Semi-Online Maintenance of Geometric Optima and Measures[*]

Timothy M. Chan[†]

January 27, 2003

Abstract

We give the first nontrivial worst-case results for dynamic versions of various basic geometric optimization and measure problems under the semi-online model, where during the insertion of an object we are told when the object is to be deleted. Problems that we can solve with sublinear update time include the Hausdorff distance of two point sets, discrete 1-center, largest empty circle, convex hull volume in three dimensions, volume of the union of axis-parallel cubes, and minimum enclosing rectangle. The decision versions of the Hausdorff distance and discrete 1-center problems can be solved fully dynamically. Some applications are mentioned.

Key words. computational geometry, dynamic data structures

AMS subject classifications. 68U05, 68Q25, 68P05

Abbreviated title. Maintenance of Geometric Optima and Measures

1 Introduction

Problems in computational geometry that admit simple and efficient static solutions can often be significantly harder to solve in the *dynamic* setting, when data are inserted and deleted and answers have to be updated quickly. For example, the *width* of a planar n-point set is an easy-to-state quantity and can be computed by a "textbook" $O(n \log n)$ algorithm, but a data structure that can maintain the width under arbitrary point updates in a manner faster than recomputing from scratch had eluded researchers for years and was found only recently [8]. In this paper, we look at more standard geometric optimization and measure problems and study their worst-case complexities in the dynamic setting, and try to gain a better understanding into generally what types of problems admit nontrivial dynamization results.

The importance of dynamic computational geometry was realized long ago [10], and while there have been many fundamental results in the area, our current knowledge is still limited. Dynamic data structures for all kinds of problems reducible to range searching [1], including linear/convex programming, are known. A class of *decomposable* query problems [5] has been recognized as easy, for which simple general tricks are known. A useful technique has been devised to deal with problems of the form, "which pair of objects minimizes a function?" [14] Yet, simple non-convex minimization

[*]A preliminary version of this paper appeared in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, 2002. This work was supported in part by an NSERC Research Grant.

[†]School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).

Semi-Online Update Time		
Hausdorff distance (2-d)	$\widetilde{O}(n^{5/6})$	(Corollary 3.2)
Discrete 1-center (2-d)	$\widetilde{O}(n^{5/6})$	(Corollary 3.2)
Largest empty circle	$\widetilde{O}(n^{7/8})$	(Corollary 5.2)
Volume of 3-d convex hull	$\widetilde{O}(n^{7/8})$	(Corollary 5.3)
Volume of union of unit cubes	$\widetilde{O}(n^{1/2})$	(Theorem 6.1)
Minimum-perimeter/area rectangle	$\widetilde{O}(n^{1/2})$ / $\widetilde{O}(n^{5/6})$	(Corollary 7.2/7.3)

Fully Dynamic Update Time (amortized)		
Hausdorff distance decision (2-d)	$\widetilde{O}(n^{1/2})$	(Corollary 4.2)
Discrete 1-center decision (2-d)	$\widetilde{O}(n^{1/2})$	(Corollary 4.2)
Largest discrete empty circle	$\widetilde{O}(1)$	(Corollary 4.4)

Table 1: Summary of results.

problems, like the width or smallest enclosing rectangle, or max-min problems, like the Hausdorff distance (see next page), defy solutions by all these methodologies.

Over a decade ago, Dobkin and Suri [11] introduced the *semi-online* model as a restricted form of dynamic computation. The model assumes that when an object is inserted, we are given the time at which the object is to be deleted. The model simultaneously generalizes the *incremental* case (when there are no deletions) and the *off-line* case (when the entire insertion/deletion sequence is known in advance). Considering the apparent difficulty in obtaining worst-case results under the original fully dynamic model, such a restriction is generally a reasonable starting point for research; it lets us obtain semi-dynamic algorithms that are provably efficient and still practically relevant. We adopt the semi-online model in this paper and initiate the study of problems of a broader scope and higher complexity than those originally considered by Dobkin and Suri themselves (as we now know, most of the problems they considered can be solved fully dynamically by the techniques mentioned, notably Eppstein's work [14]).

Table 1 gives a list of problems familiar to computational geometers and the results obtained in this paper. Background to many of the problems can be found in texts such as [27]. Statically, each problem can be solved in $O(n \log n)$ time, by construction of a planar convex hull, 3-dimensional convex hull (in particular, planar Voronoi diagram), or 3-dimensional union of cubes. Dynamically, however, one can see plenty of challenges—we are unaware of nontrivial worst-case results for any of these problems, even in the incremental case!

Specifically, the first problem is to determine the Hausdorff distance $\max_{q \in Q} \min_{p \in P} d(p, q)$ for two given sets P and Q of n points in the plane, subject to semi-online updates to both P and Q, where $d(\cdot, \cdot)$ is the Euclidean metric; the Hausdorff distance is commonly used as a measure of resemblance between two images. Given planar point set P, the discrete 1-center problem asks for the smallest circle, centered at a point of P, that encloses P; this is a popular variant of the standard 1-center problem that cannot be solved by convex programming (in fact, Dobkin and Suri [11] raised this as an open question for semi-online algorithms). In contrast, the largest empty circle problem, another example in facility location, seeks the largest circle whose interior avoids P, with center inside a given region, say, a triangle Δ (note that there are different versions of the center constraint in the literature). The next two are measure problems in three dimensions and ask for the volume of the convex hull of n points and of the union of a set of n congruent axis-parallel cubes. The last (related to the width problem) is to find the smallest rectangle, in terms of perimeter or area, that encloses a planar point set; this "bounding box" is allowed to have arbitrary orientation. All these problems have practical applications. We obtain a sublinear semi-online algorithm for each, using $\widetilde{O}(n)$ space. (Throughout the paper, the $\widetilde{O}$ notation hides factors that are $o(n^{\varepsilon})$ for any fixed $\varepsilon > 0$.)

Along the way, we notice a few easier variants that admit sublinear fully dynamic algorithms. These variants include the decision versions of the Hausdorff distance and discrete 1-center problems (decide whether the Hausdorff distance is less than a given value r, and decide whether there exists a circle of radius r, centered at a point of P, that encloses P), as well as the discrete version of the largest empty circle problem (find the largest circle, centered at a point of P, whose interior avoids P except the center).

Our semi-online algorithms all begin with a very simple strategy that was used recently by the author to tackle the width problem [8] and is outlined in the next section (Lemma 2.1). As one might guess, data structures for range searching [1] are exploited to obtain the sublinear time bounds, but in nontrivial (and at times, creative) ways. Most of our results are therefore theoretical. It is assumed that the input is in general position, without loss of generality, by known perturbation schemes.

We mention some "applications" in Section 8, including improved time bounds for Klee's measure problem [26] in the case of 4-dimensional unit hypercubes, and for the minimum-diameter spanning tree problem, which are of independent interest.

2 The strategy for semi-online dynamization

The most common dynamization strategy [5, 24, 25] is based on decomposing a set of objects into subsets, solving the problem on each subset, and combining the answers. An update affects only a small number of subsets and thus can be efficiently handled. Unfortunately, this simple approach is not viable for any of our problems, because they are not directly "decomposable"—there is no effective way to combine the answers when the set is arbitrarily decomposed into a large number of subsets.

