls, we propose a new algorithm.

We further obtain instance-optimal results for a few other standard problems in computational geometry, such as maxima in 2D and 3D, orthogonal line segment intersection in 2D, finding bichromatic L_∞-close pairs in 2D, offline orthogonal range searching in 2D, offline dominance reporting in 2D and 3D, offline half-space range reporting in 2D and 3D, and offline point location in 2D. Our framework also reveals a connection to distribution-sensitive data structures and yields new results as a byproduct, for example, on online orthogonal range searching in 2D and online half-space range reporting in 2D and 3D.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Adaptive algorithms, computational geometry, convex hull, decision trees, distribution-sensitive data structures, instance optimality, output sensitivity, maxima, line segment intersection, lower bounds, orthogonal range searching, partition trees, point location

This work is partially supported by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO); the project Fondecyt Regular no 1170366 from Conicyt; and an NSERC Discovery Grant.
A preliminary version of this work appeared in *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science* (2009), 129–138.
Authors' addresses: P. Afshani, MADALGO, Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, The IT-park, Åbogade 34, DK-8200 Aarhus N, Denmark; email: peyman@cs.au.dk; J. Barbay, Departamento de Ciencias de la Computacion (DCC), Edificio Norte, Piso 3, Avenida Beauchef 851, Universidad de Chile, 837-0456 Santiago, Chile; email: jeremy@barbay.cl; T. M. Chan, Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801, USA; email: tmc@illinois.edu.

DOI: http://dx.doi.org/10.1145/3046673

ACM Reference Format:
Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. 2017. Instance-optimal geometric algorithms. J. ACM 64, 1, Article 3 (March 2017), 38 pages.
DOI: http://dx.doi.org/10.1145/3046673

1. INTRODUCTION

Instance Optimality: Our Model(s). Standard worst-case analysis of algorithms has often been criticized as overly pessimistic. As a remedy, some researchers have turned toward *adaptive* analysis where the execution cost of algorithms is measured as a function of not just the input size but also other parameters that capture in some ways the difficulty of the input instance. For example, for problems in computational geometry (the primary domain of the present article), parameters that have been considered in the past include the output size (leading to so-called *output-sensitive* algorithms) [Kirkpatrick and Seidel 1986], the spread of an input point set (the ratio of the maximum to the minimum pairwise distance) [Erickson 2005], various measures of fatness of the input objects (e.g., ratio of circumradii to inradii) [Matoušek et al. 1994] or clutteredness of a collection of objects [de Berg et al. 2002], the number of reflex angles in an input polygon, and so on.

The ultimate in adaptive algorithms is an *instance-optimal* algorithm, that is, an algorithm A whose cost is at most a constant factor from the cost of any other algorithm A' running on the same input, for *every* input instance. Unfortunately, for many problems, this requirement is too stringent. For example, consider the 2D convex hull problem, which has $\Theta(n \log n)$ worst-case complexity in the algebraic computation tree model: for every input sequence of n points, one can easily design an algorithm A' (with its code depending on the input sequence) that runs in $O(n)$ time on that particular sequence, thus ruling out the existence of an instance-optimal algorithm.[1]

To get a more useful definition, we suggest a variant of instance optimality where we ignore the order in which the input elements are given, as formalized precisely next:

Definition 1.1. Consider a problem where the input consists of a sequence of n elements from a domain $\mathcal{D}$. Consider a class $\mathcal{A}$ of algorithms. A *correct* algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in $\mathcal{D}$.

For a set S of n elements in $\mathcal{D}$, let $T_A(S)$ denote the maximum running time of A on input σ over all $n!$ possible permutations σ of S. Let $\mathrm{OPT}(S)$ denote the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$. If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \cdot \mathrm{OPT}(S)$ for every set S, then we say A is *instance optimal in the order-oblivious setting*.

For many problems, the output is a function of the input as a set rather than a sequence, and the previous definition is especially meaningful. In particular, for such problems, instance-optimal algorithms are automatically optimal output-sensitive algorithms; in fact, they are automatically optimal adaptive algorithms with respect to *any* parameter that is independent of the input order, all at the same time! This property is satisfied by simple parameters like the spread of an input point set S or more complicated quantities like the expected size $f_r(S)$ of the convex hull of a random sample of size r from S [Clarkson 1994].

[1]The length of the program for A' may depend on n in this example. If we relax the definition to permit the "constant factor" to grow as a function of the program length of A', then an instance-optimal algorithm A exists for many problems such as sorting (or more generally problems that admit linear-time verification). This follows from a trick attributed to Levin [Jones 1997], of enumerating and simulating all programs in parallel under an appropriate schedule. To say that algorithms obtained this way are impractical, however, would be an understatement.

For many algorithms (e.g., `quickhull` [Preparata and Shamos 1985], to name one), the running time is not affected so much by the order in which the input points are given but by the relative positions of the input points. Combinatorial and computational geometers more often associate "bad examples" with bad point sets rather than bad point sequences. All this supports the reasonableness and importance of the order-oblivious form of instance optimality.

We can consider a still stronger variant of instance optimality:

Definition 1.2. For a set S of n elements in $\mathcal{D}$, let $T_A^{\text{avg}}(S)$ denote the average running time of A on input σ over all $n!$ possible permutations σ of S. Let $\text{OPT}^{\text{avg}}(S)$ denote the minimum of $T_{A'}^{\text{avg}}(S)$ over all correct algorithms $A' \in \mathcal{A}$. If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \cdot \text{OPT}^{\text{avg}}(S)$ for every set S, then we say A is *instance optimal in the random-order setting*.[2]

Note that an instance-optimal algorithm in the previous sense is immediately also competitive against *randomized* (Las Vegas) algorithms A', by the easy direction of Yao's principle. The previous definition has extra appeal in computational geometry, as it is common to see the design of "randomized incremental" algorithms where the input elements are initially permuted in random order [Clarkson and Shor 1989; Mulmuley 1993].

Instance optimality in the random-order setting also implies *average-case* optimality where we analyze the expected running time under the assumption that the input elements are random and independently chosen from a common given probability distribution. (To see this, just observe that the input sequence is equally likely to be any permutation of S conditioned to the event that the set of n input elements equals any fixed set S.) An algorithm that is instance optimal in the random-order setting can deal with all probability distributions at the same time! Random-order instance optimality also remedies a common complaint about average-case analysis, that it does not provide information about an algorithm's performance on a specific input.[3]

Convex Hull: Our Main Result. After making the case for instance-optimal algorithms under our definitions, the question remains: do such algorithms actually exist, or are they "too good to be true"? Specifically, we turn to one of the most fundamental and well-known problems in computational geometry—computing the convex hull of a set of n points. Many $O(n \log n)$-time algorithms in 2D and 3D have been proposed since the 1970s [de Berg et al. 1997; Edelsbrunner 1987; Preparata and Shamos 1985], which are worst-case optimal under the algebraic computation tree model. Optimal output-sensitive algorithms can solve the 2D and 3D problem in $O(n \log h)$ time, where h is the output size. The first such output-sensitive algorithm in 2D was found by Kirkpatrick and Seidel [1986] in the 1980s and was later simplified by Chan et al. [1997] and independently by Wenger [1997]; a different, simple, optimal output-sensitive algorithm was discovered by Chan [1996b]. In 3D, the first optimal output-sensitive algorithm was obtained by Clarkson and Shor [1989] using randomization; another version was described by Clarkson [1994]. The first deterministic optimal output-sensitive algorithm in 3D was obtained by Chazelle and Matoušek [1995] via derandomization; the approach by Chan [1996b] can also be extended to 3D and yields a simpler optimal

[2]One can also consider other variations of the definition, for example, relaxing the condition to $T_A^{\text{avg}}(S) \leq O(1) \cdot \text{OPT}^{\text{avg}}(S)$ or replacing the expected running time over random permutations with analogous high-probability statements.

[3]Other models for strengthening average-case analysis have been proposed; for example, see work on self-improving algorithms [Ailon et al. 2011], smoothed analysis [Spielman and Teng 2004], and prior-independent auctions [Dhangwatnotai et al. 2015].

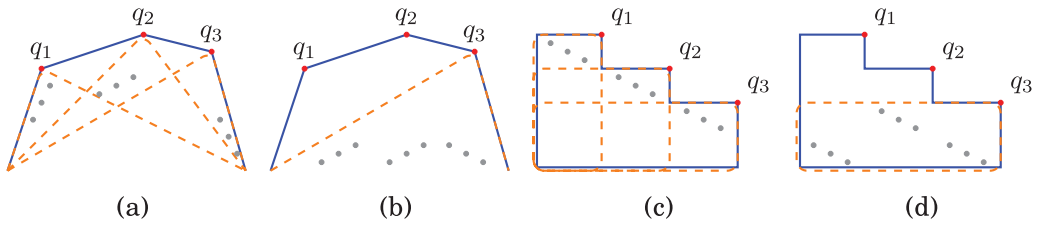

Fig. 1. (a) A "harder" point set and (b) an "easier" point set for the 2D upper hull problem. (c) A "harder" point set and (d) an "easier" point set for the 2D maxima problem. Instances where all the internal points are concentrated in one "cell" ((b) and (d)) have cost OPT $\in \Theta(n)$, while instances in which internal points are distributed more or less uniformly ((a) and (c)) have cost OPT $\in \Theta(n \log h)$.

output-sensitive algorithm. There were also average-case algorithms running in $O(n)$ expected time for certain probability distributions [Preparata and Shamos 1985], for example, when the points are independent and uniformly distributed inside a circle or a constant-sized polygon in 2D or a ball or a constant-sized polyhedron in 3D.

The convex hull problem is in some ways an ideal candidate to consider in our models. It is not difficult to think of examples of "easy" point sets and "hard" point sets (see Figure 1(a,b)). It is not difficult to think of different heuristics for pruning nonextreme points, which may not necessarily improve worst-case complexity but may help for many point sets encountered "in practice" (e.g., consider `quickhull` [Preparata and Shamos 1985]). However, it is unclear whether there is a single pruning strategy that works best on all point sets.

In this article, we show that there are indeed instance-optimal algorithms for both the 2D and 3D convex hull problem, in the order-oblivious or the stronger random-order setting. Our algorithms thus subsume all the previous output-sensitive and average-case algorithms simultaneously and are provably at least as good asymptotically as any other algorithm for every point set, so long as input order is ignored.

Techniques. We believe that our techniques—for both the upper-bound side (i.e., algorithms) and the lower-bound side (i.e., proofs of their instance optimality)—are as interesting as our results.

On the upper-bound side, we find that in the 2D case, a new algorithm is not necessary: a version of Kirkpatrick and Seidel's output-sensitive algorithm [1986], or its simplification by Chan et al. [1997], is instance optimal in the order-oblivious and random-order setting. We view this as a plus: these algorithms are simple and practical to implement [Bhattacharya and Sen 1997], and our analysis sheds new light on their theoretical complexity. In particular, our result immediately implies that a version of Kirkpatrick and Seidel's algorithm runs in $O(n)$ expected time for points uniformly distributed inside a circle or a fixed-size polygon—we were unaware of this fact before. (As another plus, our result provides a positive answer to the question in the title of Kirkpatrick and Seidel's paper, "The Ultimate Planar Convex Hull Algorithm?")

In 3D, we propose a new algorithm, as none of the previous output-sensitive algorithms seem to be instance optimal. For example, known 3D generalizations of the Kirkpatrick–Seidel algorithm have suboptimal $O(n \log^2 h)$ running time [Chan et al. 1997; Edelsbrunner and Shi 1990], while a straightforward implementation of the algorithm by Chan [1996b] fails to be instance optimal even in 2D. Our algorithm builds on Chan's technique [1996b] but requires additional ideas, notably the use of *partition trees* [Matoušek 1992].

The lower-bound side requires more innovation. We are aware of three existing techniques for proving worst-case $\Omega(n \log n)$ (or output-sensitive $\Omega(n \log h)$) lower bounds in computational geometry: (1) information-theoretic or counting arguments;

(2) topological arguments, from early work by Yao [1981] to Ben-Or [1983]; and (3) Ramsey-theory-based arguments, by Moran et al. [1985]. Ben-Or's approach is perhaps the most powerful and works in the general algebraic computation tree model, whereas Moran et al.'s approach works in a decision tree model where all the test functions have a bounded number of arguments. For an arbitrary input set S for the convex hull problem, the naive information-theoretic argument gives only an $\Omega(h \log h)$ lower bound on $\mathrm{OPT}(S)$. On the other hand, topological and Ramsey-theory approaches seem unable to give any instance-specific lower bound (e.g., modifying the topological approach is already nontrivial if we just want a lower bound for *some* integer input set [Yao 1991], let alone for *every* input set, whereas the Ramsey-theory approach considers only input elements that come from a cleverly designed subdomain).

We end up using a different lower-bound technique that is inspired by an adversary argument originally used to prove time–space lower bounds for a median finding [Chan 2010]. Note that this approach can lead to another proof of the standard $\Omega(n \log n)$ lower bounds for many geometric problems, including the problem of computing a convex hull; the proof is simple and works in any algebraic decision tree model where the test functions have at most constant degree and have at most a constant number of arguments. We build on the idea further and obtain an optimal lower bound for the convex hull problem for *every* input point set. The assumed model is more restrictive: the class $\mathcal{A}$ of allowed algorithms consists of those under a decision tree model in which the test functions are *multilinear* and have at most a constant number of arguments (e.g., this forbids testing if the Euclidean distance between two points is less than 1). Fortunately, most standard primitive operations encountered in existing convex hull algorithms satisfy the multilinearity condition (e.g., the standard determinant test does). The final proof interestingly involves partition trees [Matoušek 1992], which are more typically used in algorithms (as in our new 3D algorithm) rather than in lower-bound proofs.

So what is $\mathrm{OPT}(S)$; that is, what parameter asymptotically captures the true difficulty of a point set S for the convex hull problem? As it turns out, the bound has a simple expression (to be revealed in Section 3) and shares similarity with *entropy* bounds found in average-case or expected-case analysis of geometric data structures where query points come from a given probability distribution—these *distribution-sensitive* results have been the subject of several previous pieces of work [Arya et al. 2007a, 2007b; Collette et al. 2012; Dujmović et al. 2012; Iacono 2004]. However, lower bounds for distribution-sensitive data structures cannot be applied to our problem because our problem is offline (lower bounds for online query problems usually assume that the query algorithms fit a "classification tree" framework, but an offline algorithm may compare a query point not only with points from the dataset but also with other query points). Furthermore, although in the offline setting we can think of the query points as coming from a discrete point probability distribution, this distribution is not known in advance.[4] Lastly, distribution-sensitive data structures are usually concerned with improving the query time, but not the total time that includes preprocessing.

Other Results. The computation of the convex hull is just one problem for which we are able to obtain instance optimality. We show that our techniques can lead to instance-optimal results for many other standard problems in computational geometry, in the order-oblivious or random-order setting, including:

[4]*Self-improving* algorithms [Ailon et al. 2011] also cope with the issue of how to deal with unknown input probability distributions but are not directly comparable with our results, since in their setting each point may come from a different distribution, so input order matters.

(a) maxima in 2D and 3D;
(b) reporting/counting intersections between horizontal and vertical line segments in 2D;
(c) reporting/counting pairs of L_∞-distance at most 1 between a red point set and a blue point set in 2D;
(d) offline orthogonal range reporting/counting in 2D;
(e) offline dominating reporting in 2D and 3D;
(f) offline half-space range reporting in 2D and 3D; and
(g) offline point location in 2D.

Optimal expected-case, entropy-based data structures for the online version of (g) are known [Arya et al. 2007b; Iacono 2004], but not for (e,f)—for example, Dujmović et al. [2012] only obtained results for 2D dominance counting, a special case of 2D orthogonal range counting. Incidentally, as a consequence of our ideas, we can also get new optimal expected-case data structures for online 2D general orthogonal range counting and 2D and 3D half-space range reporting.

Related Work. Fagin et al. [2003] first coined the term "instance optimality" (when studying the problem of finding items with the k top aggregate scores in a database in a certain model), although some form of the concept has appeared before. For example, the well-known "dynamic optimality conjecture" is about instance optimality concerning algorithms for manipulating binary search trees (see Demaine et al. [2009] for the latest in a series of papers). Demaine et al. [2000] studied the problem of computing the union or intersection of k sorted sets and gave instance-optimal results for any k for union, and for constant k for intersection, in the comparison model; Barbay and Chen [2008] extended their result to the computation of the convex hull of k convex polygons in 2D for constant k. Another work about instance-optimal geometric algorithms is by Baran and Demaine [2005], who addressed an approximation problem about computing the distance of a point to a curve under a certain black-box model. Other than these, there has not been much work on instance optimality in computational geometry, especially concerning the classical problems under conventional models.

The concept of instance optimality resembles competitive analysis of online algorithms. In fact, in the online algorithms literature, our order-oblivious setting of instance optimality is related to what Boyar and Favrholdt [2007] called the *relative worst order ratio*, and our random-order setting is related to what Kenyon [1996] called the *random order ratio*. What makes instance optimality more intriguing is that we are not bounding the objective function of an optimization problem, but rather the running time of an algorithm.

General Position Assumption. Throughout the article, we assume that the input is in general position (e.g., no two points have the same x- or y-coordinate for the 2D maxima problem, and no three points are collinear for the 2D convex hull problem). The assumption can likely be removed with more care, but helps simplifying the presentation.

