Instance-Optimal Geometric Algorithms

Peyman Afshani* Jérémy Barbay† Timothy M. Chan‡

April 2, 2009

Abstract

We prove the existence of an algorithm A for computing 2-d or 3-d convex hulls that is optimal for *every point set* in the following sense: for every set S of n points and for every algorithm A' in a certain class $\mathcal{A}$, the maximum running time of A on input $\langle s_1, \ldots, s_n \rangle$ is at most a constant factor times the maximum running time of A' on $\langle s_1, \ldots, s_n \rangle$, where the maximum is taken over all permutations $\langle s_1, \ldots, s_n \rangle$ of S. In fact, we can establish a stronger property: for every S and A', the maximum running time of A is at most a constant factor times the average running time of A' over all permutations of S. 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 is 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 an interesting adversary argument. For 2-d 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 3-d convex hulls, we propose a new algorithm.

To demonstrate the potential of the concept, we further obtain instance-optimal results for a few other standard problems in computational geometry, such as maxima in 2-d and 3-d, orthogonal line segment intersection in 2-d, finding bichromatic L_∞-close pairs in 2-d, off-line orthogonal range searching in 2-d, off-line dominance reporting in 2-d and 3-d, off-line halfspace range reporting in 2-d and 3-d, and off-line point location in 2-d.

*Center for Massive Data Algorithmics (MADALGO), University of Aarhus, Aarhus, Denmark, peyman@madalgo.au.dk
†Departamento de Ciencias de la Computación (DCC), Universidad de Chile, Santiago, Chile, jeremy@dcc.uchile.cl
‡Cheriton School of Computer Science (CSCS), University of Waterloo, Waterloo, Canada, tmchan@uwaterloo.ca

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 towards *adaptive* analysis where the cost of algorithms is measured as a function of not just the input size but other parameters that capture in some ways the inherent simplicity or difficulty of the input instance. For example, for problems in computational geometry (the primary domain of the present paper), parameters that have been considered in the past include the output size (leading to so-called *output-sensitive* algorithms) [46], the spread of an input point set (the ratio of the maximum to the minimum pairwise distance) [38], various measures of fatness of the input objects (e.g., ratio of circumradii to inradii) [48] or clutteredness of a collection of objects [30], the number of reflex angles in an input polygon, and so on.

The ultimate in adaptive algorithms is an *instance-optimal* algorithm, i.e., 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 2-d 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' 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 below:

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 $T_A(S) \le O(1) \cdot \mathrm{OPT}(S)$ 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 above 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 [26].

For many algorithms (e.g., quickhull [52], to name one), the running time is not affected so much by the order in which the input points are given but by the input point set itself. 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 $T_A^{\mathrm{avg}}(S)$ denote the average running time of A on input σ over all $n!$ possible permutations σ of S. Let $\mathrm{OPT}^{\mathrm{avg}}(S)$ denote the minimum of $T_{A'}^{\mathrm{avg}}(S)$ over all correct

[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 [42], 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.

algorithms $A' \in \mathcal{A}$. If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \cdot \mathrm{OPT}^{\mathrm{avg}}(S)$ 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 above sense is immediately also competitive against *randomized* (Las Vegas) algorithms A', by the easy direction of Yao's principle. The above definition has extra appeal in computational geometry, as it is common to see the design of randomized algorithms where the input elements are initially permuted in random order [28].

Instance-optimal algorithms in the random-order setting also imply optimal *average-case* algorithms 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 instance-optimal algorithm can deal with all probability distributions at the same time! 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.

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 2-d and 3-d have been proposed since the 1970s [31, 36, 52], which are worst-case optimal under the algebraic computation tree model. Optimal output-sensitive algorithms can solve the 2-d and 3-d problem in $O(n \log h)$ time, where h is the output size. The first such output-sensitive algorithm in 2-d was found by Kirkpatrick and Seidel [46] in the 1980s and was later simplified by Chan, Snoeyink, and Yap [20] and independently Wenger [55]; a different, simple, optimal output-sensitive algorithm was discovered by Chan [15]. The first optimal output-sensitive algorithm in 3-d was obtained by Clarkson and Shor [28] using randomization; another version was described by Clarkson [26]. The first deterministic optimal output-sensitive algorithm in 3-d was obtained by Chazelle and Matoušek [25] via derandomization; the approach by Chan [15] can also be extended to 3-d and yields a simpler optimal output-sensitive algorithm. There are also average-case algorithms that run in $O(n)$ expected time for certain probability distributions [52], e.g., when the points are independent and uniformly distributed inside a circle or a constant-size polygon in 2-d, or a ball or a constant-size polyhedron in 3-d.

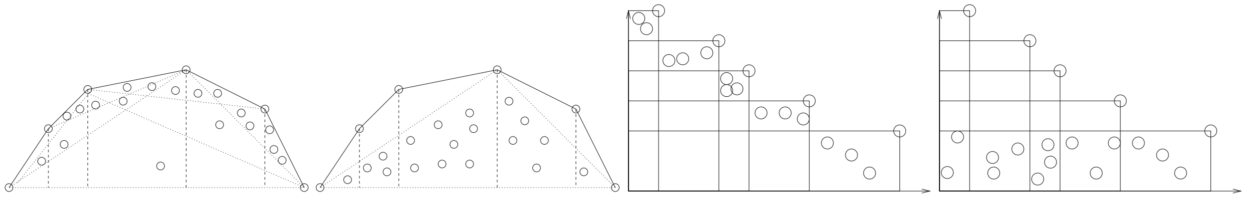
The convex hull problem is in some ways an ideal candidate to consider in our models. It is not difficult to think of examples of "easy" point sets and "hard" point sets (see Figure 1(a,b)). It is not difficult to think of different heuristics for pruning nonextreme points, which may not necessarily improve worst-case complexity but may help for many point sets encountered "in practice" (e.g., consider quickhull [52]). However, it is unclear whether there is a single pruning strategy that works best on all point sets.

In this paper, we show that there are indeed instance-optimal algorithms for both the 2-d and 3-d 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 2-d case, a new algorithm is not necessary: a version of Kirkpatrick and Seidel's output-sensitive algorithm, or its simplification by Chan, Snoeyink, and Yap, is instance-optimal in the order-oblivious and random-order setting. We view this as a plus: these algorithms

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



(a) (b) (c) (d)

Figure 1: (a) A "harder" point set and (b) an "easier" point set for the upper hull problem. (c) A point set with $\mathcal{H}(S)$ near $\log h$ and (d) an "easier" point set with $\mathcal{H}(S)$ near constant for the maxima problem.

are simple and practical to implement [12], and our analysis sheds new light into 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 also provides a more conclusive answer to the title question from Kirkpatrick and Seidel's paper.)

In 3-d we propose a new algorithm, as none of the previous output-sensitive algorithms seem to be instance-optimal (e.g., known 3-d generalizations of the Kirkpatrick–Seidel algorithm have suboptimal $O(n \log^2 h)$ running time [20, 37], while a straightforward implementation of Chan's algorithm [15] fails to be instance-optimal even in 2-d). Our algorithm builds on Chan's technique [15] but requires additional ideas, notably the use of *partition trees* [31, 49].

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: (i) information-theoretical or counting arguments, (ii) topological arguments, from early work by Yao [56] to Ben-Or's theorem [10], and (iii) Ramsey-theory-based arguments, by Moran, Snir, and Manber [50]. 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 for a decision tree model in which all the test functions have a bounded number of arguments. For an arbitrary input set S for the convex hull problem, the naive information-theoretical argument gives only an $\Omega(h \log h)$ lower bound on OPT(S). On the other hand, topological and Ramsey-theory approaches seem unable to give any instance-specific lower bound at all (e.g., modifying the topological approach is already nontrivial if we just want a lower bound for *some* integer input set [57], 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 which is inspired by an adversary argument from a recent work by Chan [18] on an unrelated problem (time–space lower bounds for median finding). Chan [19] noted that this approach can lead to another proof of the standard $\Omega(n \log n)$ lower bounds for many geometric problems including convex hull; the proof is simple and works in an 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. 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 is quite nice, in our opinion. Interestingly it involves partition trees, which are more typically used in algorithms (as in our new 3-d algorithm) rather than in lower-bound proofs.

So, what is OPT(S), i.e., what parameter truly captures the difficulty of a point set S, asymptotically, 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 (also called "expected-case") analysis of geometric data structures where query points come from a given probability distribution—these entropy-based results have been the subject of several recent papers [6, 7, 29, 35, 41]. However, lower bounds for expected-case data structures cannot be applied to our problem because our problem is off-line (lower bounds for on-line query problems usually assume that the query algorithms fit a "classification tree" framework, but an off-line algorithm may compare a query point not only with points from the data set but also with other query points). Furthermore, although in the off-line setting we can think of the query points as coming from a discrete point probability distribution, the distribution is not known in advance.[3] Lastly, expected-case data structures achieve speedup in querying but not preprocessing.