Fortunately, another very simple (but lesser known) approach works for us, using a weaker form of decomposability based on dividing the given set into *two* subsets, one of which is kept small, as explained in the lemma below. The strategy was most recently used by the author [8] (building on a previous work by Eppstein [16]) to obtain a fully dynamic algorithm for planar width with $\widetilde{O}(n^{1/2})$ amortized time, by a variant of the lemma that handles arbitrary deletions (with $\alpha = \beta = 1$). A similar idea was also used in one example from Dobkin and Suri's paper [11].

Lemma 2.1 *Consider a problem Π with the following property, where $\alpha \geq 1$ and $0 < \beta \leq 1$ are constants: we can preprocess a set S of n objects into a data structure in $\widetilde{O}(n)$ time and space, such that given any additional set S' of b objects, we can solve Π on the set $S \cup S'$* (block query) *in $\widetilde{O}(b^{\alpha} n^{1-\beta})$ time. Then we can solve Π on a set of n objects under semi-online updates in $\widetilde{O}(n^{1-\beta/(1+\alpha)})$ time per update, using $\widetilde{O}(n)$ space.*

Proof: We store most of the objects in a static set S, preprocessed in the given data structure, and put the rest in an auxiliary list S'. Insertions and deletions are done directly to S', but after every b updates, we reset S' to hold the objects with the b smallest deletion times and S to hold all other objects (this ensures that any object to be deleted in the next round of b updates is indeed in S', not S). The data structure for S has to be rebuilt, in $\widetilde{O}(n)$ time, for every b updates. At any time, S' has $O(b)$ size, so the solution can be computed by a block query in $\widetilde{O}(b^\alpha n^{1-\beta})$ time. The total time for n updates is therefore

$$\widetilde{O}((n/b) \cdot n \,+\, n \cdot (b^\alpha n^{1-\beta})) \;=\; \widetilde{O}(n^{2-\beta/(1+\alpha)}), \tag{1}$$

if we set the parameter $b \approx n^{\beta/(1+\alpha)}$. This proves an amortized time bound of $\widetilde{O}(n^{1-\beta/(1+\alpha)})$.

If the application insists on a *worst-case* time bound, a well-known modification (e.g., see [24, 25]) is required: spread the work of rebuilding the data structure for S evenly over the next $b/2$ updates. The data structure for S is available for the j-th update whenever $j \bmod b \geq b/2$. A similar "shifted" version of S and S' can deal with the other case $j \bmod b < b/2$. $\square$

All our efforts are now directed to demonstrating that our problems obey the requirement of the lemma. This task isn't necessarily straightforward, as we will see when we examine each problem in detail.

On one occasion (in a higher-dimensional application), we find the following modification of the lemma useful:

Lemma 2.2 *Lemma 2.1 still holds if we are unable to preprocess S from scratch in $\widetilde{O}(n)$ time but can create a copy of the data structure for $S \cup S'$* (block insertion)*, given the data structure for S, within $\widetilde{O}(n + b^\alpha n^{1-\beta})$ time.*

Proof: This is similar to the previous proof, except that we will use multiple levels of decomposition (increasing space by a logarithmic factor). Specifically, let $\ell = \lceil \log_2(n/b) \rceil$. For each $i = 1, \ldots, \ell$, we maintain a partition of the objects into a set S_i, stored in a data structure, and an auxiliary list S'_i: insertions and deletions are done directly to S'_i, but after every $n/2^{i+1}$ updates, we reset S'_i to hold all objects with the $n/2^i$ smallest deletion times and S_i to the complement of S'_i. The size of S'_i is always $O(n/2^i)$.

After every $n/2^{i+1}$ updates, we need to rebuild the data structure for S_i. We know that $S_{i-1} \subseteq S_i$ (because at any time, S'_{i-1} must contain all objects with the $n/2^i$ smallest deletion times). Thus, a data structure for S_i can be created by copying the structure for S_{i-1} and inserting $S'_{i-1} \setminus S'_i$ in $O(\frac{n}{2^i b})$ blocks. The total number of block insertions over n updates is therefore $O(\sum_{i=1}^{\ell} 2^{i+1} \cdot \frac{n}{2^i b}) = \widetilde{O}(n/b)$. At any time, the solution can be computed by a block query to the data structure for S_ℓ, so we again arrive at the same expression (1).

If the application insists on a worst-case time bound, we can again spread the construction of the data structures over time, and for each i, maintain two shifted versions of S_i and S'_i to ensure that one of them is available at any given time. We omit the standard details. $\square$

3 Optimizing over discrete points on a polytope

We consider first the problem of maintaining the Hausdorff distance $\max_{q \in Q} \min_{p \in P} d(p, q)$ dynamically. The difficulty here is the apparent necessity to know the nearest neighbor of *each* point $q \in Q$ to the point set P (unlike in the bichromatic closest pair problem studied by Eppstein [14], of the min-min type). As points are inserted to P, a large number of these nearest neighbors could change.

The idea is that with Lemma 2.1 (and Lemma 2.2), we do not need to maintain the newest version of these nearest neighbors after every single update, but only after a block of updates.

The method involves multi-level range searching tools. Although the description assumes familiarity with these tools, it is conceptually not complicated.

The method is quite general: we can combine the nearest neighbor distances by operators other than the maximum, and we can solve the problem in any fixed dimension. For this reason, we state a more general problem. Let d be a constant. Given a set H of hyperplanes in $\mathbb{R}^d$, define a mapping $\lambda_H : \mathbb{R}^{d-1} \to \mathbb{R}^d$ as follows: $\lambda_H(q)$ is the point obtained by lifting q vertically onto the lower envelope of H (in other words, the lowest intersection of H with the vertical line at q). Given a set of points $Q \subset \mathbb{R}^{d-1}$, we wish to maintain implicitly the set of points $\lambda_H(Q) = \{\lambda_H(q) \mid q \in Q\}$ (all lying on a polytope's boundary, as the title of this section suggests); actually, we want the output to be $\Box\lambda_H(Q)$ for some *decomposable* operator $\Box$, satisfying the requirement that for any disjoint pair of sets S_1 and S_2, $\Box S_1$ and $\Box S_2$ can be combined to form $\Box(S_1 \cup S_2)$ in constant time.

Theorem 3.1 *Suppose that we can preprocess an n-point set $Q \subset \mathbb{R}^{d-1}$ in $\widetilde{O}(n)$ time so that $\Box\lambda_h(Q)$ for any hyperplane h in $\mathbb{R}^d$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time, with $\gamma \geq 1/d$. Then we can maintain $\Box\lambda_H(Q)$ for a set Q of at most n points in $\mathbb{R}^{d-1}$ and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(n^{1-\frac{1}{d(\lfloor d/2\rfloor+1)}})$ time per semi-online update to Q and H.*

Proof: We show how to store Q and H so that $\Box\lambda_{H\cup H'}(Q \cup Q')$ can be computed efficiently given small additional blocks Q' and H'.