2. WARMUP: 2D MAXIMA

Before proving our main result on the computation of convex hulls, we find it useful to study a simpler problem: maxima in 2D. For two points p and q, we say p *dominates* q if each coordinate of p is greater than the corresponding coordinate of q. Given a set S of n points in $\mathbb{R}^d$, a point p is *maximal* if $p \in S$ and p is not dominated by any other point in S. For simplicity, we assume that the input is always nondegenerate throughout the article (e.g., no two points share the same x- or y-coordinate). The maxima problem is to report all maximal points.

For an alternative formulation, we can define the *orthant* at a point p to be the region of all points that are dominated by p. In 2D, the boundary of the union of the orthants

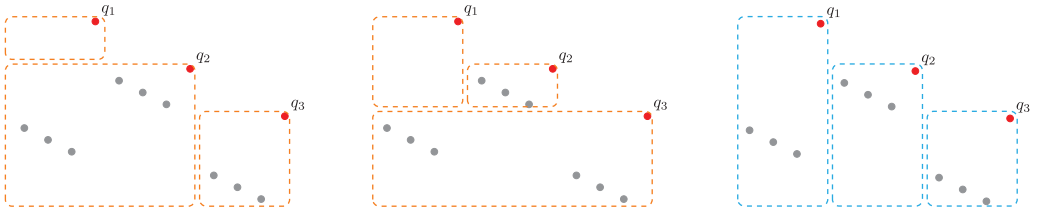

Fig. 2. Three respectful partitions of an instance of the 2D maxima problem. The two partitions on the left have entropy $\frac{1}{12}\log 12 + \frac{7}{12}\log\frac{12}{7} + \frac{4}{12}\log\frac{12}{4} \approx 1.281$. The partition Π_{vert} on the right has higher entropy $\log 3 \approx 1.585$.

at all $p \in S$ forms a *staircase*, and the maxima problem is equivalent to computing the staircase of S.

This problem has a similar history as the convex hull problem: many worst-case $O(n \log n)$-time algorithms are known; an output-sensitive algorithm by Kirkpatrick and Seidel [1985] runs in $O(n \log h)$ time for output size h; and average-case algorithms with $O(n)$ expected time have been analyzed for various probability distributions [Bentley et al. 1990; Clarkson 1994; Preparata and Shamos 1985]. The problem is simpler than computing the convex hull, in the sense that direct pairwise comparisons are sufficient. We therefore work with the class $\mathcal{A}$ of algorithms in the *comparison model,* where we can access the input points only through comparisons of the coordinate of an input point with the corresponding coordinate of another input point. The number of comparisons made by an algorithm yields a lower bound on the running time.

We define a measure $\mathcal{H}(S)$ to represent the difficulty of a point set S and prove that the optimal running time $\text{OPT}(S)$ is precisely $\Theta(n(\mathcal{H}(S) + 1))$ for the 2D maxima problem in the order-oblivious and random-order setting.

Definition 2.1. Consider a partition Π of the input set S into disjoint subsets $S_1, \ldots, S_t$. We say that Π is *respectful* if each subset S_k is either a singleton or can be enclosed by an axis-aligned box B_k whose interior is completely below the staircase of S. Define the *entropy* $\mathcal{H}(\Pi)$ of the partition Π to be $\sum_{k=1}^{t}(|S_k|/n)\log(n/|S_k|)$.[5] Define the *structural entropy* $\mathcal{H}(S)$ of the input set S to be the minimum of $\mathcal{H}(\Pi)$ over all respectful partitions Π of S.

Remark 2.2. Alternatively, we could further insist in the definition that the bounding boxes B_i are nonoverlapping and cover precisely the staircase of S. However, this will not matter, as it turns out that the two definitions yield asymptotically the same quantity (this nonobvious fact is a byproduct of our lower-bound proof in Section 2.2).

Entropy-like expressions similar to $\mathcal{H}(\Pi)$ have appeared in the analysis of expected-case geometric data structures for the case of a discrete point probability distribution, although our definition itself is nonprobabilistic. A measure proposed by Sen and Gupta [1999] is identical to $\mathcal{H}(\Pi_{\text{vert}})$, where Π_{vert} is a partition of S obtained by dividing the point set S by h vertical lines at the h maximal points of S (see Figure 2 (right) for an illustration). Note that $\mathcal{H}(\Pi_{\text{vert}})$ is at most $\log h$ (see Figure 1(c)) but can be much smaller; in turn, $\mathcal{H}(S)$ can be much smaller than $\mathcal{H}(\Pi_{\text{vert}})$ (see Figures 1(d) and 2). The complexity of the 1D multiset sorting problem [Munro and Spira 1976] also involves an entropy expression associated with one partition but does not require taking the minimum over multiple partitions.

[5]All logarithms are in base 2 unless stated otherwise.

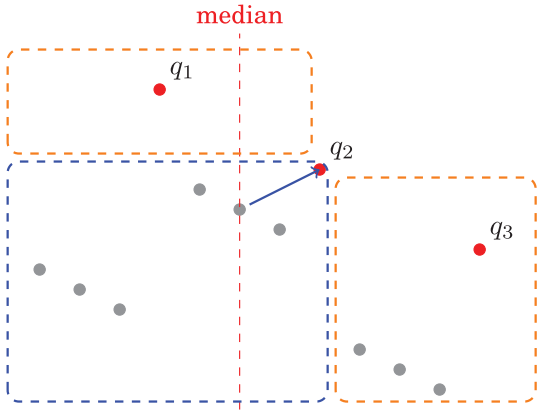

Fig. 3. Partial execution of maxima(S) after one recursion level. In this example, after computing the median x-coordinate, the algorithm found the highest point q_2 to the right of the median and pruned the six points dominated by it. Only five points are left to recurse upon, one to the left and four to the right.

2.1. Upper Bound

We use a slight variant of Kirkpatrick and Seidel's output-sensitive maxima algorithm [1985] (in their original algorithm, only points from Q_ℓ are pruned in line 4):

maxima2d(Q):

1. if $|Q| = 1$ then return Q
2. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate
3. *discover* the point q with the maximum y-coordinate in Q_r
4. *prune* all points in Q_ℓ and Q_r that are dominated by q
5. return the concatenation of maxima2d(Q_ℓ) and maxima2d(Q_r)

We call maxima2d(S) to start: Figure 3 illustrates the state of the algorithm after a single recursion level. Kirkpatrick and Seidel showed that its running time is within $O(n \log h)$, and Sen and Gupta [1999] improved this upper bound to $O(n(\mathcal{H}(\Pi_{\text{vert}}) + 1))$. Improving this bound to $O(n(\mathcal{H}(\Pi) + 1))$ for an *arbitrary* respectful partition Π of S requires a bit more finesse:

THEOREM 2.3. *Algorithm* maxima2d(S) *runs in* $O(n(\mathcal{H}(S) + 1))$ *time.*

PROOF. Consider the recursion tree of the algorithm and let X_j denote the sublist of all maximal points of S discovered during the first j recursion levels, in left-to-right order. Let $S^{(j)}$ be the subset of points of S that *survive* recursion level j, that is, that have not been pruned during levels $0, \ldots, j$ of the recursion, and let $n_j = |S^{(j)}|$. As the algorithm performs within $O(n_j)$ operations to refine level j into level $j + 1$, and there are at most $\lceil \log n \rceil$ such levels in the computation, the total running time is within $O(\sum_{j=0}^{\lceil \log n \rceil} n_j)$. Observe that

(i) there can be at most $\lceil n/2^j \rceil$ points of $S^{(j)}$ with x-coordinates between any two consecutive points in X_j, and
(ii) all points of S that are strictly below the staircase of X_j have been pruned during levels $0, \ldots, j$ of the recursion.

Let Π be *any* respectful partition of S. Consider a nonsingleton subset S_k in Π. Let B_k be a box enclosing S_k whose interior lies below the staircase of S. Fix a level j. Suppose that the upper-right corner of B_k has an x-coordinate between two consecutive

points q_i and q_{i+1} in X_j. By (ii), the only points in B_k that can survive level j must have x-coordinates between q_i and q_{i+1}. Thus, by (i), the number of points in S_k that survive level j is at most $\min\{|S_k|, \lceil n/2^j \rceil\}$. (Note that the bound is trivially true if S_k is a singleton.) Since the S_ks cover the entire point set, with a double summation we have

$$\begin{aligned}
\sum_{j=0}^{\lceil \log n \rceil} n_j &\le \sum_{j=0}^{\lceil \log n \rceil} \sum_k \min\left\{|S_k|, \lceil n/2^j \rceil\right\} \\
&= \sum_k \sum_{j=0}^{\lceil \log n \rceil} \min\left\{|S_k|, \lceil n/2^j \rceil\right\} \\
&\le \sum_k (|S_k| \lceil \log(n/|S_k|) \rceil + |S_k| + |S_k|/2 + |S_k|/4 + \cdots + 1) \\
&\le \sum_k |S_k| (\lceil \log(n/|S_k|) \rceil + 2) \\
&\in O(n(\mathcal{H}(\Pi) + 1)).
\end{aligned}$$

As Π can be *any* respectful partition of S, it can be in particular the one of minimum entropy, hence the final result. □

2.2. Lower Bound

For the lower-bound side, we first provide an intuitive justification for the bound $\Omega(n(\mathcal{H}(S) + 1))$ and point out the subtlety in obtaining a rigorous proof. Intuitively, to certify that we have a correct answer, the algorithm must gather evidence for each point p eliminated of why it is not a maximal point, by indicating at least one *witness* point in S that dominates p. We can define a partition Π by placing points with a common witness in the same subset. It is easy to see that this partition Π is respectful. The entropy bound $n\mathcal{H}(\Pi)$ roughly represents the number of bits required to encode the partition Π, so in a vague sense, $n\mathcal{H}(S)$ represents the length of the shortest "certificate" for S. Unfortunately, there could be many valid certificates for a given input set S (due to possibly multiple choices of witnesses for each nonmaximal point). If, hypothetically, all branches of an algorithm lead to a common partition Π, then a straightforward information-theoretic or counting argument would indeed prove the lower bound. The problem is that each leaf of the decision tree may give rise to a different partition Π.

In Appendix A, we show that despite the aforementioned difficulty, it is possible to obtain a proof of instance optimality via this approach, but the proof requires a more sophisticated counting argument, and also works with a different difficulty measure. Moreover, it is limited specifically to the 2D maxima problem and does not extend to 3D maxima, let alone to nonorthogonal problems such as the convex hull problem.

In this subsection, we describe a different proof, which generalizes to the other problems that we consider. The proof is based on an interesting *adversary* argument. We show in Section 4 how to adapt the proof to the random-order setting.

THEOREM 2.4. $\mathrm{OPT}(S) \in \Omega(n(\mathcal{H}(S)+1))$ *for the 2D maxima problem in the comparison model.*

PROOF. We prove that a specific respectful partition described later not only asymptotically achieves the minimum entropy among all the respectful partitions but also provides a lower bound for the running time of any comparison-based algorithm that solves the 2D maxima problem. The construction of the partition is based on *k-d trees* [de Berg et al. 1997]. We define a tree $\mathcal{T}$ of axis-aligned boxes, generated top down as

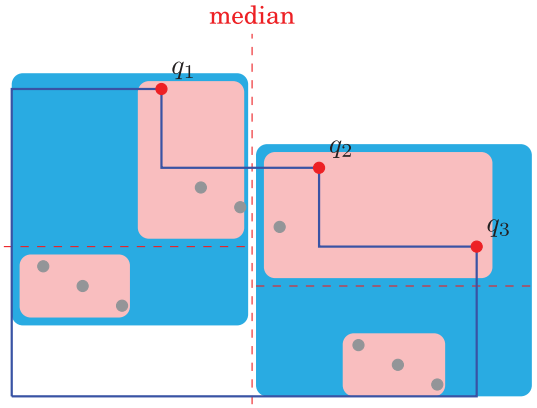

Fig. 4. The beginning of the recursive partitioning of S by the k-d tree $\mathcal{T}$, which will yield the final partition $\Pi_{\text{kd-tree}}$ for the adversarial lower bound for the 2D maxima problem. The two bottom boxes are already leaves, while the two top boxes will be divided further.

follows: The root stores the entire plane. For each node storing box B, if B is strictly below the staircase of S, or if B contains just one point of S, then B is a leaf. Otherwise, if the node is at an odd (even, respectively) depth, divide B into two subboxes by the median x-coordinate (y-coordinate, respectively) among the points of S inside B. The two subboxes are the children of B (see Figure 4 for an illustration). Note that each box B at depth j of $\mathcal{T}$ contains at least $\lfloor n/2^j \rfloor$ points of S, and consequently, the depth j is in $\Omega(\log(n/|S \cap B|))$.

Our claimed partition, denoted by $\Pi_{\text{kd-tree}}$, is one formed by the leaf boxes in this tree $\mathcal{T}$ (i.e., points in the same leaf box are placed in the same subset). By construction, $\Pi_{\text{kd-tree}}$ is respectful. We will prove that for any correct algorithm A in the comparison model, there exists a permutation of S on which the algorithm requires at least $\Omega(n\mathcal{H}(\Pi_{\text{kd-tree}}))$ comparisons. This is done using an *adversary argument*.

The adversary will construct a bad permutation of S by simulating the algorithm A on an initially unknown input sequence. In the description that follows, we distinguish between an *input element* ζ, which is represented by its index in the input sequence (its coordinates are not necessarily known), and a *point* p of S, which is represented by its coordinates (its index is not necessarily known). At each step, the algorithm can issue a comparison that involves the x- or y-coordinates of two input elements ζ and ρ. The adversary will then reveal more information about ζ and ρ to the algorithm so that the comparison can be resolved. By the end of the simulation, complete information will be revealed so that each input element ζ will be assigned to a point p of S, giving us the bad permutation of S.

More precisely, during the simulation, we maintain a box B_ζ in $\mathcal{T}$ for each input element ζ. Initially, B_ζ is set to the root box. If B_ζ corresponds to an internal node, the only information the algorithm knows about ζ currently is that it is inside B_ζ; that is, ζ can be assigned to any point in B_ζ without affecting the outcomes of the previous comparisons made.

For each box B in $\mathcal{T}$, let $n(B)$ be the number of elements ζ such that the box B_ζ is contained in B. We maintain the invariant that $n(B) \leq |S \cap B|$. If $n(B) = |S \cap B|$, we say that B is *full*. As soon as B_ζ becomes a leaf box, we assign ζ to an arbitrary point in $S \cap B_\zeta$ that has not been previously assigned (such a point exists because of the invariant); we then call ζ a *fixed* element.

Suppose that the algorithm A compares, say, the x-coordinates of two elements ζ and ρ. The easy case is when both B_ζ and B_ρ are leaf boxes. Here, ζ and ρ are already

fixed; that is, they are assigned to specific points, and thus the comparison is already resolved. The main case is when neither B_ζ nor B_ρ is a leaf box (the last remaining case when only one of them is a leaf will be considered later). The comparison is resolved in the following way:

(1) If B_ζ (B_ρ, respectively) is at even depth, we arbitrarily reset B_ζ (B_ρ, respectively) to one of its children that is not full. We can now assume that B_ζ and B_ρ are both at odd depth.
Without loss of generality, suppose that the median x-coordinate of B_ζ is less than the median x-coordinate of B_ρ. Let L_ζ be the left child of B_ζ in the k-d tree $\mathcal{T}$ and let R_ρ be the right child of B_ρ in $\mathcal{T}$. We reset B_ζ to L_ζ and B_ρ to R_ρ; if either L_ζ or R_ρ is full, we go to step 2. Now, the knowledge that ζ lies in L_ζ and ρ lies in R_ρ allows us to deduce that ζ has a smaller x-coordinate than ρ. Thus, the adversary declares to the algorithm that the x-coordinate of ζ is smaller than that of ρ and continues with the rest of the simulation.
(2) An exceptional case occurs if L_ζ is full (the case when R_ρ is full can be treated similarly). Let R_ζ be the right child of B_ζ in $\mathcal{T}$. We reset B_ζ to R_ζ, but the comparison is not necessarily resolved yet, so we go back to step 1.

Note that in both steps, the invariant is maintained. This is because L_ζ and R_ζ cannot both be full: otherwise, we would have $|S \cap B_\zeta| = |S \cap L_\zeta| + |S \cap R_\zeta| = n(L_\zeta) + n(R_\zeta)$, but $|S \cap B_\zeta| \geq n(B_\zeta) \geq n(L_\zeta) + n(R_\zeta) + 1$ (the "+1" arises because at least one element, notably, ζ, has B_ζ as its box).

The previous description can be easily modified in the case when B_ζ or B_ρ is a leaf box. If (without loss of generality) only B_ζ is a leaf, we follow step 1 except that now since ζ has been fixed, we compare the x-coordinate of ζ to the median x-coordinate of B_ρ, and reset only B_ρ.

We now prove a lower bound on the number of comparisons, T, made by the algorithm A. Let D be the sum of the depth of the boxes B_ζ in the tree $\mathcal{T}$ over all the input elements ζ at the end of the simulation of the algorithm. We will lower-bound T in terms of D. Each time we reset a box to one of its children in step 1 or 2, D is incremented; we say that an *ordinary* (*exceptional*, respectively) increment occurs at the parent box if this is done in step 1 (step 2, respectively). Each comparison generates only $O(1)$ ordinary increments. To account for exceptional increments, we use a simple amortization argument: at each box B in $\mathcal{T}$, the number of ordinary increments has to reach at least $\lfloor |S \cap B|/2 \rfloor$ first, before any exceptional increments can occur, and the number of exceptional increments is at most $\lceil |S \cap B|/2 \rceil$. Thus, the total number of exceptional increments can be upper bounded by the total number of ordinary increments, which is in $O(T)$. It follows that $D \in O(T)$, that is, $T \in \Omega(D)$.