Other results. 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: (a) maxima in 2-d and 3-d, (b) reporting/counting intersection between horizontal and vertical line segments in 2-d, (c) reporting/counting pairs of L_∞-distance at most 1 between a red point set and a blue point set in 2-d, (d) off-line orthogonal range reporting/counting in 2-d, (e) off-line dominating reporting in 2-d and 3-d, (f) off-line halfspace range reporting in 2-d and 3-d, and (g) off-line point location in 2-d.

Optimal expected-case, entropy-based data structures for the on-line version of (g) are known before [7, 41], but not for (e,f)—for example, a recent SODA'09 paper by Dujmović, Howat, and Morin [35] only obtained results for 2-d dominance counting, a special case of 2-d orthogonal range counting. Incidentally, as a consequence of our ideas, we can also get new optimal expected-case data structures for on-line 2-d general orthogonal range counting and 2-d and 3-d halfspace range reporting.

Related work. Although Fagin *et al.* [39] first coined the term "instance optimality" (when studying the problem about finding items with the k top aggregate scores in a database in a certain model), 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 [32] for the latest in a series of papers). Demaine, López-Ortiz, and Munro [34] 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; see Barbay and Chen [9] for an extension to a 2-d problem on computing the convex hulls of k convex polygons. Another work about instance-optimal geometric algorithms is by Baran and Demaine [8], 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 on-line algorithms. In fact, in the on-line algorithms literature, our order-oblivious setting of instance optimality is related to what Boyar and Favrholdt called the *relative worst order ratio* [13], and our random-order setting is related to Kenyon's *random order ratio* [43]. What makes instance optimality more intriguing is that we are not bounding the objective function of an optimization problem but the cost of an algorithm.

2 Warm-Up: 2-d Maxima

Before proving our main result on convex hull, we find it useful to study a simpler problem: maxima in 2-d. For two points p and q we say p *dominates* q if each coordinate of p is greater than that 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

[3] *Self-improving* algorithms [4, 27] 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 can come from a different distribution, so input order matters.

other point in S. For simplicity, we assume that the input is always nondegenerate throughout the paper. The maxima problem is to report all maximal points, say, from left to right.

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 2-d, the boundary of the union of the orthants 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, Kirpatrick and Seidel's output-sensitive algorithm 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 [11, 26, 52]. The problem is simpler 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 will act as 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 OPT(S) is precisely $\Theta(n(\mathcal{H}(S) + 1))$ for the 2-d 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 $\mathcal{H}(\Pi) = \sum_{k=1}^{t} (|S_k|/n) \log(n/|S_k|)$. Define $\mathcal{H}(S)$ 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 analysis).

$\mathcal{H}(\Pi)$ is of course an entropy-like expression and is similar to bounds used in expected-case geometric data structures for the case of a discrete point probability distribution, although our definition itself is non-probabilistic. A measure proposed by Sen and Gupta [53] is identical to $\mathcal{H}(\Pi_{\text{vert}})$ for a specific respectful partition Π_{vert} of S, obtained by dividing the point set S by h vertical lines at the h maximal points of S. 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 Figure 1(d)).

The complexity of the 1-d multiset sorting problem [51] also has a similar expression, but there each input multiset induces a unique partition and so the situation is much simpler.

2.1 Upper bound

The algorithm we use is a slight variant of Kirkpatrick and Seidel's output-sensitive maxima algorithm [45] (in their original algorithm, only points from Q_ℓ are pruned in line 4):

maxima(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 (computable in linear time)
4. *prune* all points from Q_ℓ and Q_r that are dominated by q
5. return the concatenation of maxima(Q_ℓ) and maxima(Q_r)

We call maxima(S) to start. It is straightforward to show that the algorithm runs in time $O(n \log h)$, or $O(n(\mathcal{H}(\Pi_{\text{vert}}) + 1))$ time, as was done by Sen and Gupta [53]. Upper-bounding the running time by $O(n(\mathcal{H}(\Pi) + 1))$ for an *arbitrary* respectful partition Π of S requires a bit more finesse:

Theorem 2.3 *The above 2-d maxima algorithm runs in $O(n(\mathcal{H}(S)+1))$ time.*

Proof: Imagine the recursion tree being generated level by level from the root. Let X_j denote the sublist of all maximal points of S discovered during levels $0,\ldots,j$ of the recursion, in left-to-right order. Observe that (i) there can be at most $\lceil n/2^j \rceil$ points of S with x-coordinates between any two consecutive points in X_j, and (ii) all points that are strictly below the staircase of X_j have been pruned during levels $0,\ldots,j$ of the recursion. Let n_j be the number of points of S that *survive* level j, i.e., that have not been pruned during levels $0,\ldots,j$. The running time is asymptotically bounded by $\sum_{j=0}^{\log n} n_j$.

Let Π be *any* respectful partition of S. Look at a 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 the upper-right corner of B_k has x-coordinate between two consecutive points q_i, 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 $\min\{|S_k|, \lceil n/2^j \rceil\}$. Since the S_k's cover the entire point set, with a double summation we have

$$\sum_{j=0}^{\log n} n_j \le \sum_k \sum_{j=0}^{\log n} \min\{|S_k|, \lceil n/2^j \rceil\} \le \sum_k O\left(|S_k|\log(n/|S_k|) + |S_k| + |S_k|/2 + |S_k|/4 + \cdots\right)$$

$$= O\left(\sum_k |S_k|(\log(n/|S_k|)+1)\right) = O(n(\mathcal{H}(\Pi)+1)). \qquad \square$$

2.2 Lower bound

For the lower-bound side, we first provide an intuitive justification for the bound $n\mathcal{H}(S)$ 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 why it is not a maximal point, by indicating at least one *witness* point in S which 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 presents 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.1, 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 definition of $\mathcal{H}(S)$. Moreover, it is limited specifically to the 2-d maxima problem and does not extend to 3-d maxima, let alone to nonorthgonal problems like convex hull.

In this subsection, we describe instead a different proof, which generalizes more easily to the other problems that we consider. The proof is based on an interesting and simple *adversary* argument. For simplicity, we concentrate on the order-oblivious setting and postpone the modification of the proof in the random-order setting to Appendix A.2.

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

Proof: We use a *k-d tree* construction [31] to define a tree $\mathcal{T}$ of axis-aligned boxes, generated top-down as 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 (resp. even) depth, divide B into two subboxes by the median x-coordinate (resp. y-coordinate) among the points of S inside B.

The two subboxes are the children of B. Note that each box at depth j of $\mathcal{T}$ contains at least $\lfloor n/2^j \rfloor$ points of S.

Let $\Pi_{\text{kd-tree}}$ be the partition of S formed by the leaf boxes in this tree $\mathcal{T}$ (i.e., points in the same leaf box are placed in the same subset). Clearly, this partition $\Pi_{\text{kd-tree}}$ is respectful. We will prove that for any correct algorithm in $\mathcal{A}$, there exists a permutation of S on which the algorithm requires at least $\Omega(n\mathcal{H}(\Pi_{\text{kd-tree}}))$ comparisons.

The adversary constructs a bad permutation by simulating the algorithm on an initially unknown input. During the simulation, we maintain a box B_p in $\mathcal{T}$ for each point p. If B_p is a leaf, the algorithm knows the exact location of p inside B_p. But if B_p is an internal node, the only information the algorithm knows about p is that p lies inside B_p. In other words, p can be assigned any point in B_p without affecting the outcome of the previous comparisons made.

For each box B in $\mathcal{T}$, let $n(B)$ be the number of points p with B_p 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*. Whenever B_p first becomes a leaf, we fix p by assigning it to an arbitrary point in $S \cap B_p$ that has previously not been assigned. The invariant ensures that such an assignment can always be made.

When the simulation encounters a comparison, say, of the x-coordinates, between two points p and q, we do the following:

1. If B_p (resp. B_q) is at even depth, we reset B_p (resp. B_q) to one of its children arbitrarily. Now we may assume that B_p and B_q are both at odd depths (if they are not leaves).

   W.l.o.g., suppose that the median x-coordinate of B_p is less than the median x-coordinate of B_q. We reset B_p to the left child B'_p of B_p and B_q to the right child B'_q of B_q. Now, the knowledge that p and q lie in B'_p and B'_q allows us to deduce that p has a smaller x-coordinate than q, so we can resolve the comparison and continue the simulation.

2. An exceptional case occurs if B'_p is full (or similarly B'_q is full). Here, we reset B_p instead to the sibling B''_p of B'_p. The invariant is maintained, since $|S \cap B_p| \geq n(B_p) \geq n(B'_p) + n(B''_p) + 1$ implies that B'_p and B''_p cannot both be full. The comparison is not necessarily resolved yet, so we go back to step 1.

The above description ignores the case when B_p is a leaf (or similarly B_q is a leaf). This case can be treated in the same way, except that in step 1, since p has been fixed, we compare the actual x-coordinate of p to the median x-coordinate of B_q, and reset only B_q. (If both B_p and B_q are leaves, the comparison is already resolved.)