Preprocess the hyperplanes H in $\widetilde{O}(n)$ time to support vertical ray shooting queries in $\widetilde{O}(n^{1-1/\lfloor d/2\rfloor})$ time [1]. For each $q \in Q$, compute $\lambda_H(q)$; for $d \leq 3$, this step takes $O(n \log n)$ time. In $\widetilde{O}(n)$ time, preprocess $\lambda_H(Q)$ for simplex range searching [1, 23] to form canonical subsets $\{Q_i\}_i$ of total size $\widetilde{O}(n)$, such that given any query simplex $\Delta \subset \mathbb{R}^d$, we can retrieve all points $q \in Q$ with $\lambda_H(q) \in \Delta$ as a union of disjoint canonical subsets $\{Q_i\}_{i\in I}$ in $\widetilde{O}(n^{1-1/d})$ time, with $\sum_{i\in I} |Q_i|^{1-1/d} = \widetilde{O}(n^{1-1/d})$. Precompute $\Box\lambda_H(Q_i)$ for every canonical subset Q_i. In addition, preprocess each Q_i as specified so that given any hyperplane h, $\Box\lambda_h(Q_i)$ can be found in $\widetilde{O}(n^{1-\gamma})$ time; the total preprocessing time is $\widetilde{O}(n)$.

Given query sets Q' and H' of size b, construct the lower envelope of H' in $\widetilde{O}(b^{\lfloor d/2\rfloor})$ time and triangulate it into $\widetilde{O}(b^{\lfloor d/2\rfloor})$ $(d-1)$-simplices. Take each such simplex Δ, defined by hyperplane $h' \in H'$, say.

- Consider the points $q \in Q$ such that $\lambda_H(q)$ lies directly below Δ. By simplex range searching, these points can be partitioned into canonical subsets $\{Q_i\}_{i\in I}$. Take each Q_i. All points $q \in Q_i$ have $\lambda_{H\cup H'}(q) = \lambda_H(q)$; so combine the current answer with $\Box\lambda_H(Q_i)$. The time required is $\widetilde{O}(n^{1-1/d})$.

- Consider the points $q \in Q$ such that $\lambda_H(q)$ lies directly above Δ. Again these points can be partitioned into canonical subsets $\{Q_i\}_{i\in I}$. Take each such Q_i. All points $q \in Q_i$ have $\lambda_{H\cup H'}(q) = \lambda_{h'}(q)$; so combine the current answer with $\Box\lambda_{h'}(Q_i)$. The time required is $\widetilde{O}(\sum_{i\in I} |Q_i|^{1-\gamma}) = \widetilde{O}(n^{1-1/d})$.

Applying this process to all simplices requires $\widetilde{O}(b^{\lfloor d/2\rfloor} n^{1-1/d})$ time overall. The current answer is $\Box\lambda_{H\cup H'}(Q)$. To get $\Box\lambda_{H\cup H'}(Q \cup Q')$, compute $\lambda_{H\cup H'}(q')$ for each $q' \in Q'$ by vertical ray shooting on H and on H', in $\widetilde{O}(bn^{1-1/\lfloor d/2\rfloor})$ total time, and combine the current answer with $\Box\lambda_{H\cup H'}(Q')$.

For $d \leq 3$, the preprocessing time is $\widetilde{O}(n)$, so Lemma 2.1 is applicable, with $\alpha = \lfloor d/2 \rfloor$ and $\beta = 1/d$. For $d \geq 4$, we cannot afford to recompute $\lambda_H(Q)$ from scratch, so Lemma 2.2 is required: to build a new data structure for $Q \cup Q'$ and $H \cup H'$, compute $\lambda_{H \cup H'}(q)$ from $\lambda_H(q)$ for every $q \in Q$, by vertical ray shooting on H', in total time $\widetilde{O}(b^{\lfloor d/2 \rfloor} + n)$; in addition, compute $\lambda_{H \cup H'}(q')$ for every $q' \in Q'$ in total time $\widetilde{O}(bn^{1-1/\lfloor d/2 \rfloor})$; now, the rest of the new data structure can be built from scratch, in $\widetilde{O}(n)$ time. $\square$

Note that we have used simplex range searching only for point sets *in convex position*. If it is possible to improve the range searching results under this special case (for example, it is not difficult in 2-d), the bound in the theorem would be improved as well.

The Hausdorff distance is just a special case of the above problem in one dimension higher by a standard transformation. The discrete 1-center problem can similarly be solved, as it asks for a similar quantity $\min_{q \in P} \max_{p \in P} d(p, q)$. In a forthcoming application, we encounter an additively-weighted variant that seeks $\max_{q \in Q} \min_{p \in P} [d(p, q) + w_q]$ or $\min_{q \in P} \max_{p \in P} [d(p, q) + w_q]$, given weights w_q; this variant can be solved similarly as well.

Corollary 3.2 *In $\mathbb{R}^d$, we can maintain the Hausdorff distance of two sets of at most n points and the discrete 1-center of a set of n points (possibly with additive weights) in $\widetilde{O}(n^{1-\frac{1}{(d+1)(\lceil d/2 \rceil + 1)}})$ time per semi-online update.*

Proof: For the unweighted Hausdorff distance problem, transform each point $p = (a_1, \ldots, a_d)$ in P to a $(d+1)$-dimensional hyperplane h in H with equation $\{(x_1, \ldots, x_{d+1}) \mid x_{d+1} = a_1^2 + \cdots + a_d^2 - 2a_1x_1 - \cdots - 2a_dx_d\}$. The points in $\lambda_H(Q)$ correspond to the nearest neighbors of the points in Q to set P. The operator $\square$ takes the maximum of $x_{d+1} + x_1^2 + \cdots + x_d^2$ (the actual nearest neighbor distance squared) over the points. The requirement in the theorem (finding $\square\lambda_h(Q)$ for a given hyperplane h) translates to finding the largest distance of Q to a given point p, which can be done by farthest neighbor queries with $\gamma = 1/\lceil d/2 \rceil$ [1].

In the weighted case, the operator $\square$ should instead return the maximum of $\sqrt{x_{d+1} + x_1^2 + \cdots + x_d^2} + w$, where w is the weight of the point $q = (x_1, \ldots, x_d)$. The requirement translates to finding weighted farthest neighbors: more precisely, given query point $(a_1, \ldots, a_d)$, find the maximum of $\sqrt{a_1^2 + \cdots + a_d^2 - 2a_1x_1 - \cdots - 2a_dx_d + x_1^2 + \cdots + x_d^2} + w$ over a set of $O(n)$ tuples $(x_1, \ldots, x_d, w)$. We can compare this maximum with any value b by half-space range search in $d + 2$ dimensions: given $(a_1, \ldots, a_d, b)$, find a tuple that satisfies $w \geq b$ or $[a_1^2 + \cdots + a_d^2 - b^2] - 2a_1x_1 - \cdots - 2a_dx_d + 2bw + [x_1^2 + \cdots + x_d^2 - w^2] \geq 0$. This gives a data structure with $\gamma = 1/(\lfloor d/2 \rfloor + 1)$ for the decision query problem, and by parametric or randomized search, for the maximization query problem as well [1]. $\square$

By linearization or the use of lower envelopes of surfaces and semi-algebraic range searching [1], sublinear results can also be obtained for other metric $d(\cdot, \cdot)$ with constant description complexity. For Hausdorff distances under the L_∞ metric, much simplification is possible, since orthogonal range searching [1, 27] replaces simplex/halfspace range searching, and an L_∞-Voronoi diagram [6] replaces the lower envelope; the time bound reduces to $O(n^{1-\frac{1}{\lceil d/2 \rceil + 1}} \operatorname{polylog} n)$.