Thus, it remains to prove our lower bound for D. We first argue that at the end of the simulation, B_ζ must be a leaf box for every input element ζ. Suppose that this is not the case. After the end of the simulation, we can do the following postprocessing: for every input element ζ where B_ζ corresponds to an internal node, we reset B_ζ to one of its nonfull children arbitrarily, and repeat. As a result, every B_ζ now becomes a leaf box, all the input elements have been assigned to points of S, and no two input elements are assigned to the same point; that is, the input is fixed to a permutation of S. The staircase of this input obviously coincides with the staircase of S. Next, consider modifying this input slightly as follows. Suppose that B_ζ was not a leaf box before the postprocessing. Then this box contained at least two points of S and was not completely underneath the staircase of S. We can either move a nonmaximal point upward or move a maximal point downward inside B_ζ and obtain a modified input that is consistent with the comparisons made but has a different set of maximal points. The algorithm would be incorrect on this modified input: a contradiction.

It follows that

$$\begin{aligned} T &\in \Omega(D) \\ &\subseteq \Omega\left(\sum_{\text{leaf}\, B} |S \cap B| \log(n/|S \cap B|)\right) \\ &\subseteq \Omega(n\mathcal{H}(\Pi_{\text{kd-tree}})) \\ &\subseteq \Omega(n\mathcal{H}(S)). \end{aligned}$$

Combined with the trivial $\Omega(n)$ lower bound, this establishes the $\Omega(n(\mathcal{H}(S)+1))$ lower bound. □

Remark 2.5. The previous proof is inspired by an adversary argument described by Chan [2010] for a 1D problem (the original proof maintains a dyadic interval for each input point, while the new proof maintains a box from a hierarchical subdivision).[6] The proof still holds for weaker versions of the problem, for example, reporting just the number of maximal points (or the parity of the number): if the algorithm stops at a point where B_ζ is not a leaf for some input element ζ, then we can modify the input such that it is consistent with all the comparisons that the algorithm has made and change the number of maximal points by one. The lower-bound proof also easily extends to any constant dimension and can be easily modified to allow comparisons of different coordinates of any two points $p = (x_1, \ldots, x_d)$ and $q = (x'_1, \ldots, x'_d)$, for example, testing whether $x_i < x'_j$ or even $x_i < x'_j + a$ for any constant a. (For a still wider class of test functions, see the next section.)

3. CONVEX HULL

We now turn to our main result on 2D and 3D convex hull. It suffices to consider the problem of computing the upper hull of an input point set S in $\mathbb{R}^d$ ($d \in \{2, 3\}$), since the lower hull can be computed by running the upper-hull algorithm on a reflection of S. (Up to constant factors, the optimal running time for convex hull is equal to the maximum of the optimal running time for upper hull and the optimal running time for lower hull, on every input.)

We work with the class $\mathcal{A}$ of algorithms in a *multilinear decision tree* model where we can access the input points only through tests of the form $f(p_1, \ldots, p_c) > 0$ for a multilinear function f, over a constant number of input points $p_1, \ldots, p_c$. We recall the following standard definition:

Definition 3.1. A function $f : (\mathbb{R}^d)^c \to \mathbb{R}$ is *multilinear* if the restriction of f is a linear function from $\mathbb{R}^d$ to $\mathbb{R}$ when any $c-1$ of the c arguments are fixed. Equivalently, f is multilinear if $f((x_{11}, \ldots, x_{1d}), \ldots, (x_{c1}, \ldots, x_{cd}))$ is a multivariate polynomial function in which each monomial has the form $x_{i_1 j_1} \cdots x_{i_k j_k}$, where $i_1, \ldots, i_k$ are all distinct (i.e., we cannot multiply coordinates from the same point).

Most of the 2D and 3D convex hull algorithms that we know fit in this framework: it supports the standard determinant test (for deciding whether p_1 is above the line through p_2, p_3 or the plane through p_2, p_3, p_4), since the determinant is a multilinear function. For another example, in 2D, comparison of the slope of the line through p_1, p_2 with the slope of the line through p_3, p_4 reduces to testing the sign of the function $(y_2 - y_1)(x_4 - x_3) - (x_2 - x_1)(y_4 - y_3)$, which is clearly multilinear. We discuss in Section 5 the relevance and limitations of the multilinear model.

[6]There are also some indirect similarities to an adversary argument for sorting due to Kahn and Kim [1995], as pointed out to us by Jean Cardinal (personal communication, 2010).

We adopt the following modified definition of $\mathcal{H}(S)$ (as before, it does not matter whether we insist that the simplices Δ_k are nonoverlapping for both the 2D and 3D problem):

Definition 3.2. A partition Π of S is *respectful* if each subset S_k in Π either is a singleton or can be enclosed by a simplex Δ_k whose interior is completely below the upper hull of S. Define the *structural entropy* $\mathcal{H}(S)$ of S to be the minimum of $\mathcal{H}(\Pi) = \sum_k (|S_k|/n) \log(n/|S_k|)$ over all respectful partitions Π of S.

3.1. Lower Bound

The lower-bound proof for computing the convex hull builds on the corresponding lower-bound proof for computing the maxima from Section 2.2 but is more involved, because a k-d tree construction no longer suffices when addressing nonorthogonal problems. Instead, we use the known following lemma:

LEMMA 3.3. *For any constant d, and for every set Q of n points in $\mathbb{R}^d$ (in general position) and $1 \le r \le n$, we can partition Q into r subsets $Q_1, \ldots, Q_r$ each of size $\Theta(n/r)$ and find r convex polyhedral cells $\gamma_1, \ldots, \gamma_r$ each with $O(\log r)$ (or fewer) facets, such that Q_i is contained in γ_i, and every hyperplane intersects at most $O(r^{1-\varepsilon})$ cells. Here, $\varepsilon > 0$ is a constant that depends only on d.*

This follows from the *partition theorem* of Matoušek [1992], who obtained the best constant $\varepsilon = 1/d$; in his construction, the cells γ_i are simplices (with $O(1)$ facets) and may overlap, and subset sizes are indeed upper and lower bounded by $\Theta(n/r)$. (A version of the partition theorem by Chan [2012] can avoid overlapping cells but does not guarantee an $\Omega(n/r)$ lower bound on the subset sizes.)

In 2D or 3D, a more elementary alternative construction follows from the four-sectioning or eight-sectioning theorem [Edelsbrunner 1987; Yao et al. 1989]: for every n-point set Q in $\mathbb{R}^2$, there exist two lines that divide the plane into four regions each with $n/4$ points; for every n-point set Q in $\mathbb{R}^3$, there exist three planes that divide space into eight regions each with $n/8$ points. Since in $\mathbb{R}^2$ a line can intersect at most three of the four regions and in $\mathbb{R}^3$ a plane can intersect at most seven of the eight regions, a simple recursive application of the theorem yields $\varepsilon = 1 - \log_4 3$ for $d = 2$ and $\varepsilon = 1 - \log_8 7$ for $d = 3$. Each resulting cell γ_i is a convex polytope with $O(\log r)$ facets, and the cells do not overlap.

We also need another fact, a geometric property about multilinear functions:

LEMMA 3.4. *If $f : (\mathbb{R}^d)^c \to \mathbb{R}$ is multilinear and has a zero in $\gamma_1 \times \cdots \times \gamma_c$ where each γ_i is a convex polytope in $\mathbb{R}^d$, then f has a zero $(p_1, \ldots, p_c) \in \gamma_1 \times \cdots \times \gamma_c$ such that all but at most one point p_i is a polytope's vertex.*

PROOF. Let $(p_1, \ldots, p_c) \in \gamma_1 \times \cdots \times \gamma_c$ be a zero of f. Suppose that some p_i does not lie on an edge of γ_i. If we fix the other $c-1$ points, the equation $f = 0$ becomes a hyperplane, which intersects γ_i and thus must intersect an edge of γ_i. We can move p_i to such an intersection point. Repeating this process, we may assume that every p_i lies on an edge $\overline{u_i v_i}$ of γ_i. Represent the line segment parametrically as $\{(1-t_i)u_i + t_i v_i \mid 0 \le t_i \le 1\}$.

Next, suppose that some two points p_i and p_j are not vertices. If we fix the other $c-2$ points and restrict p_i and p_j to lie on $\overline{u_i v_i}$ and $\overline{u_j v_j}$, respectively, the equation $f = 0$ becomes a multilinear function in two parameters $t_i, t_j \in [0,1]$. The equation has the form $at_it_j + a't_i + a''t_j + a''' = 0$ and is a hyperbola, which intersects $[0,1]^2$ and must thus intersect the boundary of $[0,1]^2$. We can move p_i and p_j to correspond to such a boundary intersection point. Then one of p_i and p_j is now a vertex. Repeating this process, we obtain the lemma. □

We are now ready for the main lower-bound proof:

THEOREM 3.5. $\mathrm{OPT}(S) \in \Omega(n(\mathcal{H}(S)+1))$ *for the upper-hull problem in any constant dimension d in the multilinear decision tree model.*

PROOF. We define a *partition tree* $\mathcal{T}$ as follows: Each node v stores a pair $(Q(v), \Gamma(v))$, where $Q(v)$ is a subset of S enclosed inside a convex polyhedral cell $\Gamma(v)$. The root stores $(S, \mathbb{R}^d)$. If $\Gamma(v)$ is strictly below the upper hull of S, or if $|Q(v)|$ drops below a constant, then v is a leaf. Otherwise, apply Lemma 3.3 with $r = b$, where b is a large enough constant, and partition $Q(v)$ into subsets $Q_1, \ldots, Q_b$ and obtain cells $\gamma_1, \ldots, \gamma_b$. For the children $v_1, \ldots, v_b$ of v, set $Q(v_i) = Q_i$ and $\Gamma(v_i) = \gamma_i \cap \Gamma(v)$. For a node v at depth j of the tree $\mathcal{T}$ we then have $|Q(v)| \geq n/\Theta(b)^j$, and consequently, the depth j is in $\Omega(\log_b(n/|Q(v)|))$. Furthermore, since $\Gamma(v)$ is the intersection of at most j convex polyhedra with at most $O(\log b)$ facets each, it has size $(j \log b)^{O(1)}$.

Let $\Pi_{\text{part-tree}}$ be the partition formed by the subsets $Q(v)$ at the leaves v in $\mathcal{T}$. Let $\widetilde{\Pi}_{\text{part-tree}}$ be a refinement of this partition obtained as follows: for each leaf v at depth j, we triangulate $\Gamma(v)$ into $(j \log b)^{O(1)}$ simplices and subpartition $Q(v)$ by placing points of $Q(v)$ from the same simplex in the same subset; if $|Q(v)|$ drops below a constant, we subpartition $Q(v)$ into singletons. Note that the subpartitioning of $Q(v)$ causes the entropy to increase[7] by at most $O((|Q(v)|/n)\log(j \log b)) \subseteq O((|Q(v)|/n)\log\log(n/|Q(v)|))$ for any constant b. The total increase in entropy is thus within $O(\mathcal{H}(\Pi_{\text{part-tree}}))$. So $\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}}) \in \Theta(\mathcal{H}(\Pi_{\text{part-tree}}))$. Clearly, $\widetilde{\Pi}_{\text{part-tree}}$ is respectful.

As in the proof of Theorem 2.4, the adversary will construct a bad permutation of S by simulating the algorithm on a sequence of initially unknown input elements. At each step, the algorithm can select c input elements $\zeta_1, \ldots, \zeta_c$ and test whether "$f(\zeta_1, \ldots, \zeta_c) > 0$?" for some multilinear function f. The adversary will then reveal more information about $\zeta_1, \ldots, \zeta_c$ so that the comparison can be resolved.

During the simulation, we maintain a node v_ζ in $\mathcal{T}$ for each input element ζ. If v_ζ is an internal node, the only information the algorithm knows about ζ currently is that it is inside $\Gamma(v_\zeta)$.

For each node v in $\mathcal{T}$, let $n(v)$ be the number of points ζ with v_ζ in the subtree rooted at v. We maintain the invariant that $n(v) \leq |Q(v)|$. If $n(v) = |Q(v)|$, we say that v is *full*. As soon as v_ζ becomes a leaf, we assign ζ to an arbitrary unassigned point in $S \cap Q(v_\zeta)$ (such a point exists because of the invariant); we then call ζ a *fixed* element.

Suppose that the simulation encounters a test "$f(\zeta_1, \ldots, \zeta_c) > 0$?". The main case is when none of the nodes v_{ζ_i} is a leaf.

(1) Consider a c-tuple $(v'_{\zeta_1}, \ldots, v'_{\zeta_c})$, where v'_{ζ_i} is a child of v_{ζ_i} for each $i \in \{1, \ldots, c\}$. We say that the tuple is *bad* if f has a zero in $\gamma(v'_{\zeta_1}) \times \cdots \times \gamma(v'_{\zeta_c})$, and *good* otherwise. We prove the existence of a good tuple by upper-bounding the number of bad tuples: if we fix all but one point ζ_i, the restriction of f can have a zero in at most $O(b^{1-\varepsilon})$ cells of the form $\gamma(v'_{\zeta_i})$, by Lemma 3.3 and the multilinearity of f. There are $O(b^{c-1}\log^{O(1)} b)$ choices of $c-1$ vertices of the cells of the form $\gamma(v'_{\zeta_1}), \ldots, \gamma(v'_{\zeta_c})$. By Lemma 3.4, it follows that the number of bad tuples is at most $O((b^{c-1}\log^{O(1)} b)\cdot b^{1-\varepsilon}) \subseteq o(b^c)$. As the number of tuples is in $\Theta(b^c)$, if b is a sufficiently large constant, then we can guarantee that some tuple $(v'_{\zeta_1}, \ldots, v'_{\zeta_c})$ is good. We reset v_{ζ_i} to v'_{ζ_i} for each $i \in \{1, \ldots, c\}$; if some v'_{ζ_i} is full, we go to step 2. Since the tuple is good, the sign of f is determined and the comparison is resolved.

[7] If $\sum_{i=1}^k q_i = q$, then by concavity of the logarithm, $\sum_{i=1}^k \frac{q_i}{n}\log\frac{n}{q_i} = \frac{q}{n}\sum_{i=1}^k \frac{q_i}{q}\log\frac{n}{q_i} \leq \frac{q}{n}\log(\sum_{i=1}^k \frac{q_i}{q}\cdot\frac{n}{q_i}) = \frac{q}{n}\log\frac{kn}{q}$, implying that $(\sum_{i=1}^k \frac{q_i}{n}\log\frac{n}{q_i}) - \frac{q}{n}\log\frac{n}{q}$ is upper bounded by $\frac{q}{n}\log k$.

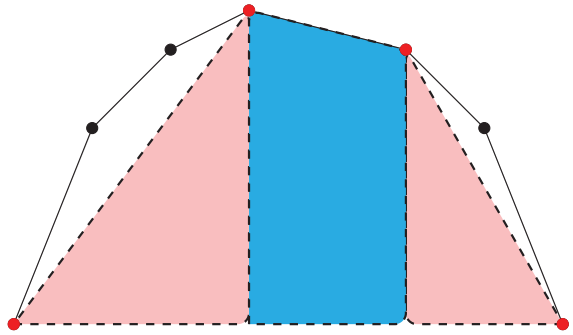

Fig. 5. Kirkpatrick and Seidel's upper-hull algorithm [1986] with an added pruning step. Line 5 prunes the points in the shaded trapezoid. The added step in line 2 prunes points in the two shaded triangles.

(2) In the exceptional case when some v'_{ζ_i} is full, we reset v_{ζ_i} instead to an arbitrary nonfull child and go back to step 1.

The previous description can be easily modified in the case when some of the nodes v_{ζ_i} are leaves, that is, when some of the elements ζ_i are fixed (we just have to lower c by the number of fixed elements).

Let T be the number of tests made. Let D be the sum of the depth of v_ζ over all input elements ζ. The same amortization argument as in the previous proof of Theorem 2.4 proves that $T \in \Omega(D)$. By an argument similar to before, at the end of the simulation, v_ζ must be a leaf for every input element ζ. It follows that

$$\begin{aligned} T &\in \Omega(D) \\ &\subseteq \Omega\left(\sum_{\text{leaf}\, v} |Q(v)| \log(n/|Q(v)|)\right) \\ &\subseteq \Omega(n\mathcal{H}(\Pi_{\text{part-tree}})) \\ &\subseteq \Omega(n\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}})) \\ &\subseteq \Omega(n\mathcal{H}(S)). \end{aligned}$$

Combined with the trivial $\Omega(n)$ lower bound, this establishes the theorem. □

The proof extends to weaker versions of the problem, for example, reporting the number of hull vertices (or its parity).

3.2. Upper Bound in 2D

To establish a matching upper bound in 2D, we use a version of the output-sensitive convex hull algorithm by Kirkpatrick and Seidel [1986] described next, where an extra pruning step is added in line 2. (This step is not new and has appeared in both `quickhull` [Preparata and Shamos 1985] and the simplified output-sensitive algorithm by Chan et al. [1997]; see Figure 5 for illustration.)