Let T be the number of comparisons made. Let D be the sum of the depth of B_p over all points $p \in S$. We will bound D in terms of T. Each time we reset a box to one of its children in step 1 or 2, D increments; we say that an *ordinary* (resp. *exceptional*) increment occurs at the parent box if this is done in step 1 (resp. step 2). Each comparison generates only $O(1)$ ordinary increments. To take exceptional increments into account, we use a little 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 exceptional increments can occur, and the number of exceptional increments is at most $\rceil |S \cap B|/2 \rceil$. Thus, the total number of exceptional increments is asymptotically at most the total number of ordinary increments, which is $O(T)$. It follows that $D = O(T)$, i.e., $T = \Omega(D)$.

After the end of the simulation, we can do the following postprocessing: whenever there is an internal node B_p, we reset B_p to one of its non-full children arbitrarily, and repeat. As a result, every B_p becomes a leaf, and all the input points have been assigned to points of S, and no two input points assigned are the same value, i.e., the input is fixed to a permutation of S.

We claim that the postprocessing is actually unnecessary, i.e., every B_p is already a leaf by the end of the simulation. If not, B_p contains at least two points and is not completely underneath the staircase of S. We can either move a nonmaximal point upward or a maximal point downward inside B_p and obtain a different input that is consistent with the comparisons made but has a different set of maximal points. The algorithm would be incorrect on this input: a contradiction.

Thus, at the end of the simulation, each B_p has depth $\Theta(\log(n/|S \cap B_p|))$. It follows that

$$T \;=\; \Omega(D) \;=\; \Omega\left(\sum_{\text{leaf } B} |S \cap B| \log(n/|S \cap B|)\right) \;=\; \Omega(n\mathcal{H}(\Pi_{\text{kd-tree}})) \;=\; \Omega(n\mathcal{H}(S)).$$

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

Remark 2.5 The above proof is inspired by an adversary argument by Chan [18] for a 1-d problem (the original proof maintains a dyadic interval for each input point, while the new proof maintains a box from a hierarchical subdivision). The proof still holds for weaker versions of the problem, e.g., where we can report the maxima in any order, or we just want the number of maximal points (or the parity of the number). The lower-bound proof easily extends to any fixed dimension and can be easily modified to allow comparisons of different coordinates of any two points $p = (x_1, \ldots, x_d)$ and $q = (x'_1, \ldots, x'_d)$, e.g., 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 2-d and 3-d 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 $f : (\mathbb{R}^d)^c \to \mathbb{R}^d$ is *multilinear* if the restriction of f is a linear function from $\mathbb{R}^d$ to $\mathbb{R}^d$ 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 2-d and 3-d convex hull algorithms we know of fit this framework. For example, 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 2-d we can compare the slope of the line through p_1, p_2 and the slope of the line through p_3, p_4 by 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. (See Appendix A.3, however, for situations where non-multilinear test functions arise.)

We adopt the following modified definition of $\mathcal{H}(S)$ (as before, it will not matter whether we insist that the simplices Δ_k below are nonoverlapping):

Definition 3.2 A partition Π of S is *respectful* if each subset S_k in Π is either a singleton or can be enclosed by a simplex Δ_k whose interior is completely below the upper hull of S. Define $\mathcal{H}(S)$ 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 Upper bound in 2-d

In 2-d, the algorithm we use is a version of Kirkpatrick and Seidel's output-sensitive upper hull algorithm [46], with obvious pruning step (line 2) added.

hull(Q):
1. if $|Q| = 2$ then return Q
2. *prune* all points from Q strictly below the line through the leftmost and rightmost point 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
5. *prune* all points from Q_ℓ and Q_r that are strictly underneath the line segment $\overline{qq'}$
6. return the concatenation of hull(Q_ℓ) and hull(Q_r)

Line 4 can be done in $O(n)$ time (without knowing the upper hull beforehand) by applying a 2-d linear programming algorithm in the dual [52]. We call hull(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 [53]. 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 in Theorem 2.3:

Theorem 3.3 *The above 2-d upper hull algorithm 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. Observe that (i) there can be at most $\lceil n/2^j \rceil$ points of S 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 the first j levels of the recursion.

Let Π be any respectful partition of S. Look at 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, 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_i that survive level j is at most $\min\left\{|S_k|, O(n/2^j)\right\}$ by (i). As before, we can then bound the running time asymptotically by $\sum_k \sum_{j=0}^{\log n} \min\left\{|S_k|, n/2^j\right\} = O(n(\mathcal{H}(\Pi) + 1))$. □

Remark 3.4 The same result holds for Chan, Snoeyink, and Yap's simplified output-sensitive algorithm, which avoids calling a 2-d linear programming algorithm. (In fact, Chan *et al.*'s paper explicitly adds the pruning step in their description, inspired by quickhull.) The only difference in the above analysis is (i): now, 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.

3.2 Lower bound

The lower-bound proof for convex hull builds on the proof for maxima from Section 2.2 but is more involved, because a k-d tree construction no longer suffices when addressing nonorthogonal problems. However, known tools in computational geometry provide an appropriate analog:

Lemma 3.5 *For every set Q of n points in $\mathbb{R}^d$ 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 of size $O(\text{polylog}\, r)$, such that Q_i is contained in γ_i, and each hyperplane intersects at most $O(r^{1-\varepsilon})$ cells. Here, $\varepsilon > 0$ is a constant that depends only on d.*

Proof: (Option 1) We can use Matoušek's partition theorem [49], which provides the best constant, namely, $\varepsilon = 1/d$. Each cell γ_i is a simplex but the cells may overlap. (Note that our application requires that the subset sizes are lower-bounded by $\Omega(n/r)$, which is guaranteed by Matoušek's construction.)

(Option 2) For $d = 2$ or 3, a more elementary solution follows from the 4-sectioning or 8-sectioning theorem [36, 58]: for every n-point set Q in $\mathbb{R}^2$, there exist 2 lines that divide the plane into 4 regions each with $n/4$ points; for every n-point set Q in $\mathbb{R}^3$, there exist 3 planes that divide space into 8 regions each with $n/8$ points. Since in $\mathbb{R}^2$ a line can intersect at most 3 of the 4 regions and in $\mathbb{R}^3$ a plane can intersect at most 7 of the 8 regions, a simple recursive application of the theorem yields $\varepsilon \approx 1 - \log_4 3$ for $d = 2$ and $\varepsilon \approx 1 - \log_8 7$ for $d = 3$. Each resulting cell γ_i may have $O(\log r)$ facets, but the cells do not overlap. $\square$

We need another fact, this time, a straightforward geometric property about multilinear functions:

Lemma 3.6 *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 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 $at_i t_j + a' t_i + a'' t_j + a''' = 0$ 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. $\square$

We are now ready for the main proof. Again, we focus on the order-oblivious setting and leave the random-order setting to Appendix A.2.

Theorem 3.7 $\mathrm{OPT}(S) = \Omega(n(\mathcal{H}(S) + 1))$ *for the upper hull problem 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 $\gamma(v)$. For each node v, let $\Gamma(v)$ denote the intersection of $\gamma(u)$ over all ancestors u of 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, fix a parameter $r = b$ and partition $Q(v)$ by Lemma 3.5 to get b subsets $Q_1, \ldots, Q_b$ and cells $\gamma_1, \ldots, \gamma_b$. The pairs (Q_i, γ_i) are the children of v. Note that for each node v at depth j of the tree $\mathcal{T}$, $|Q(v)| \ge n/\Theta(b)^j$. Note also that $\Gamma(v)$ has $O(bj)$ facets.

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 $(bj)^{O(1)}$ 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 decrease by at most $O((|Q(v)|/n)\log(bj)^{O(1)}) = O((|Q(v)|/n)\log\log(n/|Q(v)|))$ for a constant b. The total decrease in entropy is thus $o(\mathcal{H}(\Pi_{\text{part-tree}}))$. So $\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}}) = \Theta(\mathcal{H}(\Pi_{\text{part-tree}}))$. Clearly, $\widetilde{\Pi}_{\text{part-tree}}$ is respectful.

The adversary constructs a bad input as follows. During the simulation, we maintain a node v_p in $\mathcal{T}$ for each point p. If v_p is a leaf, the algorithm knows the exact location of p inside $\Gamma(v_p)$. But if v_p is an internal node, the only information the algorithm knows about p is that p lies inside $\Gamma(v_p)$.

For each node v in $\mathcal{T}$, let $n(v)$ be the number of points p with v_p in the subtree rooted at v. We maintain that $n(v) \le |Q(v)|$. If $n(v) = |Q(v)|$, we say that v is *full*. When v_p first becomes a leaf, we fix p to an arbitrary unassigned point in $Q(v_p)$. The invariant ensures that such an assignment can always be made.

Suppose the simulation encounters a test "$f(p_1, \ldots, p_c) > 0$?".