4 Some easier variants

Next, we give faster algorithms for the *decision* versions of the Hausdorff distance and discrete 1-center problem. The algorithms are in fact fully dynamic and are obtained by directly modifying known range searching structures (more specifically, by augmenting a standard partition tree to store two extra numbers at each node). The idea is actually to generalize the problem of *testing* whether each point is above the lower envelope, to *counting* the number of hyperplanes below each point.

To state the generalized problem, define the mapping $c_H : \mathbb{R}^d \to \mathbb{N}$, where $c_H(q)$ is the number of hyperplanes of H that lie below q. Implicitly, we wish to maintain the multiset of numbers $c_H(Q) = \{c_H(q) \mid q \in Q\}$ for a given set of points $Q \subset \mathbb{R}^d$; more precisely, we want to output $\Box c_H(Q)$, where the operator $\Box$ is assumed to be decomposable and furthermore satisfy the property that for any set S of numbers and a number j, $\Box(S + j)$ can be computed from $\Box S$ in constant time (where $S + j = \{i + j \mid i \in S\}$).

Theorem 4.1 *We can maintain $\Box c_H(Q)$ for a set Q of at most n points and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(n^{1-1/d})$ amortized time per update to Q and H fully dynamically, using $O(n)$ space.*

Proof: Assume that the size of H is m instead. Store the dual points of H in a dynamic data structure to support simplex range counting queries in $\widetilde{O}(m^{1-1/d})$ time and updates in $\widetilde{O}(1)$ amortized time [1]. The notation H_Δ denotes the subset of all hyperplanes of H crossing a given simplex Δ.

Matoušek's partition theorem [1, 21] asserts that any set Q of n points can be partitioned into $O(r)$ subsets $\{Q_i\}_i$, each of size at most n/r and enclosed in a simplex Δ_i, with the property that every hyperplane crosses $O(r^{1-1/d})$ of these simplices. We choose r to be a constant here; the construction time is then linear. Assuming that Q itself is enclosed in a simplex Δ, we can ensure that the Δ_i's are all inside Δ (by intersecting with Δ and retriangulating).

Let c_{Δ_i} be the number of hyperplanes in H_Δ that lie completely below Δ_i. Note that the duals of all hyperplanes intersecting Δ and below Δ_i form cells in a hyperplane arrangement of constant size. Therefore, we can compute c_{Δ_i} by a constant number of simplex range counting queries in $\widetilde{O}(m^{1-1/d})$ time.

Our data structure for (Q, Δ) consists of recursively constructed structures for $\{(Q_i, \Delta_i)\}_i$, together with the numbers $\{c_{\Delta_i}\}_i$ and the answer $\Box c_{H_\Delta}(Q)$.

Knowing the subanswers $\Box c_{H_{\Delta_i}}(Q_i)$, we can compute the answer $\Box c_{H_\Delta}(Q)$ as follows. Take each Q_i. All points $q \in Q_i$ have $c_{H_\Delta}(q) = c_{H_{\Delta_i}}(q) + c_{\Delta_i}$; so combine the current answer with $\Box(c_{H_{\Delta_i}}(Q_i) + c_{\Delta_i})$. Repeating for all Q_i's yields the desired answer in constant ($O(r)$) time. By evaluating answers bottom-up, we can thus preprocess our data structure in time $\widetilde{O}(nm^{1-1/d})$.

To insert a hyperplane h to H_Δ, we can first increment the count c_{Δ_i} for each simplex Δ_i completely above h, then recursively insert h to H_{Δ_i} for each simplex Δ_i crossed by h, and finally recompute the answer $\Box c_{H_\Delta}(Q)$ from the subanswers $\Box c_{H_{\Delta_i}}(Q_i)$ in the manner described above. To delete h from H_Δ, we proceed similarly, decrementing the counts c_{Δ_i} instead. The recurrence for the insertion/deletion time is $t(n) = O(r^{1-1/d})t(n/r) + O(r)$, which solves to $t(n) = O(n^{1-1/d+\varepsilon})$ for an arbitrarily small $\varepsilon > 0$, if r is made arbitrarily large.

To delete a point q from Q, we recursively delete q from the subset Q_i that contains it, and then recompute the answer $\Box c_{H_\Delta}(Q)$ from the subanswers as above. The required time is only $O(\log n)$.

Initially Δ can be set to $\mathbb{R}^d$. We have thus obtained a data structure that maintains the value $\Box c_H(Q)$, supports updates to H in $\widetilde{O}(n^{1-1/d})$ time, can be preprocessed for any given Q in $T(n) = \widetilde{O}(nm^{1-1/d})$ time, and supports deletions from Q in $D(n) = O(\log n)$ time.

It remains to handle insertions to Q. For this, we apply a well-known general technique of Bentley and Saxe [5, 24, 25], which decomposes Q into logarithmically many deletion-only subsets (recall that the operator $\Box$ is decomposable) and transforms any data structure with preprocessing time $T(n)$ and deletion time $D(n)$ into a data structure that handles arbitrary updates to Q in amortized time $O((T(n)/n)\log n + D(n)) = \widetilde{O}(m^{1-1/d})$. As can be verified, this transformation preserves our ability to perform insertions and deletions to H, with the update time for H increased by a logarithmic factor, which is still $\widetilde{O}(n^{1-1/d})$. $\Box$

Corollary 4.2 *Given at most n points and at most n balls in $\mathbb{R}^d$, we can determine the ball that contains the least/most points in $\widetilde{O}(n^{1-\frac{1}{d+1}})$ amortized time fully dynamically. In $\mathbb{R}^d$, we can compare the Hausdorff distance of two sets of at most n points or the discrete 1-center radius of a set of n points with a fixed value r in $\widetilde{O}(n^{1-1/d})$ amortized time fully dynamically.*

Proof: Transform each point to a $(d+1)$-dimensional hyperplane as in the proof of Corollary 3.2, transform each ball with center $(x_1, \ldots, x_d)$ and radius r to a $(d+1)$-dimensional point $(x_1, \ldots, x_d, r^2 - x_1^2 - \cdots - x_d^2)$, and take $\Box$ to mean maximum/minimum. The second statement follows from the first: the slight improvement in the time bound is due to the fact that the balls have equal radius; here, the lifted points in Q lie on a common d-dimensional surface in $\mathbb{R}^{d+1}$, and the duals of the hyperplanes in H also lie on a common d-dimensional surface, so Agarwal and Matoušek's improved partition theorem [2] is applicable. $\Box$

Returning to the exact Hausdorff distance problem, we quickly mention a straightforward but fast algorithm for the special case when we expect only a small number of nearest neighbors to change in an update. This result is not surprising, but it makes us appreciate our earlier worst-case algorithms better. Still, the special-case algorithm would be interesting in applications where the update sequence is "random" (e.g., see [15]). The subsequent corollary mentions one particular consequence involving the *all-nearest-neighbors graph*, which connects each point $p \in P$ to its nearest neighbor in $P \setminus p$.