`hull2d`(Q):

1. if $|Q| = 2$ then return Q
2. *prune* all points from Q strictly below the line through the leftmost and rightmost points of Q
3. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate p_m
4. *discover* points q, q' that define the upper-hull edge $\overline{qq'}$ intersecting the vertical line at p_m (in linear time)

5. *prune* all points from Q_ℓ and Q_r that are strictly underneath the line segment $\overline{qq'}$
6. return the concatenation of `hull2d`(Q_ℓ) and `hull2d`(Q_r)

Line 4 can be done in $O(n)$ time (without knowing the upper hull beforehand) by applying a 2D linear programming algorithm in the dual [Preparata and Shamos 1985]. We call `hull2d`(S) to start. It is straightforward to show that the algorithm, even without line 2, runs in time $O(n \log h)$, or $O(n(\mathcal{H}(\Pi_{\text{vert}}) + 1))$ for the specific partition Π_{vert} of S obtained by placing points underneath the same upper-hull edge in the same subset, as was done by Sen and Gupta [1999]. To upper-bound the running time by $O(n(\mathcal{H}(\Pi) + 1))$ for an arbitrary respectful partition Π of S, we modify the proof of Theorem 2.3:

THEOREM 3.6. *Algorithm* `hull2d`(S) *runs in* $O(n(\mathcal{H}(S) + 1))$ *time.*

PROOF. Like before, let X_j denote the sublist of all hull vertices discovered during the first j levels of the recursion, in left-to-right order. Let $S^{(j)}$ be the subset of points of S that survive recursion level j, and $n_j = |S^{(j)}|$. The running time is asymptotically bounded by $\sum_{j=0}^{\lceil \log n \rceil} n_j$. Observe that

(i) there can be at most $\lceil n/2^j \rceil$ points of $S^{(j)}$ with x-coordinates between any two consecutive vertices in X_j, and
(ii) all points that are strictly below the upper hull of X_j have been pruned during levels $0, \ldots, j$ of the recursion.

Let Π be any respectful partition of S. Consider a subset S_k in Π. Let Δ_k be a triangle enclosing S_k whose interior lies below the upper hull of S. Fix a level j. If q_i and q_{i+1} are two consecutive vertices in X_j such that $\overline{q_i q_{i+1}}$ does not intersect the boundary of Δ_k (i.e., is above Δ_k), then all points in Δ_k with x-coordinates between q_k and q_{k+1} would have been pruned during the first j levels by (ii). Since only $O(1)$ edges $\overline{q_i q_{i+1}}$ of the upper hull of X_j can intersect the boundary of Δ_k, the number of points in S_k that survive level j is at most $\min\{|S_k|, O(n/2^j)\}$ by (i). We then have

$$\sum_{j=0}^{\lceil \log n \rceil} n_j \in \sum_{j=0}^{\lceil \log n \rceil} \sum_k \min\{|S_k|, O(n/2^j)\} \subseteq O(n(\mathcal{H}(\Pi) + 1)),$$

as before. □

Remark 3.7. The same result holds for the simplified output-sensitive algorithm by Chan et al. [1997], which avoids the need to invoke a 2D linear programming algorithm. (Chan et al.'s paper explicitly added the pruning step in their algorithm description.) The only difference in the previous analysis is that there can be at most $\lceil (3/4)^j n \rceil$ points of S with x-coordinates between any two consecutive vertices in X_j, since in each recursive step of Chan et al.'s algorithm, each of the two subproblems has size at most a factor of 3/4 times the original (see Chan et al. [1997] for the details).

3.3. Upper Bound in 3D

We next present an instance-optimal algorithm in 3D that matches our lower bound. Unlike in 2D, it is unclear if any of the known algorithms can be modified for this purpose. For example, obtaining an $O(n(\mathcal{H}(\Pi_{\text{vert}}) + 1))$ upper bound is already nonrivial for the specific partition Π_{vert} where points underneath the same upper-hull facet are placed in the same subset. Informed by our lower-bound proof, we suggest an algorithm that is also based on partition trees. We need the following subroutine:

LEMMA 3.8. *Given a set of n half-spaces in $\mathbb{R}^d$ for any constant d, we can answer a sequence of r linear programming queries (finding the point that maximizes a query linear function over the intersection of the half-spaces) in total time $O(n \log r + r^{O(1)})$.*

The previous lemma was obtained by Chan [1996b, 1996c] using a simple grouping trick (which was the basis of his output-sensitive $O(n \log h)$-time convex hull algorithm); the $d \geq 3$ case required randomization. A subsequent paper by Chan [1996a] gave an alternative approach using a partition construction; this eliminated randomization.

Our new upper-hull algorithm can now be described as follows, where $\delta > 0$ is some sufficiently small constant:

`hull3d`(Q):

1. for $j = 0, 1, \ldots, \lfloor \log(\delta \log n) \rfloor$ do
2. partition Q by Lemma 3.3 to get $r_j = 2^{2^j}$ subsets $Q_1, \ldots, Q_{r_j}$ and cells $\gamma_1, \ldots, \gamma_{r_j}$
3. for each $i = 1$ to r_j do
4. if γ_i is strictly below the upper hull of Q then *prune* all points in Q_i from Q
5. 5 compute the upper hull of the remaining set Q directly

THEOREM 3.9. *Algorithm* `hull3d`(S) *runs in* $O(n(\mathcal{H}(S) + 1))$ *time.*

PROOF. Let n_j be the size of Q just after iteration j. Consider iteration $j + 1$: line 2 takes $O(n_j \log r_{j+1} + r_{j+1}^{O(1)})$ time by known algorithms for Matoušek's partition theorem [1992] (or alternatively recursive application of the eight-sectioning theorem). The test in line 4 reduces to deciding whether each of the at most $O(\log r_{j+1})$ vertices of the convex polyhedral cell γ_i is strictly below the upper hull of Q. This can be done (without knowing the upper hull beforehand) by answering a 3D linear programming query in dual space. Using Lemma 3.8, we can perform lines 3 and 4 collectively in time $O(n_j \log r_{j+1} + r_{j+1}^{O(1)})$. As $r_{j+1} \leq n^\delta$, we have $\sum_{j=1}^{\lfloor \log(\delta \log n) \rfloor} r_j^{O(1)} = O(n^{O(\delta)}) = o(n)$ if δ is chosen small enough. Line 5 is done by running any $O(n_j \log n_j)$-time algorithm since for the last index (i.e., when $j = \lfloor \log(\delta \log n) \rfloor$), we have $O(n_j \log n_j) = O(n_j \log r_{j+1})$. Thus, the total running time is asymptotically bounded by $o(n) + \sum_j n_j \log r_{j+1}$.

Let Π be any respectful partition of S. Consider a (nonsingleton) subset S_k in Π. Let Δ_k be a simplex enclosing S_k whose interior lies below the upper hull of S. Fix an iteration j. Consider the subsets $Q_1, \ldots, Q_{r_j}$ and cells $\gamma_1, \ldots, \gamma_{r_j}$ at this iteration. If a cell γ_i is completely inside Δ_k, then all points inside γ_i are pruned. Since $O(r_j^{1-\varepsilon})$ cells γ_i intersect the boundary of Δ_k, the number of points in S_k that remain in Q after iteration j is at most $\min\{|S_k|, O(r_j^{1-\varepsilon} \cdot n/r_j)\} = \min\{|S_k|, O(n/r_j^\varepsilon)\}$. The S_ks cover the entire point set, so with a double summation we have

$$\begin{aligned}
\sum_j n_j \log r_{j+1} &\leq \sum_j \sum_k \min\left\{|S_k|,\ O\left(\frac{n}{2^{\varepsilon 2^j}}\right)\right\} \cdot 2^{j+1} \\
&= \sum_k \sum_j \min\left\{|S_k|,\ O\left(\frac{n}{2^{\varepsilon 2^j}}\right)\right\} \cdot 2^{j+1} \\
&\in \sum_k O\left(\sum_{j \leq \log((1/\varepsilon)\log(n/|S_k|))+1} |S_k| 2^j \ + \sum_{j > \log((1/\varepsilon)\log(n/|S_k|))+1} \frac{n}{2^{\varepsilon 2^{j-1}}} \right) \\
&\in \sum_k O\left(|S_k|(\log(n/|S_k|) + 1)\right) \\
&\in O(n(\mathcal{H}(\Pi) + 1)),
\end{aligned}$$

which yields the theorem. □

Remark 3.10. Variants of the algorithm are possible. For example, instead of recomputing the partition in line 3 at each iteration from scratch, another option is to build the partitions hierarchically as a tree. Points are pruned as the tree is generated level by level.

One minor technicality is that the previous description of the algorithm does not discuss the low-level test functions involved. In Section 5, we elaborate on this technicality and give examples of situations that could result in test functions that are not multilinear. We also explain how a modification of the algorithm can indeed be implemented in the multilinear model.

A similar approach works for the 3D maxima problem in the comparison model. We just replace partition trees with k-d trees, and replace linear programming queries with queries to test whether a point lies underneath the staircase, which can be done via an analog of Lemma 3.8.

4. EXTENSION TO THE RANDOM-ORDER SETTING

In this section, we describe how our lower-bound proofs in the order-oblivious setting can be adapted to the random-order setting. We focus on the convex hull problem and describe how to modify the proof of Theorem 3.5. We need a technical lemma first:

LEMMA 4.1. *Suppose we randomly place n elements independently in t bins, for a parameter t, where each element is placed in the kth bin with probability n_k/n. Then the probability that the kth bin contains exactly n_k elements for all $k = 1, \ldots, t$ is at least $n^{-O(t)}$.*

PROOF. The probability is exactly $\frac{n!}{n_1! \cdots n_t!} \left(\frac{n_1}{n}\right)^{n_1} \cdots \left(\frac{n_t}{n}\right)^{n_t}$, which by Stirling's formula is

$$\frac{\Theta(\sqrt{n})(n/e)^n}{\Theta(\sqrt{n_1})(n_1/e)^{n_1} \cdots \Theta(\sqrt{n_t})(n_t/e)^{n_t}} \left(\frac{n_1}{n}\right)^{n_1} \cdots \left(\frac{n_t}{n}\right)^{n_t} \subseteq \frac{1}{O(\sqrt{n})^{t-1}},$$

yielding the result. □

We now present our lower-bound proof in the random-order setting. (The proof is loosely inspired by the randomized "bit-revealing" argument by Chan [2010].)

THEOREM 4.2. $\mathrm{OPT}^{\mathrm{avg}}(S) \in \Omega(n(\mathcal{H}(S) + 1))$ *for the upper-hull problem in any constant dimension d in the multilinear decision tree model.*

PROOF. Fix a sufficiently small constant $\delta > 0$. Let $\mathcal{T}$ be as in the proof of Theorem 3.5, except that we keep only the first $\lfloor \delta \log n \rfloor$ levels of the tree; that is, when a node reaches depth $\lfloor \delta \log n \rfloor$, it is made a leaf.

Let $\Pi_{\text{part-tree}}$ be the partition of S formed by the leaf cells in $\mathcal{T}$. Let $\widetilde{\Pi}_{\text{part-tree}}$ be a refinement of $\Pi_{\text{part-tree}}$ in which each leaf cell is further triangulated and each subset corresponding to a cell of depth $\lfloor \delta \log n \rfloor$ (which has size $\Theta(n^\delta)$) is further subpartitioned into singletons. As argued before, triangulating each leaf cell causes the entropy to increase by at most a constant factor. On the other hand, subdividing a subset of size $\Theta(n^\delta)$ into singletons also causes the entropy to increase by at most a constant factor, since $\frac{n^\delta}{n} \log \frac{n}{n^\delta}$ and $n^\delta \cdot \frac{1}{n} \log \frac{n}{1}$ have the same growth rate. Thus, $\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}}) \in \Theta(\mathcal{H}(\Pi_{\text{part-tree}}))$. Clearly, $\widetilde{\Pi}_{\text{part-tree}}$ is respectful.

The adversary proceeds differently. We do not explicitly maintain the invariant that no node v is full. Whenever some v_ζ first becomes a leaf, we assign ζ to a random point among the points in $Q(v_\zeta)$ that has previously not been assigned. If all points in $Q(v_\zeta)$ have in fact been assigned, we say that *failure* has occurred.

Suppose that the simulation encounters a test "$f(\zeta_1, \ldots, \zeta_c) > 0$." We do the following:

—We reset each v_{ζ_i} to one of its children at random, where each child v'_{ζ_i} is chosen with probability $|Q(v'_{\zeta_i})|/|Q(v_{\zeta_i})|$ (which is in $\Theta(1/b)$). If the tuple $(v'_{\zeta_1}, \ldots, v'_{\zeta_c})$ is good (as defined in the proof of Theorem 3.5), then the comparison is resolved. Otherwise, we repeat.

Since we have shown that the number of bad tuples is in $o(b^c)$, the probability that the test is not resolved in one step is in $o(b^c) \cdot \Theta(1/b)^c$, which can be made less than $1/2$ for a sufficiently large constant b. The number of iterations per comparison is thus upper bounded by a geometrically distributed random variable with mean $O(1)$.

Let T be the number of comparisons made. Let D be the sum of the depth of v_ζ over all input elements ζ at the end of the simulation. Clearly, D is upper bounded by the total number of iterations performed, which is at most a sum of T independent geometrically distributed random variables with mean $O(1)$. By a version of the Chernoff bound for geometric distributions (e.g., see Appendix A.1.2 of Mulmuley [1993]), we have $D \in O(T)$ with probability at least $1 - 2^{-\Omega(T)} \geq 1 - 2^{-\Omega(n)}$.

By the same argument as before, at the end of the simulation, v_ζ must be a leaf for every input element ζ, assuming that failure has not occurred. If failure has not occurred and $D \in O(T)$, we can conclude that

$$\begin{aligned} T &\in \Omega(D) \\ &\subseteq \Omega\left(\sum_{\text{leaf}\, v} |Q(v)| \log(n/|Q(v)|)\right) \\ &\subseteq \Omega(n\mathcal{H}(\Pi_{\text{part-tree}})) \\ &\subseteq \Omega(n\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}})) \\ &\subseteq \Omega(n\mathcal{H}(S)). \end{aligned}$$

Let (†) be the event in which failure has not occurred. Let I be the generated input permutation of S (which is well defined when (†) is true). Let (∗) be the event in which the number of comparisons made by the algorithm on I is greater than $c_0 n\mathcal{H}(S)$, where c_0 is a sufficiently large constant. To summarize, we have shown that $\Pr[(\dagger) \wedge \text{not}(*)] \leq 2^{-\Omega(n)}$.

Next, to analyze $\Pr[(\dagger)]$, consider the leaf v_ζ that an element ζ ends up with after the simulation (regardless of whether failure has occurred). This is a random variable, which equals a fixed leaf v with probability $|Q(v)|/n$. Moreover, all these random variables are independent. Failure occurs if and only if for some leaf v, the number of v_ζs that equal v is different from $|Q(v)|$. By Lemma 4.1, $\Pr[(\dagger)] \geq n^{-O(n^\delta)}$, since there are $O(n^\delta)$ leaves in $\mathcal{T}$. It follows that

$$\Pr[\text{not}(*) \mid (\dagger)] = \frac{\Pr[(\dagger) \wedge \text{not}(*)]}{\Pr(\dagger)} \in \frac{2^{-\Omega(n)}}{n^{-O(n^\delta)}} \subseteq 2^{-\Omega(n)}.$$

In other words, (∗) holds with high probability in the restricted probability space where (†) is true.

Finally, observe that $\Pr[(\dagger) \wedge (I = \sigma)]$ is the same for all fixed permutations σ of S (the probability is exactly $\prod_{\text{leaf}\, v} (\frac{|Q(v)|}{n})^{|Q(v)|} \frac{1}{|Q(v)|!}$). Thus, in the restricted probability space where (†) is true, the generated input I is a random permutation of S—in other words, the adversary has not acted adversarily after all! We conclude that the number of comparisons made by the algorithm on a random permutation of S is in $\Omega(n\mathcal{H}(S))$ with high probability. In particular, the expected number of comparisons is in $\Omega(n\mathcal{H}(S))$. □

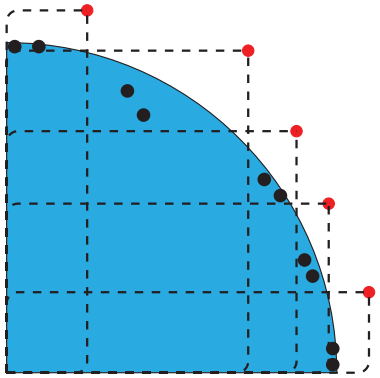

Fig. 6. An instance where $n\mathcal{H}(S)$ is no longer a lower bound if non-multilinear tests are allowed. A circular disk covers all nonmaximal points underneath the staircase. An algorithm tailored to this instance can identify the $n - h$ points that are inside the disk by $O(n)$ non-multilinear tests, and then it can compute the staircase of the remaining h points in $O(h \log h)$ time and verify that the disk is underneath the staircase by $O(h)$ additional non-multilinear tests.

Remark 4.3. Applying the same ideas to the proof of Theorem 2.4 shows that $\text{OPT}^{\text{avg}}(S) \in \Omega(n(\mathcal{H}(S) + 1))$ for the maxima problem in the comparison model.