1. Consider a c-tuple $(v'_{p_1}, \ldots, v'_{p_c})$ where v'_{p_i} is a child of v_{p_i}. We say that the tuple is *bad* if f has a zero in $\gamma(v'_{p_1}) \times \cdots \times \gamma(v'_{p_c})$, and *good* otherwise. We count the number of bad tuples: If we fix all but one point p_i, the restriction of f can have a zero in at most $O(b^{1-\varepsilon})$ cells of the form $\gamma(v'_{p_i})$, by Lemma 3.5 and the multi-linearity of f. There are $O(b^{c-1} \operatorname{polylog} b)$ choices of $c-1$ vertices of the cells of the form $\gamma(v'_{p_1}), \ldots, \gamma(v'_{p_c})$. By Lemma 3.6, it follows that the number of bad tuples is at most $O(b^{c-1} \operatorname{polylog} b \cdot b^{1-\varepsilon}) = O(b^{c-\varepsilon} \operatorname{polylog} b)$. The overall number of tuples is $\Theta(b^c)$. So, by choosing b to be a sufficiently large constant, we can guarantee that some tuple $(v'_{p_1}, \ldots, v'_{p_c})$ is good. We reset v_{p_i} to v'_{p_i} for each $i = 1, \ldots, c$. Since the tuple is good, the sign of f is determined and the comparison is resolved.

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

The above description can be easily modified in the case when some of the nodes v_{p_i} are leaves, i.e., when some of the points p_i are already fixed (we just have to lower c by the number of fixed points).

Let T be the number of tests made. Let D be the sum of the depth of v_p over all points $p \in S$. By the same amortization argument as before (after adjustments of constant factors), we can lower-bound T by $\Omega(D)$.

After the end of the simulation, we can do the following postprocessing: whenever there is an internal node v_p, we reset v_p to one of its non-full children arbitrarily, and repeat. This way, the input is completely fixed to a permutation of S. We claim that the postprocessing is unnecessary, i.e., every v_p is already a leaf at the end of the simulation. If not, we can either move a interior point upward or an extreme point downward inside $\Gamma(v_p)$ and change the upper hull. Then the algorithm would be incorrect on the modified input: a contradiction.

Thus, at the end of the simulation, each node v_p has depth $\Theta(\log(n/|Q(v_p)|))$. It follows that $T = \Omega(D) = \Omega(\sum_{\text{leaf } v} |Q(v)| \log(n/|Q(v)|)) = \Omega(n\mathcal{H}(\Pi_{\text{part-tree}})) = \Omega(n\mathcal{H}(\widetilde{\Pi}_{\text{part-tree}})) = \Omega(n\mathcal{H}(S))$. Combined with the trivial $\Omega(n)$ lower bound, this establishes the theorem. □

3.3 Upper bound in 3-d

The preceding lower-bound proof holds in any fixed dimension. We now give a 3-d upper-hull algorithm that matches the bound. Unlike in 2-d, it is unclear if any of the known algorithms can be modified for this purpose. For example, it is already nontrivial how to get an $O(n\mathcal{H}(\Pi_{\text{vert}}))$ upper bound for the specific partition Π_{vert} where points underneath the same upper-hull facet are placed in the same subset. Fortunately, informed by our lower-bound proof, we discover a solution based on partition trees.

We need the following subroutine by Chan [15, 16], which is obtained by applying a simple grouping trick in conjunction with standard data structures (see also [14]).

Lemma 3.8 *We can answer a sequence of r linear programming queries over a given set of n halfspaces in $\mathbb{R}^3$ in total time $O(n \log r + r \log n)$.*

Our new upper hull algorithm is as follows:

```
1. Q ← S
2. for j = 0, 1, ..., ⌈log(δ log n)⌉ do
3.     partition Q by Lemma 3.5 to get r_j := 2^{2^j} subsets Q_1, ..., Q_{r_j} and cells γ_1, ..., γ_{r_j}
4.     for each i do
5.         if γ_i is strictly below the upper hull of Q then prune all points in Q_i from Q
6. return the upper hull of the remaining set Q
```

Line 3 takes $O(|Q|\log r_j)$ time by known algorithms for Lemma 3.5 (either option) [49]. The test in line 5 reduces to deciding whether each vertex of γ_i is strictly below the upper hull of Q. This can be done (without knowing the upper hull beforehand) by answering a 3-d linear programming query in the dual. Using Lemma 3.8, we can perform lines 4–6 collectively in time $O(|Q|\log r_j + r_j \operatorname{polylog} r_j \log n)$; note that $\sum_j r_j = O(n^\delta)$, and so the second term is negligible by choosing a constant $\delta < 1$. Line 6 is done by running any $O(|Q|\log|Q|)$-time algorithm; note that $\log|Q| = O(\log r_j)$ in the last iteration.

Theorem 3.9 *The above 3-d upper hull algorithm runs in $O(n(\mathcal{H}(S)+1))$ time.*

Proof: Let n_j be the size of Q just after iteration j. The total running time is asymptotically bounded by $\sum_j n_{j-1}\log r_j$.

Let Π be any respectful partition of S. Look at a 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 competely inside Δ_k, then all points inside γ_i are pruned. At most $O(r_j^{1-\varepsilon})$ cells γ_i intersect the boundary of Δ_k. Hence, the number of points in S_k that remain in Q after iteration j is at most $\min\left\{|S_k|, O(r_j^{1-\varepsilon}\cdot n/r_j^{\varepsilon})\right\} = \min\left\{|S_k|, O(n/r_j^{\varepsilon})\right\}$. Since the S_k's cover the entire point set, with a double summation we have

$$\begin{aligned}\sum_j n_j \log r_{j+1} &\le \sum_k \sum_j \min\left\{O(2^j)|S_k|,\ n/2^{\Omega(\varepsilon 2^j)}\right\}\\ &= O\left(\sum_k |S_k|(\log(n/|S_k|)+1)\right) = O(n(\mathcal{H}(\Pi)+1)).\end{aligned}$$

□

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

One minor technicality is that the above description of the algorithm does not discuss the low-level test functions involved. In Appendix A.3, we explain how a modification of the algorithm can indeed be implemented in the multilinear model.

The same approach works for 3-d maxima as well. In the comparison model, the partitions can be constructed by a k-d tree construction, and linear programming queries are replaced by queries to test whether a point lies underneath the staircase, which can be done via an analog of Lemma 3.8.

4 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. Off-line halfspace range reporting in 2-d and 3-d: given a set S of n points and halfspaces, report the subset of points inside each halfspace. Algorithms with $\Theta(n\log n + K)$ running time [1, 17, 24] are known for total output size K (the 3-d algorithm is randomized).

2. Off-line dominance reporting in 2-d and 3-d: given a set S of red/blue points, report the subset of red points dominated by each blue point. The problem has similar complexity as in item 1.

3. Orthogonal segment intersection in 2-d: 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 [31, 52].

4. Bichromatic L_∞-close pairs in 2-d: given a set S of n red/blue points in 2-d, 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 [31, 52] yield algorithms with the same complexity as in item 3.

5. Off-line orthogonal range searching in 2-d: 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 item 3.

6. Off-line point location in 2-d: 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 [31, 52, 54] imply a worst-case running time of $\Theta(n \log n)$.

For each of the above problems, it is not difficult to see that certain input sets are indeed "easier" than others, e.g., 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.

Note that although some of the above 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.

4.1 Reporting problems

Many of the problems listed above belong to the following common 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. Note that by scanning the output pairs, we can collect the subset of all blue points that interact with each red point, in $O(K)$ additional time.

We say that a red (resp. blue) cell γ is *uninteresting to* S if every red (resp. blue) point in γ interacts with exactly the same subset of blue (resp. red) points in S. We redefine $\mathcal{H}(S)$ as follows:

Definition 4.1 A partition Π of S is *respectful* if each subset S_k in Π either is a singleton or is a monochromatic subset of points that can be enclosed by a simplex Δ_k that is uninteresting to S. Define $\mathcal{H}(S)$ to be the minimum of $\mathcal{H}(\Pi) := \sum_k (|S_k|/n) \log(n/|S_k|)$ over all respectful partitions Π of S.

It is straightforward to modify the proofs from Section 3.2 and Section A.2 to show an OPT(S), OPT$^{\text{avg}}(S) = \Omega(n(\mathcal{H}(S) + 1) + K)$ lower bound for this problem: We now keep two partition trees, one for each color. If $\Gamma(v)$ is uninteresting to S, we make v a leaf. At the end, if some red (resp. blue) node v_p is not a leaf, we can move p to some point inside $\Gamma(v_p)$ and change the answer. (The $\Omega(K)$ term in the lower bound is obvious, by the way.)

For the upper-bound side, we need three requirements about $\mathcal{R}$ for some constant $\alpha > 0$:

(A) There is a worst-case algorithm for the reporting problem that runs in $O(n \log n + K)$ time.

(B) There is a data structure for the blue (resp. red) points in S, with $O(n \log n)$ preprocessing time, such that we can report all κ blue (resp. red) points interacting with a query red (resp. blue) point in $O(n^{1-\alpha} + \kappa)$ time.

(C) There is a data structure for the blue (resp. red) points in S, with $O(n \log n)$ preprocessing time, such that we can test whether a query red (resp. blue) simplex γ is uninteresting to S in $O(n^{1-\alpha})$ time.