Theorem 4.3 *We can maintain $\lambda_H(Q)$ for a set Q of at most n points in $\mathbb{R}^{d-1}$ and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(kn^{1-\frac{2}{\lfloor d/2\rfloor+1}})$ amortized time fully dynamically, with $\widetilde{O}(n^{2-\frac{2}{\lfloor d/2\rfloor+1}})$ space, where k is the number of changes to $\lambda_H(Q)$.*

Proof: Store H in a dynamic data structure for vertical ray shooting with $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2\rfloor+1}})$ query and amortized update time [1]. Store $\lambda_H(Q)$ in a dynamic data structure for halfspace range reporting with $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2\rfloor+1}} + A)$ query time (A is the answer size) and $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2\rfloor+1}})$ amortized update time [1, 3]. The point set $\lambda_H(Q)$ itself can be maintained as follows: when a point q is inserted/deleted, simply insert/delete $\lambda_H(q)$, computable by vertical ray shooting; when a hyperplane h is inserted to H, find all k points q with $\lambda_H(q)$ above h by a halfspace range reporting query and reset each such $\lambda_H(q)$ to $\lambda_h(q)$; when h is deleted, retrieve all k points q with $\lambda_H(q)$ set to $\lambda_h(q)$ and recompute each such $\lambda_H(q)$ by vertical ray shooting. $\Box$

Corollary 4.4 *In $\mathbb{R}^d$, we can maintain the nearest neighbor of each point in one set of at most n points to another set of at most n points in $\widetilde{O}(kn^{1-\frac{2}{\lceil d/2\rceil+1}})$ amortized time fully dynamically, where k is the number of changes to the nearest neighbors. We can maintain the all-nearest-neighbors graph of an n-point set in $\widetilde{O}(n^{1-\frac{2}{\lceil d/2\rceil+1}})$ amortized time. We can maintain the largest discrete empty ball in the same time.*

Proof: The first statement is immediate by the standard transformation. The second statement can be obtained by a monochromatic variant of the algorithm, with the following well-known observation [28]: the degree of the all-nearest-neighbors graph is bounded by a constant, thus the number of changes to the graph is $k = O(1)$ for every update. The largest distance in this graph is the radius of the largest discrete empty ball. $\square$

The maintenance of the all-nearest-neighbors graph was raised by several researchers in connection with dynamic closest pairs [28], but considering that a dynamic all-nearest-neighbors algorithm can indirectly be used to answer nearest neighbor queries (by repeatedly inserting and then deleting the query point), the above bound is probably close to optimal.

The usual tricks could perhaps make the amortized bounds worst-case.

5 Optimizing over vertices of a 3-polytope

To solve problems like the largest empty circle (the original continuous version), we need to optimize a function over *all* points on a lower envelope rather than just a discrete set of points. Dynamization appears even harder. We may infer from the application that the optimum must be located at a vertex of the polytope (since we are actually maximizing a convex function), but the polytope, and thus its set of vertices, can change drastically in the worst case as hyperplanes are inserted and deleted. (Minimizing a convex function over a polytope is of course convex programming, but maximizing a convex function seems to require examining every vertex.)

We cannot afford to maintain the polytope explicitly after every update, so the idea is again to invoke Lemma 2.1 and use only a static structure, periodically rebuilt after a block of updates. The structure this time is more involved, as it needs to support queries on not just a set of points, but a set of facial features (such as line segments) that come from the polytope.

This approach of implicitly maintaining a polytope again works in any fixed dimension in theory, but we will focus on problems involving 3-d polytopes that have efficient static solutions. (When the dimension exceeds 3, the number of vertices may be $\Omega(n^2)$ or bigger.)

The setup is as before and assumes a decomposable operator $\square$. Given a set H of planes in $\mathbb{R}^3$, let V_H denote the set of vertices of the lower envelope of H. The objective is to maintain $\square V_H$.

Theorem 5.1 *Suppose that we can preprocess a set E of n pairs of planes in $\mathbb{R}^3$ in $\widetilde{O}(n)$ time so that $\square\{h_1 \cap h_2 \cap h \mid (h_1, h_2) \in E\}$ for any plane h can be computed in $\widetilde{O}(n^{1-\gamma})$ time, with $\gamma \geq 1/4$. Then we can maintain $\square V_H$ for a set H of n planes in $\mathbb{R}^3$ in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof: We show how to store a static set H so that $\square V_{H\cup H'}$ can be computed quickly given a block H' of b planes.

First construct the 3-d lower envelope of H in $O(n \log n)$ time [27] and preprocess it to support membership and ray shooting queries in $O(\log n)$ time [1]. In $\widetilde{O}(n)$ time, preprocess its $O(n)$ vertices

V_H for 3-d simplex range searching [1] so that given any tetrahedron Δ, we can compute $\Box(V_H \cap \Delta)$ in $\widetilde{O}(n^{2/3})$ time. Next, in $\widetilde{O}(n)$ time, preprocess the $O(n)$ edges E_H of the lower envelope for triangle-intersection queries, so as to form canonical subsets $\{E_i\}_i$ of total size $\widetilde{O}(n)$, such that given any triangle Δ, we can retrieve all edges (line segments) intersecting Δ as a union of disjoint canonical subsets $\{E_i\}_{i \in I}$ in $\widetilde{O}(n^{3/4})$ time, with $\sum_{i \in I} |E_i|^{3/4} = \widetilde{O}(n^{3/4})$—this involves multi-level range/intersection searching tools [1] (specifically, semi-algebraic range searching [2] in Plücker space; e.g., see [17, proof of Theorem 3.1], which examined a similar subproblem of quadilaterial-intersection queries for rays). For yet another level, preprocess each canonical subset E_i as specified so that given any plane h intersecting every edge of E_i, $\Box\{e \cap h \mid e \in E_i\}$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time.

Given query set H', construct the lower envelope of H' (with vertex set $V_{H'}$ and edge set $E_{H'}$) and triangulate it into $O(b)$ triangles. Take each triangle Δ defined by plane $h' \in H'$, say.

- Vertices of V_H that lie directly below Δ are also vertices of $V_{H \cup H'}$, so combine the current answer with $\Box\{v \in V_H \mid v \text{ directly below } \Delta\}$ by simplex range searching in $\widetilde{O}(n^{2/3})$ time.
- Consider the edges of E_H that intersect Δ. These edges can be partitioned into canonical subsets $\{E_i\}_{i \in I}$. Take each such E_i. The intersection of the edges of E_i with h' are all vertices of $V_{H \cup H'}$, so combine the current answer with $\Box\{e \cap h' \mid e \in E_i\}$. The time required is $\widetilde{O}(\sum_{i \in I} |E_i|^{1-\gamma}) = \widetilde{O}(n^{3/4})$.

Applying this process to all triangles requires $\widetilde{O}(bn^{3/4})$ time overall. So far, all vertices of $V_{H \cup H'}$ that are either vertices of V_H or intersections of edges of E_H with planes of H' have been accounted for. We can also take each edge $e' \in E_{H'}$, determine the at most two vertices of the intersection with the lower envelope of H, by ray shooting, and combine the answer with these vertices. We can further take each vertex of $V_{H'}$ that lies below the lower envelope of H, and combine the answer with such vertices. The additional time is $O(b \log n)$, and the end result is $\Box V_{H \cup H'}$.