5. ON THE MULTILINEAR MODEL

In this section, we take an in-depth view of the multilinear functions and in particular we address two main questions: First, can our lower bounds be made to work for non-multilinear functions, or at least can they be generalized to functions with constant degree? And second, does our upper-bound algorithm for 3D convex hull truly use only multilinear functions?

We start with the first question. Unfortunately, if non-multilinear test functions are allowed, then $n\mathcal{H}(S)$ may no longer be a valid instance-optimal lower bound under our definition of $\mathcal{H}(S)$. We can show this using a counterexample: one can design both an instance S of the 2D maxima problem with h output points, having $\mathcal{H}(S) \in \Omega(\log h)$ (see Figure 6), and an algorithm A that requires just $O(n + h \log h)$ operations on that instance using non-multilinear tests. Furthermore, each multilinear test used by the algorithm is only a degree 2 function of every variable. A similar example can be constructed for the 2D convex hull problem.

Nevertheless, many standard test functions commonly found in geometric algorithms are multilinear. For example, in 3D, the predicate $\text{ABOVE}(p_1, \ldots, p_4)$, which returns true if and only if p_1 is above the plane through p_2, p_3, p_4, can be reduced to testing the sign of a multilinear function (a determinant).

To see the versatility of multilinear tests, consider the following extended definition: we say that a function $f : (\mathbb{R}^d)^c \to \mathbb{R}^d$ is *quasi-multilinear* if $f(p_1, \ldots, p_c) = (f_1(p_1, \ldots, p_c), \ldots, f_d(p_1, \ldots, p_c))$, where $f_i = h_i(p_1, \ldots, p_c)/g(p_1, \ldots, p_c)$, in which $f_1, \ldots, f_d, g : (\mathbb{R}^d)^c \to \mathbb{R}$ are multilinear functions. In 3D, the function $\text{PLANE}(p_1, p_2, p_3)$ that returns the dual of the plane through p_1, p_2, p_3 is quasi-multilinear; similarly, the function $\text{INTERSECT}(p_1, p_2, p_3)$, which returns the intersection of the dual planes of p_1, p_2, p_3, is quasi-multilinear. This can be seen by expressing the answer as a ratio of determinants.

More generally, we have the following rules:

—If the function $f_i : (\mathbb{R}^3)^{c_i} \to \mathbb{R}^3$ is quasi-multilinear for each $i \in \{1, 2, 3\}$, then $\text{PLANE}(f_1(p_{11}, \ldots, p_{1c_1}), f_2(p_{21}, \ldots, p_{2c_2}), f_3(p_{31}, \ldots, p_{3c_3}))$ and $\text{INTERSECT}(f_1(p_{11}, \ldots, p_{1c_1}), f_2(p_{21}, \ldots, p_{2c_2}), f_3(p_{31}, \ldots, p_{3c_3}))$ are quasi-multilinear.

—If the function $f_i : (\mathbb{R}^3)^{c_i} \to \mathbb{R}^3$ is quasi-multilinear for each $i \in \{1, \ldots, 4\}$, then $\text{ABOVE}(f_1(p_{11}, \ldots, p_{1c_1}), \ldots, f_4(p_{41}, \ldots, p_{4c_4}))$ can be reduced to testing the sign of a multilinear function.

By combining these rules, more and more elaborate predicates can thus be reduced to testing the signs of multilinear functions, such as in the following example:

$$\text{ABOVE}\left(\begin{array}{l} p_{10}, \\ p_{11}, \\ p_{12}, \\ \text{INTERSECT}\left(\begin{array}{l} \text{PLANE}(p_1, p_2, p_3), \\ \text{PLANE}(p_4, p_5, p_6), \\ \text{PLANE}(p_7, p_8, p_9) \end{array}\right) \end{array}\right).$$

However, we may run into problems if a point occurs more than once in the expression, as in the following example:

$$\text{ABOVE}\left(\begin{array}{l} p_{10}, \\ p_{11}, \\ p_{1}, \\ \text{INTERSECT}\left(\begin{array}{l} \text{PLANE}(p_1, p_2, p_3), \\ \text{PLANE}(p_4, p_5, p_6), \\ \text{PLANE}(p_7, p_8, p_9) \end{array}\right) \end{array}\right).$$

Here, the expansion of the determinants may yield monomials of the wrong type. In most 2D algorithms, these kinds of tests do not arise or can be trivially eliminated. Unfortunately, they can occasionally occur in some 3D algorithms, including our 3D upper-hull algorithm in Section 3.3. We now describe how to modify our algorithm to avoid these problematic tests.

First, we consider the partition construction in Lemma 3.3. We choose the more elementary alternative based on the eight-sectioning theorem: there exist three planes that divide space into eight regions, each with $n/8$ points of Q. By perturbing the three planes one by one, we can ensure that each of the three planes passes through three input points, and that the resulting nine points are distinct, while changing the number of points of Q in each region by $\pm O(1)$. A brute-force algorithm can find three such planes in polynomial time. We can reduce the construction time by using the standard tool called *epsilon-approximations* [Matoušek 2000]: we compute an δ-approximation of Q in linear time for a sufficiently small constant $\delta > 0$, and then apply the polynomial algorithm to the constant-sized δ-approximation. This only changes the fraction $1/8$ by a small term $\pm O(\delta)$. It can be checked that known algorithms for epsilon-approximations [Matoušek 2000] require only multilinear tests (it suffices to check the implementation of the so-called *subsystem oracle*, which only requires the ABOVE predicate). We remove the nine defining points before recursively proceeding inside the eight regions. As a result, we can ensure that the facets in each convex polyhedral cell are all defined by planes that pass through three input points, where no two planes share a common defining point. A vertex v of a cell is an intersection of three such planes and is defined by a set of nine distinct input points, denoted $\text{DEF}(v)$.

Next, we consider the proof of Lemma 3.8 for answering r linear programming queries. We choose the alternative approach from Section 4 of Chan [1996a], which is based on a partition construction, which we have from the previous paragraph. Chan's algorithm [1996a] is based on a deterministic version of the sampling-based linear programming algorithm by Clarkson [1995] and it can also support up to r insertions and deletions of half-spaces intermixed with the query sequence. The algorithm can be implemented with simple predicates such as ABOVE.

Now, in the algorithm `hull3d`, we make one change: in line 5, we prune only when each vertex v of γ_i lies strictly below the upper hull of $Q - \text{DEF}(v)$ (instead of the upper hull of Q). In the dual, testing such a vertex v reduces to a linear programming query after deletion of $\text{DEF}(v)$ from Q, where the coefficient vector of the objective function is quasi-multilinear in $\text{DEF}(v)$. Since $\text{DEF}(v)$ has been deleted from Q, we avoid the problem of test functions where some point appears more than once in the expression. It can be checked that applying the algorithm for linear programming queries from the previous paragraph indeed requires only multilinear tests now.

Since the pruning condition has been weakened, the analysis of `hull3d` needs to be changed. Recall that the partition in line 3 is constructed by recursive application of the eight-sectioning theorem. At half the depth of recursion, we obtain an intermediate partition of Q with $O(\sqrt{r_j})$ subsets Q'_ℓ and corresponding cells γ'_ℓ, where each subset has $O(n/\sqrt{r_j})$ points and every plane intersects at most $O(\sqrt{r_j}^{1-\varepsilon})$ of these cells γ'_ℓ, for $\varepsilon = 1 - \log_8 7$. Furthermore, for a fixed γ'_ℓ, every plane intersects at most $O(\sqrt{r_j}^{1-\varepsilon})$ of the cells γ_i of the final partition inside γ'_ℓ.

In the second paragraph of the proof of Theorem 3.9, we do the analysis differently. Consider a cell γ_i of the partition, which is contained in a cell γ'_ℓ of the intermediate partition. We claim that if (1) γ'_ℓ is strictly contained in Δ_k and (2) γ_i is strictly contained in γ'_ℓ, then all points inside γ_i are pruned. To see this, notice that by (1), all points in Q'_ℓ are strictly below the upper hull of Q, and by (2), the defining points $\text{DEF}(v)$ of any vertex v of γ_i are in Q'_ℓ. Thus, the points in $\text{DEF}(v)$ are strictly below the upper hull of Q; that is, the upper hull of $Q - \text{DEF}(v)$ is the same as the upper hull of Q. As each vertex v of γ_i is strictly below the upper hull of $Q - \text{DEF}(v)$, all points inside γ_i are indeed pruned.

At most $O(\sqrt{r_j}^{1-\varepsilon})$ cells γ'_ℓ can intersect the boundary of Δ_k. For each of the $O(\sqrt{r_j})$ cells γ'_ℓ strictly contained in Δ_k, at most $O(\sqrt{r_j}^{1-\varepsilon} \log r_j)$ cells γ_i inside γ'_ℓ can intersect the $O(\log r_j)$ boundary facets of γ'_ℓ. Hence, the number of points in S_k that remain in Q after iteration j is at most $\min\{|S_k|, O(\sqrt{r_j}^{1-\varepsilon} \cdot n/\sqrt{r_j} + \sqrt{r_j} \cdot (\sqrt{r_j}^{1-\varepsilon} \log r_j) \cdot n/r_j)\} = \min\{|S_k|, O((n/r_j^{\varepsilon/2}) \log r_j)\}$. The rest of the proof is then the same, after readjusting ε by about a half.

6. OTHER APPLICATIONS

We can apply our techniques to obtain instance-optimal algorithms for a number of geometric problems in the order-oblivious and random-order setting:

(1) *Offline half-space range reporting in 2D and 3D*: Given a set S of n points and half-spaces, report the subset of points inside each half-space. Algorithms with $\Theta(n \log n + K)$ running time [Chazelle et al. 1985; Chan 2000; Afshani and Chan 2009; Chan and Tsakalidas 2015] are known for total output size K.
(2) *Offline dominance reporting in 2D and 3D*: Given a set S of n red/blue points, report the subset of red points dominated by each blue point. The problem has similar complexity as (1).
(3) *Orthogonal segment intersection in 2D*: Given a set S of n horizontal/vertical line segments, report all intersections between the horizontal and vertical segments, or count the number of such intersections. The problem is known to have worst-case complexity $\Theta(n \log n + K)$ in the reporting version for output size K, and complexity $\Theta(n \log n)$ in the counting version [de Berg et al. 1997; Preparata and Shamos 1985].
(4) *Bichromatic L_∞-close pairs in 2D*: Given a set S of n red/blue points in 2D, report all pairs (p, q) where p is red, q is blue, and p and q have L_∞-distance at most 1, or count the number of such pairs. Standard techniques in computational geometry

[de Berg et al. 1997; Preparata and Shamos 1985] yield algorithms with the same complexity as in (3).

(5) *Offline orthogonal range searching in 2D*: Given a set S of n points and axis-aligned rectangles, report the subset of points inside each rectangle, or count the number of such points inside each rectangle. The worst-case complexity is the same as in (3).

(6) *Offline point location in 2D*: Given a set S of n points and a planar connected polygonal subdivision of size $O(n)$, report the face in the subdivision containing each point. Standard data structures [de Berg et al. 1997; Preparata and Shamos 1985; Snoeyink 1997] imply a worst-case running time of $\Theta(n \log n)$.

For each of these problems, it is not difficult to see that certain input sets are indeed "easier" than others; for example, if the horizontal segments and the vertical segments respectively lie inside two bounding boxes that are disjoint, then the orthogonal segment intersection problem can be solved in $O(n)$ time. For each of these problems, we will define a measure of difficulty $\mathcal{H}(S)$ that is similar to the ones from Sections 2 and 3 and obtain an algorithm with running time $O(\mathcal{H}(S))$ and a matching lower bound $\Omega(\mathcal{H}(S))$.

Note that although some of the aforementioned problems may be reducible to others in terms of worst-case complexity, the reductions may not make sense in the instance-optimality setting. For example, an instance-optimal algorithm for a problem does not imply an instance-optimal algorithm for a restriction of the problem in a subdomain, because in the latter case, we are competing against algorithms that have to be correct only for input from this subdomain.

6.1. A General Framework for Reporting Problems

We describe our techniques for offline reporting problems in a general framework. Let $\mathcal{R} \subset \mathbb{R}^d \times \mathbb{R}^{d'}$ be a relation for some constant dimensions d and d'. We say that a red point $p \in \mathbb{R}^d$ and a blue point $q \in \mathbb{R}^{d'}$ *interact* if $(p, q) \in \mathcal{R}$. We consider the *reporting* problem: given a set S containing red points in $\mathbb{R}^d$ and blue points in $\mathbb{R}^{d'}$ of total size n, report all K interacting red/blue pairs of points in S. (By scanning the output pairs, we can then collect the subset of all blue points that interact with each red point, in $O(K)$ additional time.)

We redefine $\mathcal{H}(S)$ as follows:

Definition 6.1. Given a region γ colored red (blue, respectively), we say that γ is *safe for* S if every red (blue, respectively) point in γ interacts with exactly the same subset of blue (red, respectively) points in S. We say that a partition Π of S is *respectful* if each subset S_k in Π is a singleton (i.e., contains only one element), or a subset of red points enclosed by a safe red simplex Δ_k for S, or a subset of blue points enclosed by a safe blue simplex Δ_k for S. Define the *structural entropy* $\mathcal{H}(S)$ of S to be the minimum of $\mathcal{H}(\Pi) = \sum_k (|S_k|/n) \log(n/|S_k|)$ over all respectful partitions Π of S.

THEOREM 6.2. $\mathrm{OPT}(S), \mathrm{OPT}^{\mathrm{avg}}(S) \in \Omega(n(\mathcal{H}(S) + 1) + K)$ *for the reporting problem in the multilinear decision tree model.*

PROOF. This follows from a straightforward modification of the proofs of Theorems 3.5 and 4.2. The main differences are that now we have two lists of elements (one list of red elements and another list of blue elements) and we now keep two partition trees, one for the red (blue, respectively) points in S, with cells colored red (blue, respectively). If a cell $\Gamma(v)$ is safe for S, or if the number of red (blue, respectively) points in $\Gamma(v)$ drops below a constant, then we make v a leaf in the red (blue, respectively) partition tree. Similarly, every red (blue, respectively) element is associated with a red (blue, respectively) cell.

At the end, we argue that v_ζ must be a leaf for every red (blue, respectively) input element ζ. Otherwise, the red (blue, respectively) cell $\Gamma(v_\zeta)$ contains at least two red (blue, respectively) points and is not safe, so we can move ζ to another point inside $\Gamma(v_\zeta)$ and change the answer. The algorithm would be incorrect on the modified input. (The $\Omega(K)$ term in the lower bound is obvious.) □

For the upper-bound side, we assume the availability of three oracles concerning $\mathcal{R}$, where α is some positive constant:

(A) A worst-case algorithm for the reporting problem that runs in $O(n \log n + K)$ time
(B) A data structure with $O(n \log n)$ preprocessing time, such that we can report all κ blue (red, respectively) points in S interacting with a query red (blue, respectively) point in $O(n^{1-\alpha} + \kappa)$ time
(C) A data structure with $O(n \log n)$ preprocessing time, such that we can test whether a query red or blue convex polyhedral cell γ of size a is safe for S in $O(an^{1-\alpha})$ time

Note that we can reduce to preprocessing time in (B) to $O((n/m) \cdot m \log m) = O(n \log m)$ while increasing the query time to $O((n/m) \cdot m^{1-\alpha} + \kappa)$ for any given $1 \le m \le n$. This follows from the grouping trick by Chan [1996c]: namely, divide S into $\lceil n/m \rceil$ subsets of size $O(m)$ and build a data structure for each subset. By setting $m = \min\{r^{1/\alpha}, n\}$, we can then answer r queries in total time $O(n \log m + r \cdot (n/m) \cdot m^{1-\alpha} + \kappa) \subseteq O(n \log r + \max\{n, rn^{1-\alpha}\} + \kappa) \subseteq O(n \log r + r^{O(1)} + \kappa)$ for total output size κ. Similarly, for (C), by setting $m = \min\{(ar)^{1/\alpha}, n\}$, we can answer r queries in total time $O(n \log m + r \cdot (n/m) \cdot am^{1-\alpha} + \kappa) \subseteq O(n \log(ar) + (ar)^{O(1)})$, which simplifies to $O(n \log r + r^{O(1)})$ in our application with $a \le \log^{O(1)} r$. The grouping trick is applicable because the query problems in (B) and (C) are *decomposable*; that is, the answer of a query for a union of subsets can be obtained from the answers of the queries for the subsets.

We now solve the reporting problem by a variant of the `hull3d` algorithm in Section 3.3:

`report`(Q):

1. for $j = 0, 1, \ldots, \lfloor \log(\delta \log n) \rfloor$ do
2. partition the red points in Q by Lemma 3.3 to get $r_j = 2^{2^j}$ subsets $Q_1, \ldots, Q_{r_j}$ and red cells $\gamma_1, \ldots, \gamma_{r_j}$
3. for each $i = 1$ to r_j do
4. if γ_i is safe for Q then
5. let Z_i be the subset of blue points in Q that interact with an arbitrary red point in Q_i
6. output $Q_i \times Z_i$
7. prune all red points in Q_i from Q
8. redo lines 2–7 with "red" and "blue" reversed
9. solve the reporting problem for the remaining set Q directly

The test in line 4 for each convex polyhedral cell γ_i of size at most $O(\log r_j)$ can be done by querying the data structure in (C), and line 6 can be done by querying the data structure in (B); the cost of $O(r_j)$ queries is $O(|Q| \log r_j + r_j^{O(1)})$ plus the output size. Line 9 can be done by the algorithm in (A).