Under these assumptions, it is straightforward to modify the algorithm from Section **??** to an $O(n(\mathcal{H}(S)+1) + K)$-time algorithm: In line 3, we partition the red points of Q first. In line 5, if some red cell γ_i is uninteresting to Q, then we find the subset Z of blue points interacting with an arbitrary red point in γ_i, output all pairs between the red points of Q_i and the blue points of Z, and prune the red points of Q_i from Q. The test requires querying the data structure in (C) (after triangulating γ_i); the subset Z can be found by querying the data structure in (B). The grouping technique by Chan [16] yields an analog of Lemma 3.8 with running time $O(n \log r + rn^{1-\alpha})$ for r queries of type (C), and $O(n \log r + rn^{1-\alpha} + \kappa)$ for r queries of type (B) with total output size κ (since the problems in (B) and (C) are "decomposable"). Before moving to the next iteration, we redo lines 3–5, this time partitioning the blue points of Q and pruning red points. At the end, in line 6, we switch to the algorithm in (A). The same analysis then goes through, by choosing a constant $\delta < \alpha$.

Note that for orthogonal problems in the comparison model, we can make all the cells (all the γ's and Δ's) axis-aligned boxes, by reverting to a k-d tree construction.

We now check that the requirements are satisfied for some specific reporting problems.

- Off-line halfspace range reporting in 2-d and 3-d: It suffices to consider lower halfspaces in the input. Color the given points red, and map the given lower halfspaces to blue points by duality. The data structure problem in (B) is just halfspace range reporting. The data structure problem in (C) is equivalent to testing whether a query simplex intersects a given set of hyperplanes (lines in 2-d or planes in 3-d); this reduces to ray shooting (or segment emptiness) queries in a hyperplane arrangement, for which there are known results [3, 49]. Requirement (A) is satisfied in 2-d and 3-d (the 3-d algorithm is randomized).

- Off-line dominance reporting in 2-d and 3-d: 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 point set. This reduces to orthogonal range counting [2, 31, 52].

- Orthogonal segment intersection in 2-d: 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 $\overline{(x, y')(x, y'')}$ to a blue point $(x, y', y'') \in \mathbb{R}^3$. These mappings to $\mathbb{R}^3$ are bijective. The data structure problem in (B) corresponds to reporting the vertical segments from a given set that intersect a query horizontal segment. The data structure problem in (C) is more complicated: for a query box $\gamma = [\xi_1, \xi_2] \times [\xi_3, \xi_4] \times [\xi_5, \xi_6]$, we want to decide whether there exists a horizontal segment $\overline{(x', y)(x'', y)}$ with $(x', x'', y) \in \gamma$ that intersects a given set of vertical segments. This is equivalent to testing whether a query rectangle $[\min\{\xi_1, \xi_3\}, \max\{\xi_2, \xi_4\}] \times [\xi_5, \xi_6]$ intersects a given set of vertical segments. Both data structure problems reduce to orthogonal intersection searching (which in turn reduces to orthogonal range searching by lifting to a higher dimension, and thus admits data structures with $O(n \log n)$ preprocessing time and $O(n^\varepsilon)$ query time). Clearly, the resulting algorithm works in the comparison model.

- Bichromatic L_∞-close pairs in 2-d: The problem in (B) corresponds to reporting all points of a given point set that are inside a query square of side length 2. The problem in (C) corresponds to deciding, for a query box $\gamma = [\xi_1, \xi_2] \times [\xi_3, \xi_4]$, whether $[\xi_1 - 1, \xi_2 + 1] \times [\xi_3 - 1, \xi_4 + 1]$ contains a point from a given set. Both data structure problems reduce to orthogonal range searching.

Note that here the resulting algorithm requires slightly more general tests of the form mentioned in Remark 2.5, which are allowed in the lower-bound proof.

- Off-line orthogonal range reporting in 2-d: Color the given points red, and map each rectangle with corners $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$ to a blue point $(x_1, x_2, y_1, y_2) \in \mathbb{R}^4$. The mapping to $\mathbb{R}^4$ is bijective. The blue data structure problem in (B) corresponds to reporting all points from a given set that are inside a query rectangle. The red data structure problem in (B) corresponds to reporting all rectangles from a given set that contain a query point.

  The red data structure problem in (C) corresponds to deciding, for a query box $\gamma = [\xi_1, \xi_2] \times [\xi_3, \xi_4] \times [\xi_5, \xi_6] \times [\xi_7, \xi_8]$, whether all rectangles with corners $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$, $(x_1, x_2, y_1, y_2) \in \gamma$, contain the same number of points from a given set. This is equivalent to testing whether the rectangle $[\min\{\xi_1, \xi_3\}, \max\{\xi_2, \xi_4\}] \times [\min\{\xi_5, \xi_7\}, \max\{\xi_6, \xi_8\}]$ contains the same number of points from a given set as the rectangle $[\max\{\xi_1, \xi_3\}, \min\{\xi_2, \xi_4\}] \times [\max\{\xi_5, \xi_7\}, \min\{\xi_6, \xi_8\}]$.

  The blue data structure problem in (C) corresponds to deciding, for a query point $\gamma = [\xi_1, \xi_2] \times [\xi_3, \xi_4]$, whether γ intersects any rectangle from a given set.

  All these data structure problems reduce to orthogonal range/intersection searching.

4.2 Counting problems

We can also consider counting problems where we want the total number of interacting red/blue pairs. We just need to change requirement (A) to the existence of a counting algorithm that runs in $O(n \log n)$ time, and requirement (B) to the existence of a similar counting data structure without the $O(\kappa)$-term penalty. These requirements are satisfied by orthogonal segment intersection counting, bichromatic L_∞-close pairs, and off-line orthogonal range counting. The same lower- and upper-bound proofs yields $\Theta(n(\mathcal{H}(S) + 1))$.

If we want individual counts, i.e., the number of red points that interact with each blue point, we need a further assumption—that the data structure in (B) can operate in the semigroup model [2]. (This assumption is true for the specific problems mentioned in the preceding paragraph.) This way, we can report all interacting red/blue pairs as a disjoint union of bicliques $P_i \times Q_i$ with total sizes $\sum_i (|P_i| + |Q_i|)$ bounded by $O(n(\mathcal{H}(S) + 1))$, without the $O(K)$-term penalty. We can keep a counter for each blue point, scan through each biclique, and add the number of red points in the biclique to the counter of each blue point in the biclique, in total additional time $O(\sum_i (|P_i| + |Q_i|))$, which is absorbed in the overall cost. We assume that the algorithm in requirement (A) can produce individual counts but does not need to be the semigroup model. At the end (line 6), we can add the individual counts produced by this algorithm to the corresponding counters of each blue point.

4.3 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 "uninteresting": a red (resp. blue) cell γ is now considered *uninteresting to S* if no red (resp. blue) point in γ interacts with any blue (resp. red) points in S. We change requirements (A) and (B) to the existence of counting 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 the proof of the lower bound $\Omega(n(\mathcal{H}(S) + 1))$ only goes through for instances with a NO answer: at the end, if some red (resp. blue) node v_p is not a leaf, we can move p to some point inside $\Gamma(v_p)$ and change the answer from NO to YES.

YES instances are problematic, but this is not a weakness of our technique but of the model: on every input set S with a YES answer, 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.

4.4 More off-line/on-line querying problems

We now study problems from another framework. Let $\mathcal{M}$ be a mapping from points in $\mathbb{R}^d$ to "answers" in some space; 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 *off-line querying* problem: given a set S of n points in $\mathbb{R}^d$, compute $\mathcal{M}(q)$ for every $q \in S$. In addition, we consider the following *on-line querying* problem: given a set S of n points in $\mathbb{R}^d$, build a data structure for S 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 $q \in S$.

We redefine $\mathcal{H}(S)$ by redefining "uninteresting": a cell γ is now considered *uninteresting to* $\mathcal{M}$ if every point q in γ has the same answer $\mathcal{M}(q)$.

For the off-line problem, our lower-bound proof gives $\Omega(n(\mathcal{H}(S) + 1))$ even if $\mathcal{M}$ has been preprocessed in advance. For the on-line problem, the same proof shows that running a sequence of n queries over some permutation of S requires $\Omega(n(\mathcal{H}(S)+1))$ time, even if the set S (not the permutation) has been preprocessed in advance. So, the average query time is $\Omega(\mathcal{H}(S) + 1)$. (In contrast, lower bounds for the on-line problem do not necessarily translate to lower bounds for the off-line problem.)

For the upper-bound side, we need two requirements about $\mathcal{M}$ for some constant $\alpha > 0$ and some parameter m describing the size of $\mathcal{M}$. We assume that $\mathcal{M}$ has been preprocessed in some data structure.

(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.

(C) Given a simplex γ, we can test whether γ is uninteresting to $\mathcal{M}$ in $O(m^{1-\alpha})$ time.