The conclusion now follows from Lemma 2.1 with $\alpha = 1$ and $\beta = 1/4$. $\Box$

Corollary 5.2 *We can maintain the largest empty circle of an n-point set in $\mathbb{R}^2$, with center restricted inside any given triangle Δ, in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof: Apply the same transformation as in the proof of Corollary 3.2 to obtain n planes in $\mathbb{R}^3$. Add three nearly vertical planes along the edges of Δ (to ensure that the n planes cannot be seen from below when outside Δ). The largest empty circle must be centered at a vertex of the lower envelope of these $n+3$ planes H (usually a Voronoi vertex, except in boundary cases). The optimal radius is then $\Box V_H$, with the same operator $\Box$ to maximize $x_3 + x_1^2 + x_2^2$. The requirement in the theorem (finding $\Box\{h_1 \cap h_2 \cap h \mid (h_1, h_2) \in E\}$) seeks, for a given query plane h, the maximum of $O(n)$ functions in terms of h, parametrizable in 2 variables (since our planes are defined as liftings of points in $\mathbb{R}^2$). These nasty-looking bivariate functions nonetheless have constant description complexity, and by known semi-algebraic range searching and vertical ray shooting techniques in 3-d [1, 2], we can achieve $\gamma = 1/3$. $\Box$

Corollary 5.3 *We can maintain the volume of the convex hull of an n-point set in $\mathbb{R}^3$ in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof: Apply duality to transform each given point p to a plane p^* so that facets of the upper hull of the points corresponds to vertices of the lower envelope of the planes. For a vertex v defined by planes p_1^*, p_2^*, and p_3^*, we associate with it the volume of the tetrahedron $op_1p_2p_3$ for some fixed point o sufficiently far below the convex hull. The operator $\square$ just sums the volumes associated with the vertices of the given set. Applying a similar process to the lower hull and taking the difference yields the volume of the convex hull.

The theorem requires the sum of the volumes of op_1p_2p over n given pairs (p_1, p_2) and a query point p. By inspecting the proof of the theorem, we can ensure that the generated pairs (p_1, p_2) are consistently oriented with respect to the query point p. The volume of op_1p_2p is then a linear function in p (defined by a standard determinant), so the sum is also a linear function in p. By precomputing the coefficients in linear time, we can answer the volume-sum query in constant time for any given p, so $\gamma = 1$. $\square$

One can imagine more applications of Theorem 5.1: for example, counting the number of vertices of a convex hull in $\mathbb{R}^3$, determining the smallest/largest-area Delaunay triangle in $\mathbb{R}^2$, etc. (The surface area of the convex hull though behaves differently, as we have to sum square roots.)

6 Measuring a union of unit cubes

We can apply the same approach to implicitly maintain structures other than a 3-polytope. To illustrate the idea on a simple example, we now explore the union of n unit axis-parallel cubes in three dimensions, a structure also known to have linear complexity [6]. $O(\sqrt{n}\log n)$ dynamic algorithms for measuring the union of squares (in fact, arbitrary axis-parallel rectangles) were previously known by simple variants of the k-d tree (see [26]).

Theorem 6.1 *We can maintain the volume of the union of a collection C of n unit axis-parallel cubes in $\mathbb{R}^3$ in $\widetilde{O}(\sqrt{n})$ time per semi-online update.*

Proof: We show how to store C so that the volume of $\bigcup(C \cup C')$ can be computed quickly given an additional set C' of b unit cubes.

First compute $\bigcup C$ and decompose this 3-d region into a collection S of $O(n)$ disjoint boxes (boxes here are axis-parallel); this computation can be done in $O(n \log n)$ time, as shown in [9]. Preprocess S in $\widetilde{O}(n)$ time so that given any query box $q \subset \mathbb{R}^3$, we can determine the sum of the volumes of $\sigma \cap q$ over all $\sigma \in S$ in $\widetilde{O}(1)$ time; in a moment, we will see exactly how this can be accomplished by orthogonal range searching.

Given query set C', compute $\bigcup C'$ and decompose the complement of the region into a collection S' of $O(b)$ disjoint boxes. Take each box $\sigma' \in S'$. Perform the above query to find the total volume of $\sigma \cap \sigma'$ over all $\sigma \in S$ in $\widetilde{O}(1)$ time. Summing over all $\sigma' \in S'$ yields the total volume of $\bigcup C$ outside $\bigcup C'$. Adding the volume of $\bigcup C'$ itself yields the final answer. The overall time is $\widetilde{O}(b)$, so we can apply Lemma 2.1 with $\alpha = \beta = 1$.

It remains to describe how to sum volumes of $\sigma \cap q$ over all $\sigma \in S$ for a given query box q. Lift each box $\sigma \in \mathbb{R}^3$ to a point $\sigma^* \in \mathbb{R}^6$ by taking the six coordinates of the box. By orthogonal range searching [1, 27], we can retrieve all boxes in S whose liftings lie in one of the 2^6 quadrants at q^* as a union of $\widetilde{O}(1)$ canonical subsets in $\widetilde{O}(1)$ time, after $\widetilde{O}(n)$-time preprocessing into canonical subsets of $\widetilde{O}(n)$ total size. Now, boxes σ inside each such quadrant intersect q (if at all) in a consistent "pattern" so that the volume of $\sigma \cap q$ can be characterized by a polynomial function (degree 3 at most) on the coordinates of q. By precomputing the (constant number of) coefficients of the sum of these polynomial functions in linear time for each canonical subset, we can therefore compute the sum of the volume of $\sigma \cap q$ over all σ^* inside a quadrant at q^* in $\widetilde{O}(1)$ time, for any given q. Summing over all quadrants answers the query. $\square$

The running time above is actually $O(\sqrt{n}\,\mathrm{polylog}\, n)$. With more effort, we should be able to maintain the surface area of the union of n unit axis-parallel cubes in $\mathbb{R}^3$, or the area/perimeter of a union of n homothets of a constant-size convex polygon in $\mathbb{R}^2$ (the latter union also has linear complexity).

7 Optimizing with multiple convex polygons

We started with optimization problems dealing with the interaction of points and a polytope. We will close with a more complicated form of optimization, but in the 2-d plane, dealing with interaction between two or more convex polygons. The planar width problem is perhaps the simplest in this category and was addressed in a previous paper [8]. Its relative, the minimum enclosing rectangle problem, is the main subject of this section; unlike in [8], we are not able to obtain a fully dynamic algorithm because of the added complications.

We first state the abstract problem. Given a set of lines H, let V_H be as before (the set of vertices of the lower envelope), but abusing notation slightly, let λ_H be instead a mapping from $\mathbb{R}^2$ to $\mathbb{R}^2$: $\lambda_H(q)$ is the point lying on the lower envelope of H and the vertical line at q. Let s be a constant. Below, $\langle \lambda_{H_1}, \ldots, \lambda_{H_s}\rangle(Q)$ denotes the set of tuples of points $\{\langle \lambda_{H_1}(q), \ldots, \lambda_{H_s}(q)\rangle \mid q \in Q\}$. The objective is to maintain $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_s}\rangle(V_{H_1} \cup \cdots \cup V_{H_s})$ for s sets of lines $H_1, \ldots, H_s$ and some decomposable operator $\square$.