THEOREM 6.3. *Given oracles (A), (B), and (C), algorithm* `report`(S) *runs in* $O(n(\mathcal{H}(S) + 1) + K)$ *time.*

PROOF. The analysis is as in the proof of Theorem 3.9. □

The partition construction in line 2 can be done in the multilinear model, as described in Section 5. Whether the rest of the algorithm works in the multilinear model depends on the implementation of the oracles.

For orthogonal-type problems dealing with axis-aligned objects, such as problems (2) through (5) in our list, we can work instead in the comparison model. We just replace simplices with axis-aligned boxes in the definition of $\mathcal{H}(S)$, replace convex polyhedral cells with axis-aligned boxes in oracle (C), and replace partition trees with k-d trees in both the lower-bound proof and the algorithm.

We can immediately apply our framework to the reporting versions of problems (1) through (5) after checking the oracle requirements for (B) and (C) in each case:

(1) *Offline halfspace range reporting in 2D and 3D*: For the design of the needed data structures, it suffices to consider just the lower half-spaces in the input. Color the given points red, and map the given lower half-spaces to blue points by duality. The data structure problem in (B) is just half-space range reporting. The data structure problem in (C) is equivalent to testing whether any of the $O(a)$ edges of a query convex polyhedral cell intersects a given set of n hyperplanes (lines in 2D or planes in 3D). This reduces to simplex range searching [Agarwal and Erickson 1999; Matoušek 1992] by duality; known results achieve $O(n \log n)$ preprocessing time and close to $O(an^{1-1/d})$ query time. It can be checked that the entire algorithm is implementable in the multilinear model if we use the randomized algorithm by Chan [2000] for (A).

(2) *Offline dominance reporting in 2D and 3D*: The data structure problem in (B) is just dominance reporting. The data structure problem in (C) is equivalent to testing whether all the corners of a query box are dominated by the same number of points from a given n-point set. This reduces to orthogonal range counting [Agarwal and Erickson 1999; de Berg et al. 1997; Preparata and Shamos 1985]; although better data structures are known, k-d trees are sufficient for our purposes, with $O(n \log n)$ preprocessing time and $O(n^{1-1/d})$ query time. The entire algorithm works in the comparison model.

(3) *Orthogonal segment intersection in 2D*: Map each each horizontal line segment $\overline{(x, y)(x', y)}$ to a red point $(x, x', y) \in \mathbb{R}^3$ and each vertical line segment $\overline{(\xi, \eta)(\xi, \eta')}$ to a blue point $(\xi, \eta, \eta') \in \mathbb{R}^3$. Each point in $\mathbb{R}^3$ is the image of a horizontal/vertical line segment. The data structure problem in (B) for red queries corresponds to reporting all points from a given n-point set that lie in a query range of the form $\{(\xi, \eta, \eta') \in \mathbb{R}^3 : ((x \le \xi \le x') \vee (x' \le \xi \le x)) \wedge ((\eta \le y \le \eta') \vee (\eta' \le y \le \eta))\}$ for some x, x', y. This reduces to 3D orthogonal range reporting. The data structure problem in (C) for red queries corresponds to testing whether a query box in $\mathbb{R}^3$ intersects any of the boundaries of n given ranges, where each range is of the form $\{(x, x', y) \in \mathbb{R}^3 : ((x \le \xi \le x') \vee (x' \le \xi \le x)) \wedge ((\eta \le y \le \eta') \vee (\eta' \le y \le \eta))\}$ for some ξ, η, η'. This is an instance of 3D orthogonal intersection searching [Agarwal and Erickson 1999], which reduces to orthogonal range searching in a higher dimension. Again, k-d trees are sufficient for our purposes. Blue queries are symmetric. The entire algorithm works in the comparison model.

(4) *Bichromatic L_∞-close pairs in 2D*: The problem in (B) corresponds to reporting all points of a given point set that lie inside a query square of side length 2. This is an instance of orthogonal range reporting. The problem in (C) corresponds to testing whether a query box intersects any of the edges of n given squares of side length 2. This is an instance of orthogonal intersection searching. Note that here the resulting algorithm requires a slight extension of the comparison model, to include tests of the form $x_i \le x'_j + a$ mentioned in Remark 2.5, which are allowed in the lower-bound proof.

(5) *Offline orthogonal range reporting in 2D*: Color the given points red, and map each rectangle $[\xi, \xi'] \times [\eta, \eta']$ to a blue point $(\xi, \xi', \eta, \eta') \in \mathbb{R}^4$. Every point in $\mathbb{R}^4$ is the image of a rectangle. The problem in (B) for red queries corresponds to 2D rectangle stabbing, that is, reporting all rectangles, from a given set of n rectangles, that contain a query point. The problem in (B) for blue queries corresponds to 2D orthogonal range reporting. The problem in (C) for red queries corresponds to deciding whether a query box in $\mathbb{R}^2$ intersects any of the edges of n given rectangles. The problem in (C) for blue queries corresponds to deciding whether a query box in $\mathbb{R}^4$ intersects any of the boundaries of n given ranges, where each range is of the form $\{(\xi, \xi', \eta, \eta') \in \mathbb{R}^4 : ((\xi \le x \le \xi') \vee (\xi' \le x \le \xi)) \wedge ((\eta \le y \le \eta') \vee (\eta' \le y \le \eta))\}$ for some x, y. All these data structure problems reduce to orthogonal range or intersection searching. Again, the algorithm works in the comparison model.

6.2. Counting Problems

Our framework can also be applied to *counting problems*, where we simply want the total number of interacting red/blue pairs. We just change oracle (A) to a counting algorithm without the $O(K)$ term, and oracle (B) to counting data structures without the $O(\kappa)$ term. In line 5 of the algorithm, we compute $|Z|$, and in line 6, we add $|Q_i| \times |Z|$ to a global counter. The same lower- and upper-bound proofs yield an $\Theta(n(\mathcal{H}(S)+1))$ bound. The new oracle requirements are satisfied for (3) orthogonal segment intersection counting, (4) bichromatic L_∞-close pairs, and (5) offline orthogonal range counting.

We can also modify the algorithm to return *individual counts*, that is, compute the number of red points that interact with each blue point and the number of blue points that interact with each red point. Here, we need to not only strengthen oracle (A) to produce individual counts but also modify oracle (B) to the following:

(B) A data structure with $O(n \log n)$ preprocessing time, forming a collection of canonical subsets of total size $O(n \log n)$, such that we can express the subset of all blue (red, respectively) points in S interacting with a query red (blue, respectively) point, as a union of $O(n^{1-\alpha})$ canonical subsets, in $O(n^{1-\alpha})$ time.

As before, the grouping trick can be used to reduce the preprocessing time and total size of the canonical subsets. In line 4 of the algorithm, we express Z as a union of canonical subsets. In line 5, we add $|Z|$ to the counter of each red point in Q_i and add $|Q_i|$ to the counter of each canonical subset for Z. At the end of the loop in lines 3 through 7, we make a pass over each canonical subset and add its counter value to the counters of its blue points, before resetting the counter of the canonical subset. Line 8 is similar. The analysis of the running time remains the same. The strengthened oracle requirements are satisfied for problems (3), (4), and (5) by known orthogonal range searching results.

6.3. Detection Problems?

We can also consider *detection problems* where we simply want to decide whether there exists an interacting red/blue pair. Here, we redefine $\mathcal{H}(S)$ by redefining "safe": a red (blue, respectively) region γ is now considered *safe for* S if no red (blue, respectively) point in γ interacts with any blue (red, respectively) points in S. We change oracles (A) and (B) to analogous detection algorithms and data structures, without the $O(K)$ and $O(\kappa)$ terms.

The proof of the upper bound $O(n(\mathcal{H}(S)+1))$ is the same, but unfortunately, the proof of the lower bound $\Omega(n(\mathcal{H}(S)+1))$ only works for instances with a NO answer: at the end, if v_ζ is not a leaf for some red (blue, respectively) input element ζ, then $\Gamma(v_\zeta)$ contains at least two red (blue, respectively) points and is not safe, so we can move ζ to some point inside $\Gamma(v_\zeta)$ and change the answer from NO to YES.

YES instances are problematic; however, this is not a weakness of our technique but of the model: on every input set S with a YES answer, $\mathrm{OPT}(S)$ is in fact $O(n)$. To see this, consider an input set S for which there exists an interacting pair (p, q). An algorithm that is "hardwired" with the ranks of p and q in S with respect to, say, the x-sorted order of S can first find p and q from their ranks by linear-time selection, verify that p and q interact in constant time, and return YES if true or run a brute-force algorithm otherwise. Then, on every permutation of this particular set S, the algorithm always takes linear time. Many problems admit $\Omega(n \log n)$ worst-case lower bounds even when restricted to YES instances, and for such problems, instance optimality in the order-oblivious setting is therefore not possible on all instances.

6.4. Another General Framework for Offline Querying Problems

We now propose another general framework that can handle point location and related problems. Let $\mathcal{M}$ be a mapping from points in $\mathbb{R}^d$ to "answers" in some space for some constant d (the answer $\mathcal{M}(q)$ of a point $q \in \mathbb{R}^d$ may or may not have constant size depending on the context). We consider the following *offline querying* problem: given a set S of n points in $\mathbb{R}^d$, compute $\mathcal{M}(q)$ for every $q \in S$. Let K denote the total size of the answers.

We redefine $\mathcal{H}(S)$ by redefining "safe":

Definition 6.4. We say that a cell γ is *safe* if every point q in γ has the same answer $\mathcal{M}(q)$. We say that a partition Π of S is *respectful* if each subset S_k in Π is a singleton, or a subset of points enclosed by a safe simplex Δ_k. Define the *structural entropy* $\mathcal{H}(S)$ of S to be the minimum of $\mathcal{H}(\Pi) = \sum_k (|S_k|/n) \log(n/|S_k|)$ over all respectful partitions Π of S.

THEOREM 6.5. $\mathrm{OPT}(S), \mathrm{OPT}^{\mathrm{avg}}(S) \in \Omega(n(\mathcal{H}(S) + 1) + K)$ *for the offline querying problem in the multilinear decision tree model.*

PROOF. This follows from a straightforward modification of the proofs of Theorems 3.5 and 4.2. As before, if a cell $\Gamma(v)$ is safe, then we make v a leaf. At the end, we argue that v_ζ must be a leaf for every element ζ. Otherwise, $\Gamma(v_\zeta)$ is not safe, so we can move ζ to another point inside $\Gamma(v_\zeta)$ and change the answer. The algorithm would be incorrect on the modified input. □

The previous lower bound holds even if we ignore the cost of preprocessing $\mathcal{M}$ (i.e., we allow the algorithm unlimited time to build whatever data structures necessary with respect to $\mathcal{M}$, and only measure the running time after giving the algorithm the set S of n query points). Furthermore, the test functions are only required to be multilinear with respect to S, not $\mathcal{M}$.

For the upper-bound side, we assume that $\mathcal{M}$ has been preprocessed in an oracle data structure supporting the following types of queries:

(A) Given $q \in \mathbb{R}^d$, we can compute $\mathcal{M}(q)$ in $O(\log m + \kappa)$ worst-case time for output size κ, where m is a parameter describing the size of $\mathcal{M}$.
(B) Given a convex polyhedral cell γ of size a, we can test whether γ is safe in $O(am^{1-\alpha})$ time.

The algorithm is simpler this time. Instead of using a 2^{2^j} progression, we can use a more straightforward b-way recursion, for a sufficiently large constant b (the resulting recursion tree mimics the tree $\mathcal{T}$ from the lower-bound proof in Theorem 3.5, on purpose):

`off-line-queries` (Q, Γ), where $Q \subset \Gamma$:

1. if $|Q|$ drops below n/m^{δ} then compute the answers directly and return
2. partition Q by Lemma 3.3 to get b subsets $Q_1, \ldots, Q_b$ and cells $\gamma_1, \ldots, \gamma_b$
3. for $i = 1$ to b do
4. if $\gamma_i \cap \Gamma$ is safe then
5. compute $\mathcal{M}(q)$ for an arbitrary point $q \in \gamma_i \cap \Gamma$
6. output $\mathcal{M}(q)$ as the answer for all points in Q_i
7. else `off-line-queries`$(Q_i, \gamma_i \cap \Gamma)$

We call `off-line-queries`$(S, \mathbb{R}^d)$ to start. Line 1 takes $O(|Q| \log m + \kappa)$ time for output size κ by querying the data structure for (A); note that each point in Q in this case has participated in $\Omega(\log m)$ levels of the recursion, and we can account for the first term by charging each point unit cost for every level it participates in. Line 2 takes $O(|Q|)$ time for a constant b by known constructions of Matoušek's partition theorem [1992] (or alternatively recursive application of the four- or eight-sectioning theorem in the 2D or 3D case). The test in line 4 takes $O(m^{1-\alpha} \log^{O(1)} m)$ time by querying the data structure for (C), since the convex polyhedral cell $\gamma_i \cap \Gamma$ has at most $O(\log m)$ facets (and thus $O(\log^{O(1)} m)$ size). As the tree has $O(m^{\delta})$ nodes, the cost of line 4 is negligible, since its total over the entire recursion tree is sublinear in m by choosing a sufficiently small constant $\delta < \alpha$. Line 5 takes $O(\log m + \kappa)$ time for output size κ, by (A); the $O(\log m)$ term is again negligible, since its total over the entire recursion tree is sublinear in m.

THEOREM 6.6. *After $\mathcal{M}$ has been preprocessed for (A) and (C), algorithm* `off-line-queries`$(S, \mathbb{R}^d)$ *runs in $O(n(\mathcal{H}(S) + 1) + K) + o(m)$ time for total output size K.*

PROOF. Let n_j be number of points in S that *survive* level j, that is, participate in subsets Q at level j of the recursion. The total running time for the offline problem is asymptotically bounded by $\sum_j n_j$, ignoring a $o(m)$ extra term.

Let Π be any respectful partition of S. Consider a subset S_k in Π. Let Δ_k be a safe simplex enclosing S_k. Fix a level j. Let Q_is and γ_is be the subsets Q and cells γ at level j. Each Q_i has size at most $n/\Theta(b)^j$. The number of γ_is that intersect the boundary of Δ_k is at most $O(b^{1-\varepsilon})^j$. Thus, the number of points in S_k that survive level j is at most $\min\{|S_k|, O(b^{1-\varepsilon})^j \cdot n/\Theta(b)^j\}$. Since the S_ks cover the entire point set, with a double summation we have, for a sufficiently large constant b,

$$\begin{aligned}
\sum_j n_j &\leq \sum_j \sum_k \min\left\{|S_k|, n/\Theta(b)^{\varepsilon j}\right\} \\
&= \sum_k \sum_j \min\left\{|S_k|, n/\Theta(b)^{\varepsilon j}\right\} \\
&\subseteq \sum_k O(|S_k|(\log(n/|S_k|) + 1)) \\
&= O(n(\mathcal{H}(\Pi) + 1)),
\end{aligned}$$

which yields the theorem. □

For orthogonal-type problems, we can work instead in the comparison model. We just replace simplices with axis-aligned boxes in the definition of $\mathcal{H}(S)$, replace convex polyhedral cells with axis-aligned boxes in oracle (C), and replace partition trees with k-d trees in both the lower-bound proof and the algorithm.

We can apply our framework to solve problem (6):

(6) *Offline point location in 2D*: For (A), data structures for planar point location with $O(\log m)$ worst-case query time are known, with $O(m)$ preprocessing time and space [Kirkpatrick 1983; Chazelle 1991; Snoeyink 1997]. The data structure problem in (C) is equivalent to testing whether any of the $O(a)$ edges of a query polygon intersect the given polygonal subdivision. This reduces to ray shooting (or segment emptiness) queries in the subdivision, for which known results [Chazelle et al. 1994] achieve $O(\log m)$ query time, with $O(m)$ preprocessing time and space. For this problem, each answer has constant size, so the $O(K)$ and $O(\kappa)$ terms can be omitted. The total running time is $O(n(\mathcal{H}(S) + 1))$, even if preprocessing time is included, for a subdivision of size $m = O(n)$. It can be checked that the entire algorithm works in the multilinear model.

6.5. Online Querying Problems and Distribution-Sensitive Data Structures

We can also use the general framework of the preceding section to study online versions of point location and related problems. Consider the following *online querying* problem: given a set S of n points in $\mathbb{R}^d$, build a data structure so that we can compute $\mathcal{M}(q)$ for any query point $q \in \mathbb{R}^d$, while trying to minimize the *average query cost over all* $q \in S$.

Our offline lower bound states that the total time required to answer queries for all n points in S is $\Omega(n(\mathcal{H}(S) + 1))$. This immediately implies that the average query time over all $q \in S$ must be $\Omega(\mathcal{H}(S) + 1)$. (In contrast, lower bounds for the online problem do not necessarily translate to lower bounds for the offline problem.)

On the other hand, our algorithm for offline queries can be easily modified to give a data structure for online queries. We just build a data structure corresponding to the recursion tree generated by `off-line-queries`$(S, \mathbb{R}^d)$, in addition to the data structure for (A) and (C). It can be shown that with such a data structure, the average query time over all $q \in S$ is $O(\mathcal{H}(S) + 1)$ (more details are given later).

We can extend the online querying problem to the setting where each point in S is weighted and the goal is to bound the weighted average query time over the query points in S. Even more generally, we can consider the setting where S is replaced by a (possibly continuous) probability distribution and the goal is to bound the expected query time for a query point randomly chosen from S. We now provide more details for the changes needed in this most general setting.