The algorithm this time is actually simpler, because there is only one color. Instead of using a 2^{2^j} progression, we use a straightforward b-way recursion, for some fixed parameter b (the resulting recursion tree mimics the tree $\mathcal{T}$ from the lower-bound proof in Theorem 3.7, on purpose):

```
off-line-queries(Q, Γ), where Q ⊂ Γ:
1.  if |Q| drops below n/m^δ then return answers directly
2.  partition Q by Lemma 3.5 to get b subsets Q_1, ..., Q_b and cells γ_1, ..., γ_b
3.      for each i do
4.          if γ_i ∩ Γ is uninteresting to M then
5.              compute M(q) for an arbitrary point q ∈ γ_i ∩ Γ
6.              output M(q) as the answer for the points in Q_i
7.          else off-line-queries(Q_i, γ_i ∩ Γ)
```

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 switching to 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 [49]. Line 4 takes $O(m^{1-\alpha} \operatorname{polylog} m)$ time ($\gamma_i \cap \Gamma$ has $O(\operatorname{polylog} m)$ vertices), by (C); this cost is negligible by choosing a sufficiently small constant $\delta < \alpha$, since the recursion tree has $O(m^{\delta})$ nodes. Line 5 takes $O(\log m + \kappa)$ time for output size κ, by (A); the $O(\log m)$ term is again negligible.

For the on-line problem, we just build a data structure corresponding to the recursion tree generated above, in addition to the data structure for (A); the extra space is $O(m^{\delta})$.

Theorem 4.2 *The above off-line querying algorithm runs in $O(n(\mathcal{H}(S)+1)+K)$ time for total output size K. For the on-line querying problem, it produces a data structure that has average query cost $O(\mathcal{H}(S)+1+\kappa)$ for output size κ.*

Proof: Let n_j be number of points in S that *survive* level j, i.e., participate in subsets Q at level j of the recursion. The total running time for the off-line problem is asymptotically bounded by $\sum_j n_j$. Similarly, for the on-line problem, the total query cost over all $q \in S$ is asymptotically bounded by $\sum_j n_j$.

Let Π be any respectful partition of S. Look at a subset S_k in Π. Let Δ_k be a simplex enclosing S_k that is contained inside one face of M. Fix a level j. Let Q_i's and γ_i's 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_i's that intersect each side 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 $\min\left\{|S_k|, O(b^{1-\varepsilon})^j \cdot n/\Theta(b)^j\right\}$. Since the S_k's cover the entire point set, with a double summation we have, for a sufficiently large constant b,

$$\sum_j n_j \;\le\; \sum_k \sum_j \min\left\{|S_k|, \lceil n/\Theta(b)^{\varepsilon j}\rceil\right\} \;=\; O\left(\sum_k |S_k|(\log(n/|S_k|)+1)\right) \;=\; O(n(\mathcal{H}(\Pi)+1)).$$

□

For the on-line problem, the above approach works, after straightforward modifications, for weighted point sets S where we want to minimize the weighted average query cost. In principle, the approach works not only for discrete point sets S but also for continuous probability distributions, since the query bound does not depend on the size n of S explicitly and can be imagined to approach infinity. "Average query cost" over a finite set of query points now becomes "expected query cost" over a query point distribution. (The preprocessing time can also be made independent of n, under some computational assumptions about the distribution.)

Below, we briefly mention applications to some specific off-line/on-line querying problems.

- Off-line/on-line point location queries in 2-d: For the off-line planar point location problem, the data structure for requirement (A) only needs $O(m)$ preprocessing time and space [22, 44, 54]. The data structure problem in (C) reduces to testing whether a triangle is contained in a face of the subdivision; this reduces to ray shooting (or segment emptiness) queries in a polygonal subdivision, for which there are known results [23]. The total running time is $O(n(\mathcal{H}(S)+1))$, including preprocessing, if the subdivision has size $m = O(n)$. (For this problem, output sizes can be ignored.)

  For the on-line version, we immediately get optimal $O(\mathcal{H}(S)+1)$ average query cost, with an $O(m)$-space data structure for a subdivision of size m. This on-line point location result is already known [6, 7, 29, 41] (some of these previous work even optimize the constant factor in the query cost).

- On-line halfspace range reporting queries in 2-d and 3-d: Here, we map query lower halfspaces to points by duality. The known data structure for (A) needs $O(m)$ space [1, 24]. The data structure for (C) is the same as in Section 4.1. We get optimal $O(\mathcal{H}(S)+1+\kappa)$ average query cost for output size κ, with an $O(m)$-space data structure for a given point set of size m in 2-d or 3-d. This result is new.

- On-line dominance reporting queries in 2-d and 3-d: The story is similar to halfspace range reporting.

- On-line orthogonal range reporting/counting queries in 2-d: Here, we map query rectangles to points in 4-d as in Section 4.1. The known data structure for (A) needs $O(m \log m)$ preprocessing time and $O(m)$

space [21]. The data structure for (C) is the same as in Section 4.1. We get optimal $O(\mathcal{H}(S) + 1 + \kappa)$ average query cost for output size κ, with an $O(m)$-space data structure for a given point set of size m in 2-d. (For counting, $\kappa = O(1)$.) The resulting algorithm works in the comparison model. This result is apparently new, as it extends Dujmović, Howat, and Morin's recent result on 2-d dominance counting [35] and unintentionally answers one of their main open problems (and at the same time improves their space bound from $O(m \log m)$ to $O(m)$).

5 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 which 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 for, say, the convex hull problem.

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 [5] on experimental results for the red/blue segment intersection problem).

Note that specializations of our techniques to 1-d also can easily lead to instance-optimal results for the multiset-sorting problem and the problem of computing the intersection of two (unsorted) sets. Adaptive algorithms for similar 1-d problems (e.g., [51]) were studied in settings different 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-theoretical 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, e.g., arbitrary fixed-degree algebraic functions.

It remains to see 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 (nonorthogona) line segments and a set of disjoint blue line segments in 2-d; (b) computing the L_2- or L_∞-closest pair between a set of red points and a set of blue points in 2-d; (c) computing the diameter or the width of a 2-d point set; (d) computing the lower envelope of a set of (perhaps disjoint) line segments in 2-d.

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., off-line triangular range searching, 3SUM-hard problems, …) would be even more fascinating.

References

[1] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[4] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Self-improving algorithms. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithm*, pages 261–270, 2006.

[5] D. S. Andrews, J. Snoeyink, J. Boritz, T. Chan, G. Denham, J. Harrison, and C. Zhu. Further comparisons of algorithms for geometric intersection problems. In *In Proc. 6th International Symposium on Spatial Data Handling*, pages 709–724, 1994.

[6] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Transactions on Algorithms*, 3(2):17, 2007.

[7] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. *SIAM Journal on Computing*, 37(2):584–610, 2007.

[8] I. Baran and E. D. Demaine. Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. *International Journal of Computational Geometry and Applications*, 15(4):327–350, 2005.

[9] J. Barbay and E. Chen. Adaptive planar convex hull algorithm for a set of convex hulls. In *Proc. 20th Canadian Conference on Computational Geometry*, pages 47–50, 2008.

[10] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 80–86, 1983.

[11] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 179–187, 1990.

[12] B. K. Bhattacharya and S. Sen. On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm. *Journal of Algorithms*, 25(1):177–193, 1997.

[13] J. Boyar and L. M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms*, 3(2):22, May 2007.

[14] T. M. Chan. Fixed-dimensional linear programming queries made easy. In *Proc. 12th ACM Symposium on Computational Geometry*, pages 284–290, 1996.

[15] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry*, 16:361–368, 1996.

[16] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and r elated problems. *Discrete and Computational Geometry*, 16:369–387, 1996.

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

[18] T. M. Chan. Comparison-based time–space lower bounds for selection. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 140–149, 2009.

[19] T. M. Chan. $\Omega(n \log n)$ lower bounds made easy. Note, in preparation.

[20] T. M. Chan, J. Snoeyink, and C.-K. Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete and Computational Geometry*, 18:433–454, 1997.

[21] B. Chazelle. A functional approach to data structures and its use in multidi mensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

[22] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

[23] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[24] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

[25] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry: Theory and Applications*, 5:27–32, 1995.

[26] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 695–702, 1994.

[27] K. L. Clarkson and C. Seshadhri. Self-improving algorithms for Delaunay triangulations. In *Proc. 24th ACM Symposium on Computational Geometry*, pages 148–155, 2008.

[28] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[29] S. Collette, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Distribution-sensitive point location in convex subdivisions. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 912–921, 2008. See also http://arxiv.org/abs/0901.1908.

[30] M. de Berg, M. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.

[31] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997.

[32] E. D. Demaine, D. Harmon, J. Iacono, D. Kane, and M. Pătraşcu. The geometry of binary search trees. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505, 2009.

[33] E. D. Demaine and A. López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, 48(1):2–15, 2003.

[34] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752, 2000.

[35] V. Dujmović, J. Howat, and P. Morin. Biased range trees. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 486–495, 2009.

[36] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

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

[38] J. Erickson. Dense point sets have sparse Delaunay triangulations. *Discrete and Computational Geometry*, 33:83–115, 2005.

[39] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[40] A. Golynski. Cell probe lower bounds for succinct data structures. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 625–634, 2009.

[41] J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, 29:19–22, 2004.

[42] N. Jones. *Computability and Complexity: From a Programming Perspective*. MIT Press, 1997.

[43] C. Kenyon. Best-fit bin-packing with random order. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.