Notation: Given a proper subset of indices $J \subset \{1, \ldots, s\}$, the *$J$-selector* refers to the following operator $\otimes$: $\langle p_1, \ldots, p_s\rangle \otimes \langle q_1, \ldots, q_s\rangle = \langle r_1, \ldots, r_s\rangle$, where $r_j = p_j$ if $j \in J$, and $r_j = q_j$ if $j \notin J$. There are a constant number $(2^s - 1)$ of such selectors.

Theorem 7.1 *Suppose that we can preprocess a set $S \subset (\mathbb{R}^2)^s$ of n tuples (each consisting of s points on a common vertical line) in $\widetilde{O}(n)$ time so that given any s lines $h_1, \ldots, h_s$ and selector $\otimes$, $\square\{\langle \lambda_{h_1}(p_1), \ldots, \lambda_{h_s}(p_1)\rangle \otimes \langle p_1, \ldots, p_s\rangle \mid \langle p_1, \ldots, p_s\rangle \in P\}$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time. Then we can maintain $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_s}\rangle(V_{H_1} \cup \cdots \cup V_{H_s})$ for given sets $H_1, \ldots, H_s$ of at most n lines in $\mathbb{R}^2$ in $\widetilde{O}(n^{1-\gamma/2})$ time per semi-online update to $H_1, \ldots, H_s$.*

Proof: We show how to store $H_1, \ldots, H_s$ so that $\square\langle \lambda_{H_1 \cup H_1'}, \ldots, \lambda_{H_s \cup H_s'}\rangle(V_{H_1 \cup H_1'} \cup \cdots \cup V_{H_s \cup H_s'})$ can be computed efficiently given additional blocks $H_1', \ldots, H_s'$ of size b.

Compute the lower envelope of each H_j (a convex polygon) in $O(n \log n)$ time. For each $v \in V_{H_1} \cup \cdots \cup V_{H_s}$, find $\lambda_{H_j}(v)$ by binary search. Using a binary tree construction, we can form canonical subsets $\{V_i\}_i$ of total size $O(n \log n)$, such that for any j, we can retrieve all points $v \in V_{H_j}$ inside a vertical slab as a union of $O(\log n)$ disjoint canonical subsets. For each canonical subset V_i, preprocess the tuples $\{\langle \lambda_{H_1}(v), \ldots, \lambda_{H_s}(v)\rangle \mid v \in V_i\}$ in the specified data structure.

Given query sets $H_1', \ldots, H_s'$, construct the lower envelope of each H_j' in $O(b \log b)$ time. Draw vertical lines at the vertices of these s envelopes. In addition, intersect each edge of these envelopes with each of the lower envelopes of $H_1, \ldots, H_s$, by binary search [27], in total $O(b \log n)$ time. Draw vertical lines at these intersections. As a result, the plane is divided into $O(b)$ vertical slabs. Take each vertical open slab σ. Within σ, the lower envelope of $H_j \cup H'_j$ coincides with either the lower envelope of H_j or a single line $h'_j \in H'_j$. So, $V_{H_j \cup H'_j} \cap \sigma$ is either $V_{H_j} \cap \sigma$ or $\emptyset$. Assume the former.

Partition the point set $V_{H_j} \cap \sigma$ into $O(\log n)$ canonical subsets. Take each canonical subset V_i. Now, $\lambda_{H_k \cup H'_k}(V_i)$ is either $\lambda_{H_k}(V_i)$ or $\lambda_{h'_k}(V_i)$ for each k. Thus, $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle(V_i)$ is merely $\square\{\langle \lambda_{h'_1}(v), \ldots, \lambda_{h'_s}(v)\rangle \otimes \langle \lambda_{H_1}(v), \ldots, \lambda_{H_s}(v)\rangle \mid v \in V_i\}$ for some selector $\otimes$, and so can be computed in $\widetilde{O}(n^{1-\gamma})$ time per canonical subset V_i.

We conclude that $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle((V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s}) \cap \sigma)$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time. Repeating this process over all slabs requires $\widetilde{O}(bn^{1-\gamma})$ time overall. In addition, each of the $O(b)$ vertical lines drawn may pass through a vertex v_0 of $V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s}$. If so, compute $\langle \lambda_{H_1 \cup H'_1}(v_0), \ldots, \lambda_{H_s \cup H'_s}(v_0)\rangle$, by binary searches on the lower envelopes of H_j and H'_j, and combine with the answer. The additional time is $O(b \log n)$, and the end result is $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle(V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s})$.

The conclusion now follows from Lemma 2.1 with $\alpha = 1$ and $\beta = \gamma$. $\square$

Corollary 7.2 *We can maintain the minimum-perimeter rectangle enclosing an n-point set $P \subset \mathbb{R}^2$ in $\widetilde{O}(n^{1/2})$ time per semi-online update.*

Proof: We represent the rectangle using five parameters $\xi, \eta_1, \ldots, \eta_4$:

$$\{(x, y) \mid -\eta_1 \le \xi x + y \le \eta_2, \quad -\eta_3 \le x - \xi y \le \eta_4\}.$$

The perimeter is $2(\eta_1 + \cdots + \eta_4)/\sqrt{1+\xi^2}$.

So, transform each point $(a, b) \in P$ to the following four lines: $\{(\xi, \eta_1) \mid \eta_1 = -\xi a - b\}$ in H_1, $\{(\xi, \eta_2) \mid \eta_2 = \xi a + b\}$ in H_2, $\{(\xi, \eta_3) \mid \eta_3 = -a + \xi b\}$ in H_3, and $\{(\xi, \eta_4) \mid \eta_4 = a - \xi b\}$ in H_4. Define the operator $\square$ to minimize $(\eta_1 + \cdots + \eta_4)/\sqrt{1+\xi^2}$ over all tuples $\langle(\xi, \eta_1), \ldots, (\xi, \eta_4)\rangle$ of the given set. Our problem is equivalent to determining $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_4}\rangle(\mathbb{R}^2)$. Since one of four sides of the optimal rectangle must be defined by a convex hull edge of the original points (see the Appendix), our problem reduces to finding $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_4}\rangle(V_{H_1} \cup \cdots \cup V_{H_4})$, so Theorem 7.1 is applicable with $s = 4$.