We first redefine $\mathcal{H}(S)$ for a probability distribution S.

Definition 6.7. We say that a region γ is *safe* if every point q in γ has the same answer $\mathcal{M}(q)$. A partition Π of $\mathbb{R}^d$ into regions is *respectful* if each region S_k of Π can be enclosed by a safe simplex Δ_k. Define the *structural entropy* $\mathcal{H}(S)$ of a probability distribution S to be the minimum of $\mathcal{H}(\Pi) = \sum_k \mu_S(S_k) \log(1/\mu_S(S_k))$ over all respectful partitions Π of $\mathbb{R}^d$, where μ_S denotes the probability measure corresponding to S.

We need a continuous version of Lemma 3.3, which follows by straightforward modification to the proof of the partition theorem [Matoušek 1992] (or alternatively, recursive application of the four- or eight-sectioning theorem in the 2D or 3D case).

Lemma 6.8. *For any probability measure in* $\mathbb{R}^d$ *and* $1 \le r \le n$ *for any constant d, we can partition* $\mathbb{R}^d$ *into r (not necessarily convex or connected) polyhedral regions* $Q_1, \ldots, Q_r$ *each with measure* $\Theta(1/r)$ *and with* $r^{O(1)}$ *(or fewer) facets, and find r convex polyhedral cells* $\gamma_1, \ldots, \gamma_r$ *each with* $O(\log r)$ *(or fewer) facets, such that* Q_i *is contained in* γ_i*, and every hyperplane intersects at most* $O(r^{1-\varepsilon})$ *cells. Here,* $\varepsilon > 0$ *is a constant that depends only on d.*

The lower-bound proof is easier for online problems, so we present the simplified proof in full later. Because the input to a query algorithm is just a single point (unlike in the offline setting where we could perform a test that has more than one query point as arguments), multilinear tests can now be replaced with linear tests.

THEOREM 6.9. *Any algorithm for the online querying problem requires* $\Omega(\mathcal{H}(S)+1+\kappa)$ *expected query time for output size* κ*, for any probability distribution* S *in the linear decision tree model.*

PROOF. We define a *partition tree* $\mathcal{T}$ as follows: Each node v stores a pair $(Q(v), \Gamma(v))$, where $Q(v)$ is a region enclosed inside a convex polyhedral cell $\Gamma(v)$. The root stores $(\mathbb{R}^d, \mathbb{R}^d)$. If $\mu_S(Q(v))$ drops below $1/m^\delta$ or $\Gamma(v)$ is safe, then v is a leaf. Otherwise, apply Lemma 3.3 with $r = b$ to the restriction of μ_S to $Q(v)$, and obtain regions $Q_1, \ldots, Q_b$ and cells $\gamma_1, \ldots, \gamma_b$. For the children $v_1, \ldots, v_b$ of v, set $Q(v_i) = Q_i \cap Q(v)$ and $\Gamma(v_i) = \gamma_i \cap \Gamma(v)$. For a node v at depth j of the tree $\mathcal{T}$ we then have $\mu_S(Q(v)) \geq 1/\Theta(b)^j$, and consequently, the depth j is in $\Omega(\log_b(1/\mu_S(Q(v))))$.

Let $\Pi_{\text{part-tree}}$ be the partition formed by the cells $\Gamma(v)$ at the leaves v in $\mathcal{T}$. Let $\widetilde{\Pi}_{\text{part-tree}}$ be a refinement of this partition after triangulating the leaf cells. Note that the subpartitioning of a leaf cell at depth j causes the entropy to increase by at most $O(\mu_S(Q(v))\log(j\log b)) \subseteq O(\mu_S(Q(v))\log\log(1/\mu_S(Q(v))))$ for any constant b. So $\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}}) \in \Theta(\mathcal{H}(\Pi_{\text{part-tree}}))$. Clearly, $\widetilde{\Pi}_{\text{part-tree}}$ is respectful.

The adversary constructs a bad query point q as follows. During the simulation, we maintain a node v_q in $\mathcal{T}$, where the only information the algorithm knows about q is that q lies inside $\Gamma(v_q)$.

Suppose that the simulation encounters a test to determine which side q lies inside a hyperplane h. We do the following:

—We reset v_q to one of its children at random, where each child v'_q is chosen with probability $\mu_S(Q'_q)/\mu_S(Q_q)$ (which is in $\Theta(1/b)$). If Γ'_q does not intersect h, then the comparison is resolved. Otherwise, we repeat.

The probability that the comparison is not resolved in a single step is at most $O(b^{1-\varepsilon}) \cdot \Theta(1/b)$, which can be made less than $1/2$ for a sufficiently large constant b.

Let T be the number of tests made. Let D be the depth of v_q at the end. For $i \leq T$, let $T_i = 1$ and D_i be the number of steps needed to resolve the ith test; then $\mathbb{E}[D_i] \leq 2$. For $i > T$, let $T_i = D_i = 0$. Since $\mathbb{E}[2T_i - D_i] \geq 0$ for all i, by linearity of expectation, $\mathbb{E}[D] = \mathbb{E}[\sum_i D_i] \leq 2\,\mathbb{E}[\sum_i T_i] = 2\,\mathbb{E}[T]$.

At the end of the algorithm, v_q must be a leaf. Thus,

$$\begin{aligned}
\mathbb{E}[T] &\in \Omega(\mathbb{E}[D]) \\
&\subseteq \Omega\left(\sum_{\text{leaf } v} \mu_S(Q(v))\log(1/\mu_S(Q(v)))\right) \\
&\subseteq \Omega(\mathcal{H}(\Pi_{\text{part-tree}})) \\
&\subseteq \Omega(\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}})) \\
&\subseteq \Omega(\mathcal{H}(S)).
\end{aligned}$$

At the end, we can assign q to a random point in $Q(v_q)$ chosen from the distribution S. Then q satisfies precisely the probability distribution S (in other words, the adversary has not acted adversarily after all). Thus, $\mathbb{E}[T]$ is the expected query time for a query point randomly chosen from the distribution S. □

For the upper-bound side, following is the pseudocode for the preprocessing algorithm and the query algorithm, which are derived from our previous algorithm `off-line-`

`queries`; here, b is a sufficiently large constant. The query algorithm assumes an oracle data structure for (A) (oracle (C) is only needed in the preprocessing algorithm).

`preprocess` (Q, Γ):

1. if $\mu_S(Q) < 1/m^\delta$ then return
2. apply Lemma 6.8 to the restriction of μ_S to Q to get b regions $Q_1, \ldots, Q_b$ and cells $\gamma_1, \ldots, \gamma_b$
3. for $i = 1$ to b do
4. if $\gamma_i \cap \Gamma$ is safe then store $\mathcal{M}(q)$ for an arbitrary point $q \in \gamma_i \cap \Gamma$
5. else `preprocess`$(Q_i \cap Q, \gamma_i \cap \Gamma)$

`on-line-query` (q, Q, Γ):

1. if $\mu_S(Q) < 1/m^\delta$ then compute $\mathcal{M}(q)$ directly and return
2. locate the region Q_i containing q
3. if $\gamma_i \cap \Gamma$ was marked as safe then look up the stored answer and return
4. else `on-line-query`$(q, Q_i \cap Q, \gamma_i \cap \Gamma)$

The space of the tree generated by `preprocess`$(\mathbb{R}^d, \mathbb{R}^d)$ is only $O(m^\delta)$, and so the space for the data structure for (A) dominates. We will not focus on the preprocessing time, which depends on the construction time for Lemma 6.8, which in turn depends on the distribution S. The preprocessing can be done efficiently, for example, for a discrete n-point distribution and for many other distributions.

In algorithm `on-line-query`, line 1 takes $O(\log m)$ time by (A); note that this case occurs only when the query point q participates in $\Omega(\log m)$ levels of the recursion, and we can account for the cost by charging one unit to each level of the recursion. Line 2 takes $O(1)$ time for a constant b. We can adapt the previous proof of Theorem 6.6 for the rest of the query time analysis:

THEOREM 6.10. *After $\mathcal{M}$ has been preprocessed for (A) and* `preprocess`$(\mathbb{R}^d, \mathbb{R}^d)$ *has been executed, algorithm* `on-line-query`$(q, \mathbb{R}^d, \mathbb{R}^d)$ *runs in $O(\mathcal{H}(S)+1+\kappa)$ expected time for output size κ, for a query point q randomly chosen from the distribution S.*

PROOF. Let n_j be 1 if the query point participates at level j of the recursion, and 0 otherwise. Then the query time is asymptotically bounded by $\sum_j n_j$.

Let Π be any respectful partition of S. Consider a region S_k in Π, enclosed in a safe simplex Δ_k. Fix a level j. Let Q_is and Γ_is be the regions and cells at level j of the recursion tree. Each Q_i satisfies $\mu_S(Q_i) \le 1/\Theta(b)^j$. The number of Γ_is that intersect the boundary of Δ_k is at most $O(b^{1-\varepsilon})^j$. Thus, $\Pr[(n_j = 1) \wedge (q \in S_k)] \le \min\{\mu_S(S_k),\ O(b^{1-\varepsilon})^j \cdot 1/\Theta(b)^j\}$ for a point q randomly chosen from the distribution S. Since the S_ks cover $\mathbb{R}^d$, with a double summation we have, for a sufficiently large constant b,

$$\begin{aligned}
\mathbb{E}\left[\sum_j n_j\right] &\le \sum_j \sum_k \min\{\mu_S(S_k), 1/\Theta(b)^{\varepsilon j}\} \\
&= \sum_k \sum_j \min\{\mu_S(S_k), 1/\Theta(b)^{\varepsilon j}\} \\
&\subseteq \sum_k O\left(\mu_S(S_k)(\log(1/\mu_S(S_k)) + 1)\right) \\
&= O(\mathcal{H}(\Pi) + 1),
\end{aligned}$$

which yields the theorem. □

For orthogonal-type problems, we can again work in the comparison model, by replacing simplicial and convex polyhedral cells with axis-aligned boxes.

We can apply our framework to online versions of several problems:

(6) *Online point location queries in 2D*: We immediately obtain optimal $O(\mathcal{H}(S) + 1)$ expected query cost, with an $O(m)$-space data structure for a subdivision of size m, for any given distribution S. The query algorithm works in the linear decision tree model. This online point location result is known before [Arya et al. 2007a, 2007b; Collette et al. 2012; Iacono 2004] (some of these previous works even optimize the constant factor in the query cost).

(1) *Online half-space range reporting queries in 2D and 3D*: Here, we map query lower half-spaces to points by duality. For (A), data structures for 2D and 3D half-space range reporting with $O(\log m + \kappa)$ worst-case time are known, with $O(m)$ space [Chazelle et al. 1985; Afshani and Chan 2009]. We thus obtain optimal $O(\mathcal{H}(S) + 1 + \kappa)$ expected query cost for output size κ, with an $O(m)$-space data structure for a given 2D or 3D m-point set, for any given distribution S. The query algorithm works in the linear decision tree model. This result is new.

(2) *Online dominance reporting queries in 2D and 3D*: The story is similar to half-space range reporting. The query algorithm now works in the comparison model.

(4) *Online orthogonal range reporting/counting queries in 2D*: Here, we map query rectangles to points in 4D as in Section 6.1. For (A), data structures for 2D orthogonal range reporting with $O(\log m + \kappa)$ worst-case query time are known, with $O(m \log^{\varepsilon} m)$ space; and data structures for 2D orthogonal range counting with $O(\log m)$ worst-case query time are known, with $O(m)$ space [Chazelle 1988]. For reporting, we thus obtain optimal $O(\mathcal{H}(S) + 1 + \kappa)$ expected query cost for output size κ, with an $O(m \log^{\varepsilon} m)$-space data structure for a given 2D m-point set, for any given distribution S; for counting, we get optimal $O(\mathcal{H}(S) + 1)$ expected query cost with an $O(m)$-space data structure. The query algorithm works in the comparison model. This result is apparently new, as it extends Dujmović, Howat, and Morin's result on 2D dominance counting [Dujmović et al. 2012] and unintentionally answers one of their main open problems (and at the same time improves their space bound from $O(m \log m)$ to $O(m)$).

Remark 6.11. Some months after the appearance of the conference version of the present article, a similar general technique for distribution-sensitive data structures was rediscovered by Bose et al. [2010].

Since the conference version of our article that outlined the new distribution-sensitive data structures, Nguyen [2015] has also expanded the description of our approach (and in particular provided self-contained proofs of the continuous version of the partition theorem).

7. DISCUSSION

Although we have argued for the order-oblivious form of instance optimality, we are not denigrating adaptive algorithms that exploit the order of the input. Indeed, for some geometric applications, the input order may exhibit some sort of locality of reference that can speed up algorithms. There are various parameters that one can define to address this issue, but it is unclear how a unified theory of instance optimality can be developed for order-dependent algorithms.

We do not claim that the algorithms described here are the best in practice, because of possibly larger constant factors (especially those that use Matoušek's partition trees), although some variations of the ideas might actually be useful. In some sense, our results can be interpreted as a theoretical explanation for why heuristics based on bounding boxes and BSP trees perform so well (e.g., see Andrews et al. [1994] on

experimental results for the red/blue segment intersection problem), as many of our instance-optimal algorithms prune input based on bounding boxes and spatial tree structures.

Note that specializations of our techniques to 1D can also lead to order-oblivious instance-optimal results for the multiset-sorting problem and the problem of computing the intersection of two (unsorted) sets. Adaptive algorithms for similar 1D problems (e.g., Munro and Spira [1976]) were studied in different settings from ours.

Not all standard geometric problems admit nontrivial instance-optimal results in the order-oblivious setting. For example, computing the Voronoi diagram of n points or the trapezoidal decomposition of n disjoint line segments, both having $\Theta(n)$ sizes, requires $\Omega(n \log n)$ time for every point set by the naive information-theoretic argument. Computing the (L_∞-)closest pair for a *monochromatic* point set requires $\Omega(n \log n)$ time for every point set by our adversary lower-bound argument.

An open problem is to strengthen our lower-bound proofs to allow for a more general class of test functions beyond multilinear functions, for example, arbitrary fixed-degree algebraic functions.

It remains to be seen how widely applicable the concept of instance optimality is. To inspire further work, we mention the following geometric problems for which we currently are unable to obtain instance-optimal results:

(a) Reporting all intersections between a set of disjoint red (nonorthogonal) line segments and a set of disjoint blue line segments in 2D
(b) Computing the L_2- or L_∞-closest pair between a set of red points and a set of blue points in 2D
(c) Computing the diameter or the width of a 2D point set
(d) Computing the lower envelope of a set of (perhaps disjoint) line segments in 2D

Finally, we should mention that all our current results concern at most logarithmic-factor improvements. Obtaining some form of instance-optimal results for problems with $\omega(n \log n)$ worst-case complexity (e.g., offline triangular range searching, 3SUM-hard problems, etc.) would be even more fascinating.

APPENDIX

A. AN ALTERNATIVE PROOF FOR 2D MAXIMA

In this appendix, we describe an alternative approach to the 2D maxima problem, which uses a new definition of a difficulty measure and a vastly different lower-bound proof based on an interesting encoding argument. This approach is more specialized and does not seem to work for 3D maxima or other problems, but the lower-bound proof has the advantage of being generalizable to nondeterministic algorithms.

We begin by defining a measure of difficulty $\mathcal{F}(S)$ specific to the 2D maxima problem, which is seemingly different from the structural entropy $\mathcal{H}(S)$ defined previously. The new definition appears simpler in the sense that we do not need to take the minimum over all partitions but measure the contribution of each point directly, but as a byproduct of our analyses, $\mathcal{F}(S)$ is asymptotically equivalent to $n\mathcal{H}(S)$ (which is why we do not give it a name).

Definition A.1. Given a point set S, let $q_1, \ldots, q_h$ denote the maximal points of S from left to right, with $q_0 = (-\infty, \infty)$ and $q_{h+1} = (\infty, -\infty)$. Given a point $p \in S$, let $q_i, \ldots, q_\ell$ be all the maximal points that dominate p. Define $F(p)$ to be the subset of all points in S in the slab $(q_{i-1}.x, q_{\ell+1}.x) \times \mathbb{R}$, where we use $p.x$ and $p.y$ to denote the x- and y-coordinates of p. Define $\mathcal{F}(S) = \sum_{p \in S} \log(n/|F(p)|)$. (See Figure 7.)

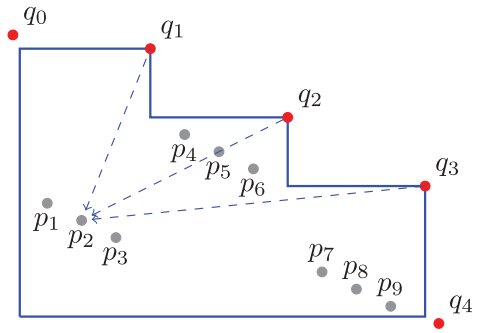

Fig. 7. Definition of $\mathcal{F}(S)$: In this instance, points p_1, p_2, p_3 are dominated by q_1, q_2, q_3, and so $|F(p_1)| = |F(p_2)| = |F(p_3)| = n = 12$. Points p_4, p_5, p_6, q_2 are dominated only by q_2, and so $|F(p_4)| = |F(p_5)| = |F(p_6)| = |F(q_2)| = 7$. Similarly, $|F(p_7)| = |F(p_8)| = |F(p_9)| = |F(q_3)| = 4$ and $|F(q_1)| = 7$. Thus, $\mathcal{F}(S) = 3\log\frac{12}{12} + 5\log\frac{12}{7} + 4\log\frac{12}{4}$.