[44] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[45] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proc. 1st ACM Symposium on Computational Geometry*, pages 89–96, 1985.

[46] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

[47] J. Matoušek. Derandomization in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 559–595. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[48] J. Matousek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM Journal on Computing*, 23(1):154–169, 1994.

[49] J. Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992.

[50] S. Moran, M. Snir, and U. Manber. Applications of Ramsey's theorem to decision tree complexity. *Journal of the ACM*, 32(4):938–949, 1985.

[51] J. I. Munro and P. M. Spira. Sorting and searching in multisets. *SIAM Journal on Computing*, 5(1):1–8, 1976.

[52] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[53] S. Sen and N. Gupta. Distribution-sensitive algorithms. *Nordic Journal on Computing*, 6:194–211, 1999.

[54] J. Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 30, pages 559–574. CRC Press LLC, Boca Raton, FL, 1997.

[55] R. Wenger. Randomized quickhull. *Algorithmica*, 17:322–329, 1997.

[56] A. C.-C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28:780–787, 1981.

[57] A. C.-C. Yao. Lower bounds for algebraic computation trees with integer inputs. *SIAM Journal on Computing*, 20(4):655–668, 1991.

[58] F. F. Yao, D. P. Dobkin, H. Edelsbrunner, and M. S. Paterson. Partitioning space for range queries. *SIAM Journal on Computing*, 18(2):371–384, 1989.

A Appendices

A.1 Alternative proof for 2-d maxima

In this subsection, we describe an alternative proof of instance optimality for the 2-d maxima problem. Here, we work with a seemingly different definition of difficulty $\mathcal{F}(S)$, as given below. This 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. Of course, $\mathcal{F}(S)$ will turn out to be asymptotically equivalent to $n\mathcal{H}(S)$, as a byproduct of our analyses. Note that this definition does not seem generalizable to 3-d maxima or other problems.

Definition A.1 Given a point set S, let $q_1, \ldots, q_h$ denote the maximal points of S from left to right. 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 $\mathcal{F}(S) = \sum_{p\in S} \log(n/|F(p)|)$.

The upper-bound proof is similar to our earlier proof:

Theorem A.2 *The 2-d maxima algorithm from Section 2.1 runs in $O(\mathcal{F}(S))$ 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 (i) implies that some maximal point from $\{q_i, \ldots, q_\ell\}$ must been discovered, and (ii) implies that p does not survive level j. Thus, p can survive only for $O(\log(n/|F(p)|))$ levels. We can bound the running time by $O(\sum_j n_j) = O(\sum_p \log(n/|F(p)|))$. □

For the lower-bound side, we first consider a slightly stronger problem which we call *maxima with witnesses*: given a point set S, report 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 $\mathrm{OPT}(S) = \Omega(\mathcal{F}(S))$ *for the 2-d "maxima with witness" problem in the comparison model.*

Proof: The proof is a counting argument, which we express in terms of encoding schemes (see [33, 40] 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. (All logarithms are in base 2.)

To describe the encoding scheme, we imagine that the permutation σ is initially unknown, and as we proceed, we record bits of information about σ so that at the end, σ can be uniquely determined from these bits. In the description below, we distinguish between an *input point*, as represented its index/position in the input permutation σ (its actual location is not necessarily known), and an *actual point* in S, as represented by its coordinates (its position in σ is not necessarily known). At any moment, if we know which input point corresponds to an actual point p, we say (naturally) that p is *known*.

We first simulate the algorithm on σ and record the outcome of the comparisons made; this requires at most $T_A(S)$ bits. Let M be the list of maximal input points returned. For each input point q_i, let $W(q_i)$ be the list of all nonmaximal input points that have q_i as witness. For each maximal actual point, we record its position in M, using at most $h \lceil \log h \rceil$ bits total. Now all maximal points are known.

We process the nonmaximal actual points of S from left to right, and make them known as follows. To process an actual 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 $W(q_i) \cup \cdots \cup W(q_j) - L$. 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 $F(p)$ defined above—a lucky coincidence. Thus, the number of bits required is at most $\lceil \log |F(p)| \rceil$. Now p is known and we can continue the process.

The encoding has total length at most

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

Hence, $\log(n!) \ \le\ T_A(S) + h \log h + n \log n - \mathcal{F}(S) + O(n)$, yielding $T_A(S) = \Omega(\mathcal{F}(S) - n - h \log h)$. Combined with the trivial lower bound $\Omega(n)$ and the naive information-theoretic lower bound $T_A(S) = \Omega(h \log h)$ (as the problem definition requires the output to be in sorted order), this implies that $T_A(S) = \Omega(\mathcal{F}(S))$. □

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

Observation A.4 *Any algorithm for the 2-d maxima problem in the comparison model can be made to solve the 2-d "maxima with witnesses" problem without any further comparisons on every input.*

Proof: Consider the partial order $\prec_x$ over S formed by the outcomes of the x-comparisons made by the maxima algorithm. Define the partial order $\prec_y$ similarly. Fix a nonmaximal point p. We show that there is a point $q \in S$ such that $p \prec_x q$ and $p \prec_y q$. If not, we can modify the x- and y-coordinates, without violating any of the comparisons made, so that all points q with $p \not\prec_x q$ now have $p.x > q.x$, and all points q with $p \not\prec_y q$ now have $p.y > q.y$. Then in the modified point set, p would now be a maximal point, 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 further 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 repeat. At the end, all witnesses are maximal, and no new comparisons are made. □

Remark A.5 The proof still works for the weaker problem where the algorithm can report the maxima in arbitrary order, since by a similar observation, any such algorithm already knows the x-order of the maxima without making any further comparisons.

This proof does not appear to work for problems besides 2-d maxima. One obvious issue is that Observation A.4 only applies to comparison-based algorithms for nonorthogonal problems. Even more critically, however, the proof of Theorem A.3 relies on a coincidence that is special to 2-d maxima.

Curiously, this lower-bound proof holds even for nondeterministic algorithms, i.e., algorithms 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.* [39] and Demaine *et al.* [34] all hold in the nondeterministic settings. Perhaps this strength of the proof prevents its applicability to other geometric problems, whereas our adversary-based proofs more powerfully exploits the deterministic nature of the algorithms.

A.2 The random-order setting

In this subsection, we describe how the preceding lower-bound proofs in the order-oblivious setting can be modified in the random-order setting.

First, the proof in Section A.1 can easily be made to work in the random-order setting, since in any encoding scheme, only a very small (at most $2^{-c_0 n}$) fraction of the $n!$ permutations can have encoding length less than $\log(n!) - c_0 n$ for a constant c_0.

Modifying the proof of Theorem 2.4 requires more effort. We need a technical lemma first:

Lemma A.6 *Suppose we place n random elements independently in t bins, where each element is placed in the k-th bin with probability n_k/n. Then the probability that the k-th bin contains exactly n_k elements for all $k = 1, \ldots, t$ is at least $n^{-O(t)}$.*

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

$$\frac{\Theta(\sqrt{n})(n/e)^{n/e}}{\Theta(\sqrt{n_1})(n_1/e)^{n_1/e}\cdots\Theta(\sqrt{n_t})(n_t/e)^{n_t/e}}\left(\frac{n_1}{n}\right)^{n_1}\cdots\left(\frac{n_t}{n}\right)^{n_t} \geq \frac{1}{\Theta(\sqrt{n})^t}. \qquad \square$$

The new lower-bound proof is loosely inspired by the randomized "bit-revealing" argument by Chan [18]:

Theorem A.7 $\mathrm{OPT}^{\mathrm{avg}}(S) = \Omega(n(\mathcal{H}(S)+1))$ *for the 2-d maxima problem in the comparison model.*

Proof: Fix a sufficiently small constant $\delta > 0$. Let $\mathcal{T}$ be as in the proof of Theorem 2.4, except that we keep only the first $\lfloor \delta \log n \rfloor$ levels of the tree, i.e., when a node reaches depth $\lfloor \delta \log n \rfloor$, it is made a leaf.

Let $\Pi_{\text{kd-tree}}$ be the partition of S formed by the leaf boxes in $\mathcal{T}$. Let $\widetilde{\Pi}_{\text{kd-tree}}$ be a refinement of $\Pi_{\text{kd-tree}}$ in which each subset corresponding to a box of depth $\lfloor \delta \log n \rfloor$ is further subpartitioned into singletons. Note that each such subset has size $\Theta(n^\delta)$ and contributes $\Theta((n^\delta/n)\log n)$ to both the entropy of $\Pi_{\text{kd-tree}}$ and $\widetilde{\Pi}_{\text{kd-tree}}$. Thus, $\mathcal{H}(\widetilde{\Pi}_{\text{kd-tree}}) = \Theta(\mathcal{H}(\Pi_{\text{kd-tree}}))$. Clearly, $\widetilde{\Pi}_{\text{kd-tree}}$ is respectful.

The adversary proceeds differently. We do not explicitly maintain the invariant that no box B is full. Whenever some B_p first becomes a leaf, we assign p at random among the points in $S \cap B_p$ that has previously not been assigned. If all points in $S \cap B_p$ have in fact been assigned, we say that *failure* occurs.