For the upper-bound side, we use the same algorithm and an analysis similar to before:

THEOREM A.2. *Algorithm* `maxima2d`(S) *from Section 2.1 runs in* $O(\mathcal{F}(S) + n)$ *time.*

PROOF. We proceed as in the proof of Theorem 2.3, but a simpler argument replaces the second paragraph: Fix a point $p \in S$. Let $q_i, \ldots, q_\ell$ be all the maximal points that dominate p. Fix a level j. If $|F(p)| > \lfloor n/2^j \rfloor$, then by (i), some maximal point from $\{q_i, \ldots, q_\ell\}$ must been discovered, and by (ii), this implies that p does not survive level j. Thus, p can survive only for $O(\log(n/|F(p)|)+1)$ levels. We can asymptotically bound the running time by $\sum_j n_j \in O(\sum_p(\log(n/|F(p)|) + 1)) = O(\mathcal{F}(S) + n)$. □

For the lower-bound side, we first consider a slightly strengthened problem, which we call *maxima with witnesses*: given a point set S, report (the indices of) all maximal points in left-to-right order, and for each nonmaximal point p in S, report a maximal point (a *witness*) that dominates p.

THEOREM A.3. $\mathrm{OPT}(S), \mathrm{OPT}^{\mathrm{avg}}(S) \in \Omega(\mathcal{F}(S) + n)$ *for the 2D "maxima with witness" problem in the comparison model.*

PROOF. The proof is a counting argument, which we express in terms of encoding schemes (see Demaine and López-Ortiz [2003] and Golynski [2009] for more sophisticated examples of counting arguments based on encoding/decoding). We will describe a way to encode an arbitrary permutation σ of S, so that the length of the encoding can be upper bounded in terms of the running time of the given algorithm A on input σ. Since the worst-case encoding length must be at least $\log(n!)$, the running time must be large for some permutation σ.

To describe the encoding scheme, we imagine that the permutation σ is initially unknown to the decoder, and as we proceed, we record bits of information about σ so that at the end, the decoder can uniquely determine σ from the bits recorded, given S. In the description that follows, we distinguish between an *input element*, which is represented by its index in the input permutation σ (its coordinates are not necessarily known to the decoder), and a *point* of S, which is represented by its coordinates (its index in σ is not necessarily known). At any moment, if the bits recorded so far allow the decoder to infer which input element corresponds to a point p of S, we say that p is *known*.

We first run the algorithm on σ and record the outcomes of the comparisons made; this requires $T_A(\sigma)$ bits, where $T_A(\sigma)$ denotes the number of comparisons made by A on σ. Let M be the list of maximal input elements returned. For each input element q_i, let $W(q_i)$ be the list of all nonmaximal input elements that have q_i as witness. The decoder can determine these lists M and $W(q_i)$ by simulating the algorithm on σ using the comparison outcomes recorded. For each maximal point of S, we record its position in M, using $h\lceil \log h \rceil$ bits in total. Now all maximal points of S are known.

We process the nonmaximal points of S from left to right and make them known as follows. To process a point p, let $q_i, \ldots, q_j$ be all the maximal points that dominate p, which are all known. Observe that p must be in $W(q_i) \cup \cdots \cup W(q_j)$. Let L be all the points that are left of p, which are all known. We record the position of p in the list $(W(q_i) \cup \cdots \cup W(q_j)) - L$ of input elements (say, ordered by their indices). This requires $\lceil \log(|(W(q_i) \cup \cdots \cup W(q_j)) - L|\rceil$ bits. Observe that $W(q_i) \cup \cdots \cup W(q_j)$ is contained in $(-\infty, q_j.x) \times \mathbb{R}$. So, $(W(q_i) \cup \cdots \cup W(q_j)) - L$ is contained in the subset $F(p)$ from our Definition A.1—a lucky coincidence. Thus, the number of bits required is $\lceil \log |F(p)| \rceil$. Now p is known and we can continue the process.

By our construction, any permutation σ of S can be uniquely decoded from its encoding, for any given set S. The encoding has total length at most

$$T_A(\sigma) + h \log h + \sum_p \log |F(p)| + O(n) \;=\; T_A(\sigma) + h \log h + n \log n - \mathcal{F}(S) + O(n).$$

Taking the maximum over all permutations σ of S, we thus obtain $\log(n!) \;\leq\; T_A(S) + h \log h + n \log n - \mathcal{F}(S) + O(n)$, yielding $T_A(S) + n + h \log h \in \Omega(\mathcal{F}(S))$. Combined with the trivial lower bound $\Omega(n)$ and the naive information-theoretic lower bound $T_A(S) \in \Omega(h \log h)$ (as the problem definition requires the output to be in sorted order), this implies that $T_A(S) \in \Omega(\mathcal{F}(S) + n)$.

The proof works in the random-order setting as well: in any encoding scheme, at most a fraction 2^{-cn} of the $n!$ permutations can have encoding length less than $\log(n!) - cn$ for any constant c. Thus, for a random permutation σ, with high probability the encoding length is at least $\log(n!) - O(n)$, implying $T_A(\sigma) \in \Omega(\mathcal{F}(S) + n)$. In particular, $T_A^{\text{avg}}(S) \in \Omega(\mathcal{F}(S) + n)$. □

Combining the previous theorem with the following observation yields a complete proof of the $\Omega(\mathcal{F}(S) + n)$ lower bound:

Observation A.4. *Any algorithm for the 2D maxima problem in the comparison model can be made to solve the 2D "maxima with witnesses" problem without needing to make any additional comparisons.*

Proof. Consider the partial order $\prec_x$ over S formed by the outcomes of the x-comparisons made by the maxima algorithm A. Define the partial order $\prec_y$ similarly. Fix a nonmaximal point p. We argue that there must be a point $q \in S$ such that $p \prec_x q$ and $p \prec_y q$. Suppose that every $q \in S$ has $p \not\prec_x q$ or $p \not\prec_y q$. Consider modifying the point set as follows: Increase the x-coordinates of p and all points in $\{q \in S : p \prec_x q\}$ by a sufficiently large common value; this does not affect the outcomes of the comparisons made and ensures that all points q with $p \not\prec_x q$ now have $p.x > q.x$. Similarly, increase the y-coordinates of p and all points in $\{q \in S : p \prec_y q\}$ by a sufficiently large common value; all points q with $p \not\prec_y q$ now have $p.y > q.y$. Then every $q \in S$ now has $p.x > q.x$ or $p.y > q.y$; that is, p is now maximal, and the algorithm would be incorrect on the modified point set: a contradiction.

For every nonmaximal point p, we can thus find a witness point q that dominates p, without making any additional comparisons. One issue remains: the witness point may

not be maximal. If not, we can change p's witness to the witness of the witness, and so on, until p's witness is maximal.

Finally, note that we do not require the given algorithm A to report the maximal points in left-to-right order. We argue that at the end we already know the x-order of the maximal points. Suppose that $q \not\prec_x q'$ for two consecutive maximal points q and q'. Consider modifying the point set as follows: increase the x-coordinates of q and all points in $\{p \in S : q \prec_x p\}$ by a sufficiently large common value; this does not affect the outcomes of the comparisons made and ensures that we now have $q.x > q'.x$ (while maintaining $q.y > q'.y$). Then q' is now nonmaximal, and the algorithm would be incorrect on the modified point set: a contradiction. □

Remark A.5. The proof can be modified for weaker versions of the problem, for example, reporting just the number of maximal points (or its parity).

The proof does not appear to work for problems other than maxima in 2D. One obvious issue is that Observation A.4 only applies to comparison-based algorithms for orthogonal-type problems. Even more critically, the proof of Theorem A.3 relies on a coincidence that is special to 2D maxima.

Curiously, this lower-bound proof holds even for nondeterministic algorithms, that is, algorithms that can make guesses but must verify that the answer is correct; here we assume that each bit guessed costs unit time. In the proof of Theorem A.3, we just record the guesses in the encoding. The previous proofs of instance optimality by Fagin et al. [2003] and Demaine et al. [2000] all hold in nondeterministic settings. Perhaps this strength of the proof prevents its applicability to other geometric problems, whereas our adversary-based proofs more powerfully exploit the deterministic nature of the algorithms.

ACKNOWLEDGMENTS

We thank the referees for their helpful comments.

REFERENCES

Peyman Afshani and Timothy M. Chan. 2009. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*. 180–186.

Pankaj K. Agarwal and Jeff Erickson. 1999. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, B. Chazelle, J. E. Goodman, and R. Pollack (Eds.). Contemporary Mathematics, Vol. 223. American Mathematical Society, Providence, RI, 1–56.

Nir Ailon, Bernard Chazelle, Kenneth L. Clarkson, Ding Liu, Wolfgang Mulzer, and C. Seshadhri. 2011. Self-improving algorithms. *SIAM Journal on Computing* 40 (2011), 350–375.

D. S. Andrews, Jack Snoeyink, Jim Boritz, Timothy M. Chan, Graham Denham, Jenny Harrison, and Chong Zhu. 1994. Further comparisons of algorithms for geometric intersection problems. In *Proc. 6th International Symposium on Spatial Data Handling*. 709–724.

Sunil Arya, Theocharis Malamatos, and David M. Mount. 2007a. A simple entropy-based algorithm for planar point location. *ACM Transactions on Algorithms* 3, (2007), Article 17.

Sunil Arya, Theocharis Malamatos, David M. Mount, and Ka Chun Wong. 2007b. Optimal expected-case planar point location. *SIAM Journal on Computing* 37 (2007), 584–610.

Ilya Baran and Erik D. Demaine. 2005. Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. *International Journal of Computational Geometry and Applications* 15 (2005), 327–350.

Jérémy Barbay and Eric Chen. 2008. Adaptive planar convex hull algorithm for a set of convex hulls. In *Proc. 20th Canadian Conference on Computational Geometry*. 47–50.

Michael Ben-Or. 1983. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Symposium on Theory of Computing*. 80–86.

Jon Louis Bentley, Kenneth L. Clarkson, and David B. Levine. 1990. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*. 179–187.

Binay K. Bhattacharya and Sandeep Sen. 1997. On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm. *Journal of Algorithms* 25 (1997), 177–193.

Prosenjit Bose, Luc Devroye, Karim Douïeb, Vida Dujmović, James King, and Pat Morin. 2010. Odds-on trees. *arXiv* abs/1002.1092 (2010).

Joan Boyar and Lene M. Favrholdt. 2007. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms* 3, (2007), Article 22.

Timothy M. Chan. 1996a. Fixed-dimensional linear programming queries made easy. In *Proc. 12th ACM Symposium on Computational Geometry*. 284–290.

Timothy M. Chan. 1996b. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry* 16 (1996), 361–368.

Timothy M. Chan. 1996c. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete and Computational Geometry* 16 (1996), 369–387.

Timothy M. Chan. 2000. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM Journal on Computing.* 30 (2000), 561–575.

Timothy M. Chan. 2010. Comparison-based time–space lower bounds for selection. *ACM Transactions on Algorithms* 6, (2010), Article 26.

Timothy M. Chan. 2012. Optimal partition trees. *Discrete and Computational Geometry* 47 (2012), 661–690.

Timothy M. Chan, Jack Snoeyink, and Chee-Keng Yap. 1997. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete and Computational Geometry* 18 (1997), 433–454.

Timothy M. Chan and Konstantinos Tsakalidas. 2015. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. In *Proc. 31st Symposium on Computational Geometry*.

Bernard Chazelle. 1988. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing* 17 (1988), 427–462.

Bernard Chazelle. 1991. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry* 6 (1991), 485–524.

Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. 1994. Ray shooting in polygons using geodesic triangulations. *Algorithmica* 12 (1994), 54–68.

Bernard Chazelle, Leo J. Guibas, and D. T. Lee. 1985. The power of geometric duality. *BIT* 25 (1985), 76–90.

Bernard Chazelle and Jiří Matoušek. 1995. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry: Theory and Applications* 5 (1995), 27–32.

Kenneth L. Clarkson. 1994. More output-sensitive geometric algorithms. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*. 695–702.

Kenneth L. Clarkson. 1995. Las Vegas algorithms for linear and integer programming. *Journal of the ACM* 42 (1995), 488–499.

Kenneth L. Clarkson and Peter W. Shor. 1989. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry* 4 (1989), 387–421.

Sébastien Collette, Vida Dujmović, John Iacono, Stefan Langerman, and Pat Morin. 2012. Entropy, triangulation, and point location in planar subdivisions. *ACM Transactions on Algorithms* 8, (2012), Article 29.

Mark de Berg, Matthew Katz, A. Frank van der Stappen, and Jules Vleugels. 2002. Realistic input models for geometric algorithms. *Algorithmica* 34 (2002), 81–97.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. 1997. *Computational Geometry: Algorithms and Applications*. Springer-Verlag.

Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. 2009. The geometry of binary search trees. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*. 496–505.

Erik D. Demaine and Alejandro López-Ortiz. 2003. A linear lower bound on index size for text retrieval. *Journal of Algorithms* 48 (2003), 2–15.

Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. 2000. Adaptive set intersections, unions, and differences. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*. 743–752.

Peerapong Dhangwatnotai, Tim Roughgarden, and Qiqi Yan. 2015. Revenue maximization with a single sample. *Games and Economic Behavior* 91 (2015), 318–333.

Vida Dujmović, John Howat, and Pat Morin. 2012. Biased range trees. *Algorithmica* 62 (2012), 21–37.

Herbert Edelsbrunner. 1987. *Algorithms in Combinatorial Geometry*. Springer-Verlag.

Herbert Edelsbrunner and Weiping Shi. 1990. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM Journal on Computing.* 20 (1990), 259–269.

Jeff Erickson. 2005. Dense point sets have sparse Delaunay triangulations. *Discrete and Computational Geometry* 33 (2005), 83–115.

Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences* 66 (2003), 614–656.

Alexander Golynski. 2009. Cell probe lower bounds for succinct data structures. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*. 625–634.

John Iacono. 2004. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications* 29 (2004), 19–22.

Neil D. Jones. 1997. *Computability and Complexity: From a Programming Perspective*. MIT Press.

Jeff Kahn and Jeong Han Kim. 1995. Entropy and sorting. *Journal of Computer and System Sciences* 51 (1995), 390–399.

Claire Kenyon. 1996. Best-fit bin-packing with random order. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*. 359–364.

David G. Kirkpatrick. 1983. Optimal search in planar subdivisions. *SIAM Journal on Computing* 12 (1983), 28–35.

David G. Kirkpatrick and Raimund Seidel. 1985. Output-size sensitive algorithms for finding maximal vectors. In *Proc. 1st ACM Symposium on Computational Geometry*. 89–96.

David G. Kirkpatrick and Raimund Seidel. 1986. The ultimate planar convex hull algorithm? *SIAM Journal on Computing* 15 (1986), 287–299.

Jiří Matoušek. 1992. Efficient partition trees. *Discrete and Computational Geometry* 8 (1992), 315–334.

Jiří Matoušek. 2000. Derandomization in computational geometry. In *Handbook of Computational Geometry*, Jörg-Rüdiger Sack and Jorge Urrutia (Eds.). Elsevier Science Publishers B.V., North-Holland, Amsterdam, 559–595.

Jiří Matoušek, Janos Pach, Micha Sharir, Shmuel Sifrony, and Emo Welzl. 1994. Fat triangles determine linearly many holes. *SIAM Journal on Computing* 23 (1994), 154–169.

Shlomo Moran, Marc Snir, and Udi Manber. 1985. Applications of Ramsey's theorem to decision tree complexity. *Journal of the ACM* 32 (1985), 938–949.

Ketan Mulmuley. 1993. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ.

J. Ian Munro and Philip M. Spira. 1976. Sorting and searching in multisets. *SIAM Journal on Computing* 5 (1976), 1–8.

Cuong P. Nguyen. 2015. *Optimal Dominance Counting for a Non-Uniform Query Distribution in Linear Space*. Dalhousie University. Bachelor of Computer Science Honours Thesis.

Franco P. Preparata and Michael I. Shamos. 1985. *Computational Geometry: An Introduction*. Springer-Verlag.

Sandeep Sen and Neelima Gupta. 1999. Distribution-sensitive algorithms. *Nordic Journal on Computing* 6 (1999), 194–211.

Jack Snoeyink. 1997. Point location. In *Handbook of Discrete and Computational Geometry*, Jacob E. Goodman and Joseph O'Rourke (Eds.). CRC Press, Boca Raton, FL, Chapter 30, 559–574.

Daniel A. Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM* 51, 3 (2004), 385–463.

Rephael Wenger. 1997. Randomized quickhull. *Algorithmica* 17 (1997), 322–329.

Andrew Chi-Chih Yao. 1981. A lower bound to finding convex hulls. *Journal of the ACM* 28 (1981), 780–787.

Andrew Chi-Chih Yao. 1991. Lower bounds for algebraic computation trees with integer inputs. *SIAM Journal on Computing* 20 (1991), 655–668.

F. Frances Yao, David P. Dobkin, Herbert Edelsbrunner, and Michael S. Paterson. 1989. Partitioning space for range queries. *SIAM Journal on Computing* 18 (1989), 371–384.

Received April 2015; revised October 2016; accepted November 2016