When the simulation encounters a comparison, say, of the x-coordinates, between two points p and q, we do the following:

- We reset B_p to one of its children at random, where each child B'_p is chosen with $|S \cap B'_p|/|S \cap B_p|$ (which is about $1/2$ for a k-d tree construction). We reset B_q similarly to one of its children at random. If the new B_p and B_q are now vertically separated, then the comparison is already resolved. Otherwise, we repeat.

Observe that in the above, if B_p and B_q are both at odd depths and w.l.o.g. the median x-coordinate of B_p is less than the median x-coordinate of B_q, then the comparison is resolved when we choose the left child of B_p and the right child of B_q, and this occurs with probability at least a positive constant (about $1/4$). Thus, with at least a positive constant probability, the comparison is resolved within 2 iterations. 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 B_p over all points $p \in S$ 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 $O(1)$. Let $(*)$ be the event that $D \leq c_0 T$ for a sufficiently large constant c_0. By the Chernoff bound, $\Pr[(*)] \geq 1 - 2^{-\Omega(T)} \geq 1 - 2^{-\Omega(n)}$.

After the end of the simulation, we can do the following postprocessing: whenever there is an internal node B_p, we reset B_p to one of its children at random as above. As a result, every B_p becomes a leaf, and the input is fixed to a permutation of S, so long as failure does not occur.

By the same argument as before, we see that every B_p is already a leaf by the end of the simulation, or failure occurs during simulation or postprocessing. Let $(\dagger)$ be the event that failure does not occur. Thus, if $(*)$ and $(\dagger)$ are both true, then

$$T \;=\; \Omega(D) \;=\; \Omega\left(\sum_{\text{leaf } B} |S \cap B| \log(n/|S \cap B|)\right) \;=\; \Omega(n\mathcal{H}(\Pi_{\text{kd-tree}})) \;=\; \Omega(n\mathcal{H}(\widetilde{\Pi}_{\text{kd-tree}})) \;=\; \Omega(n\mathcal{H}(S)).$$

To analyze $\Pr[(\dagger)]$, consider the leaf box B_p that a point p ends up with after the simulation and post-processing (regardless of whether failure has occurred). This is a random variable, which equals a fixed leaf box B with probability $|S \cap B|/n$. Moreover, all these random variables are independent. Failure occurs iff for some leaf box B, the number of B_p's that equal B is different from $|S \cap B|$. By Lemma A.6, $\Pr[(\dagger)] \geq n^{-O(n^\delta)}$, since there are at most $O(n^\delta)$ leaves in $\mathcal{T}$. It follows that

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

Finally, observe that $\Pr[(\dagger) \wedge \text{the input equals } \sigma]$ is the same for all fixed permutations σ of S (namely the probability is $\prod_{\text{leaf } B} \left(\frac{|S\cap B|}{n}\right)^{|S\cap B|} \frac{1}{|S\cap B|!}$). In other words, conditioned to $(\dagger)$, the input is a random permutation of S, i.e., the adversary does not act adversarily at all! It follows that $T = \Omega(n\mathcal{H}(S))$ with high probability for a random permutation of S. In particular, $E[T] = \Omega(n\mathcal{H}(S))$ for a random permutation of S. □

Applying the same ideas to the proof of Theorem 3.7 shows that $\mathrm{OPT}^{\mathrm{avg}}(S) = \Omega(n(\mathcal{H}(S) + 1))$ for the 2-d upper hull problem in the multilinear decision tree model.

A.3 On the multilinear model

Many commonly encountered test functions in geometric algorithms are multilinear. For example, in 3-d, the predicate $\text{ABOVE}(p_1, \ldots, p_4)$ which returns true iff p_1 is above the plane through p_2, p_3, p_4, reduces to testing signs of multilinear functions.

More generally, say that a function $f : (\mathbb{R}^d)^c \to \mathbb{R}^d$ is *quasi-multilinear* if $f(p_1, \ldots, p_c) = (f_1(p_1, \ldots, p_c)/g(p_1, \ldots, p_c), \ldots, f_d(p_1, \ldots, p_c)/g(p_1, \ldots, p_c))$ for some multilinear functions $f_1, \ldots, f_d, g : (\mathbb{R}^d)^c \to \mathbb{R}$. For example, in 3-d, the function $\text{PLANE}(p_1, \ldots, p_4)$ which returns the dual of the plane through $p_1, \ldots, p_4$, or the function $\text{INTERSECT}(p_1, \ldots, p_4)$ which returns the intersection of the dual planes of $p_1, \ldots, p_4$, are quasi-multilinear. We can get more quasi-multilinear and multilinear functions by composition: e.g., if $f_1, \ldots, f_4 : (\mathbb{R}^3)^4 \to \mathbb{R}^3$ are quasi-multilinear, then $\text{INTERSECT}(f_1(p_1, \ldots, p_4), f_2(p_5, \ldots, p_8), \ldots, f_4(p_{13}, \ldots, p_{16}))$ is quasi-multilinear in $p_1, \ldots, p_{16}$, by expanding all the determinants. More elaborately, a predicate such as

$$\text{ABOVE}(p_{17}, p_{18}, p_{19}, \text{INTERSECT}(\text{PLANE}(p_1, \ldots, p_4), \text{PLANE}(p_5, \ldots, p_8), \ldots, \text{PLANE}(p_{13}, \ldots, p_{16})))$$

also reduces to multilinear tests. However, we may run into problems if a point occurs more than once, e.g.,

$$\text{ABOVE}(p_{17}, p_{18}, p_1, \text{INTERSECT}(\text{PLANE}(p_1, \ldots, p_4), \text{PLANE}(p_5, \ldots, p_8), \ldots, \text{PLANE}(p_{13}, \ldots, p_{16}))),$$

since expansion of the determinants may yield monomials of the wrong type. In most 2-d algorithms, this kind of tests does not arise. Unfortunately, they can occasionally happen in our 3-d upper hull algorithm in Section 3.3. We describe some modifications to the algorithm that can avoid these problematic tests.

First, for the partition construction, it would be easier to choose the second option in the proof of Lemma 3.5. By perturbing the dividing planes, one can show the existence of 3 planes each passing through 3 input points, where the 9 points are distinct, so that each of the resulting 8 regions contains $n/8 \pm O(1)$ points. A brute-force algorithm can find the 3 planes in polynomial time. We can reduce the construction time by using the standard technique of *ε-approximations* [47] (at the expense of a small change in the constant). It can be checked that known constructions for ε-approximations fits in the multilinear model (it suffices to check the implementation of the "subsystem oracle"). As a result, we can ensure that the cells are all defined by planes that pass through 3 input points, where no two planes share a common defining point. A vertex v of a cell is an intersection of 3 such planes and is defined by a set of 9 distinct input points, denoted $\text{DEF}(v)$.

A problem occurs in testing whether a vertex v of a cell γ_i lies below the upper hull, specifically, when we try to compare v against a feature that share a common defining point. For this reason, we weaken the test in line 5: we prune only when each vertex v of γ_i lies strictly below the upper hull of $Q - \text{DEF}(v)$. It can be checked that some version of Lemma 3.8 can support $O(r)$ such queries in the multilinear model, in $O(n \log r + rn^{1-\alpha})$ time for some $\alpha > 0$.

Since the pruning condition is weaker, the analysis needs more effort. We assume that the partition in line 3 is generated hierarchically in the following way: first we find a partition of Q by Lemma 3.5 with $\sqrt{r_j}$ subsets Q'_ℓ and cells γ'_ℓ; then for each subset Q'_ℓ, we find a partition of Q'_ℓ again by Lemma 3.5 with $\sqrt{r_j}$ subsubsets Q_i and cells γ_i.

In the second paragraph of the proof of Theorem 3.9, we proceed differently. Suppose a point p lies in the subset Q'_ℓ and the subsubset Q_i. Observe that if the corresponding cells γ'_ℓ and γ_i are both completely inside Δ_k, then all points inside γ_i are pruned. This is because for each vertex v of the cell γ_i, the defining points $\text{DEF}(v)$ are contained in $Q'_\ell \subset \gamma'_\ell$ and so cannot appear on the upper hull of Q; the vertex v lies strictly below the upper hull of Q, which coincides with the upper hull of $Q - \text{DEF}(v)$.

At most $O(\sqrt{r_j}^{1-\varepsilon})$ cells γ'_ℓ intersect the boundary of Δ_k. At most $O(\sqrt{r_j} \cdot \sqrt{r_j}^{1-\varepsilon})$ cells γ_i intersect the boundary of Δ_k. Hence, the number of points in S_k that remain in Q after iteration j is at most $\min\left\{|S_k|, O(\sqrt{r_j}^{1-\varepsilon} \cdot n/\sqrt{r_j} + \sqrt{r_j} \cdot \sqrt{r_j}^{1-\varepsilon} \cdot n/r_j)\right\} = \min\left\{|S_k|, O(n/r_j^{\varepsilon/2})\right\}$. The rest of the proof is then the same, after readjusting ε.