Now, given the selector's index set $J \subset \{1, \ldots, 4\}$, the theorem requires a data structure to store a set S of n tuples so that given query lines $\{(x, y) \mid y = s_j x + t_j\}$ $(j = 1, \ldots, 4)$, we can find the tuple $\langle(\xi, \eta_1), \ldots, (\xi, \eta_4)\rangle \in S$ that maximizes

$$\frac{\sum_{j \in J}(s_j \xi + t_j) + \sum_{j \notin J} \eta_j}{\sqrt{1+\xi^2}} \;=\; \frac{\xi}{\sqrt{1+\xi^2}} \sum_{j \in J} s_j + \frac{1}{\sqrt{1+\xi^2}} \sum_{j \in J} t_j + \frac{\sum_{j \notin J} \eta_j}{\sqrt{1+\xi^2}}.$$

By storing the plane $\{(X, Y, Z) \mid Z = \frac{\xi}{\sqrt{1+\xi^2}} X + \frac{1}{\sqrt{1+\xi^2}} Y + \frac{\sum_{j \notin J} \eta_j}{\sqrt{1+\xi^2}}\}$ associated with each tuple for vertical ray shooting in 3-d [1, 27] (by constructing a 3-d convex hull and performing 2-d point location), we can achieve $\gamma = 1$. $\square$

Corollary 7.3 *We can maintain the minimum-area rectangle enclosing an n-point set $P \subset \mathbb{R}^2$ in $\widetilde{O}(n^{5/6})$ time per semi-online update.*

Proof: Proceed as in the previous proof, but with a different objective to minimize $(\eta_1+\eta_2)(\eta_3+\eta_4)/(1+\xi^2)$.

The data structuring requirement is now more complex. Again we want to preprocess n tuples of the form $\langle(\xi,\eta_1),\ldots,(\xi,\eta_4)\rangle$. Let $\{(x,y)\mid y=s_jx+t_j\}$ $(j=1,\ldots,4)$ be the query lines. We consider the following cases of selectors only, as all other cases are symmetric or trivial.

- $J=\{1\}$. We want to minimize

$$\frac{(s_1\xi+t_1+\eta_2)(\eta_3+\eta_4)}{1+\xi^2} \;=\; \frac{\xi(\eta_3+\eta_4)}{1+\xi^2}s_1+\frac{\eta_3+\eta_4}{1+\xi^2}t_1+\frac{\eta_2(\eta_3+\eta_4)}{1+\xi^2}.$$

  By storing the plane $\{(X,Y,Z)\mid Z=\frac{\xi(\eta_3+\eta_4)}{1+\xi^2}X+\frac{\eta_3+\eta_4}{1+\xi^2}Y+\frac{\eta_2(\eta_3+\eta_4)}{1+\xi^2}\}$ associated with each tuple for 3-d vertical ray shooting as before, we can handle this type of queries in $\widetilde{O}(1)$ time.

- $J=\{1,2\}$. We want to minimize

$$\frac{(s_1\xi+t_1+s_2\xi+t_2)(\eta_3+\eta_4)}{1+\xi^2} \;=\; \frac{\xi(\eta_3+\eta_4)}{1+\xi^2}[s_1+s_2]+\frac{\eta_3+\eta_4}{1+\xi^2}[t_1+t_2].$$

  This case reduces to 2-d vertical ray shooting.

- $J=\{1,3\}$. This is the most involved case and we will use linearization here [1]. We want to minimize

$$\begin{aligned}\frac{(s_1\xi+t_1+\eta_2)(s_3\xi+t_3+\eta_4)}{1+\xi^2} \;=\;& s_1s_3+\frac{1}{1+\xi^2}[t_1t_3-s_1s_3]+\frac{\xi}{1+\xi^2}[s_1t_3+s_3t_1]+\\ & \frac{\xi\eta_4}{1+\xi^2}s_1+\frac{\eta_4}{1+\xi^2}t_1+\frac{\xi\eta_2}{1+\xi^2}s_3+\frac{\eta_2}{1+\xi^2}t_3+\frac{\eta_2\eta_4}{1+\xi^2}.\end{aligned}$$

  By storing the hyperplane $\{(X_1,\ldots,X_7)\mid X_7=\frac{1}{1+\xi^2}X_1+\frac{\xi}{1+\xi^2}X_2+\frac{\xi\eta_4}{1+\xi^2}X_3+\frac{\eta_4}{1+\xi^2}X_4+\frac{\xi\eta_2}{1+\xi^2}X_5+\frac{\eta_2}{1+\xi^2}X_6+\frac{\eta_2\eta_4}{1+\xi^2}\}$ associated with each tuple for 7-d vertical ray shooting, we can handle this type of queries in $\widetilde{O}(n^{2/3})$ time with $\widetilde{O}(n)$ preprocessing [1, 22].

- $J=\{1,2,3\}$. This is similar to the previous case. We want to minimize

$$\begin{aligned}\frac{(s_1\xi+t_1+s_2\xi+t_2)(s_3\xi+t_3+\eta_4)}{1+\xi^2} \;=\;& (s_1+s_2)s_3+\frac{1}{1+\xi^2}[(t_1+t_2)t_3-(s_1+s_2)s_3]+\\ & \frac{\xi}{1+\xi^2}[(s_1+s_2)t_3+s_3(t_1+t_2)]+\frac{\xi\eta_4}{1+\xi^2}[s_1+s_2]+\frac{\eta_4}{1+\xi^2}[t_1+t_2].\end{aligned}$$

  Again we can use vertical ray shooting, this time in 4-d.

Thus, we can achieve $\gamma=1/3$. □

The running time for Corollary 7.2 is actually $O(\sqrt{n}\,\mathrm{polylog}\,n)$ (only elementary tools are used). We will leave the reader with the question of whether the same approach works for similar problems like the minimum enclosing equilateral triangle (or convex polygon of a fixed angle sequence).

8 Some consequences

Faster dynamic data structures can generally lead to faster implementations of static algorithms. We briefly indicate a few sample applications of our results to illustrate their importance.

The generalized discrete 2-center problem. Given an n-point set $P \subset \mathbb{R}^2$, we want to find two points $p_1, p_2 \in P$ to minimize $f(r_1, r_2)$, such that every point $q \in P$ is within radius r_1 of p_1 *or* within radius r_2 of p_2. Here, $f(\cdot,\cdot)$ is some constant-time computable function that is monotone increasing in both arguments. Agarwal *et al.* [4] studied the most basic version with $f(r_1, r_2) = \max\{r_1, r_2\}$ and gave an $\widetilde{O}(n^{4/3})$ algorithm, but other functions, such as $f(r_1, r_2) = r_1 + r_2$, are reasonable in some situations. (See [13] for results on the generalized version of the original 2-center problem, where p_1 and p_2 can be arbitrary points in $\mathbb{R}^2$.)

An exhaustive algorithm may try each point $p_1 \in P$, shrink a ball B centered at p_1 with a decreasing radius r_1, and compute $r_2 = \min_{q \in P} \max_{p \in P \setminus B} d(p, q)$. We need to try n possible radii r_1 for each of the n choices for p_1, and if we compute r_2 from scratch in $O(n \log n)$ time via the farthest-point Voronoi diagram each time, the total running time would be $O(n^3 \log n)$. By applying the method of Corollary 3.2 and noting that $P \setminus B$ is subjected to insertions only as B shrinks (the insertion sequence is actually off-line), we immediately obtain an improved time bound of $\widetilde{O}(n^{3-1/6})$. In any constant dimension d, the bound is $\widetilde{O}(n^{3-\frac{1}{(d+1)(\lceil d/2 \rceil+1)}})$.

The minimum-diameter spanning tree. An interesting application of the generalized discrete 2-center problem was considered by Ho *et al.* [20]: given an n-point set $P \subset \mathbb{R}^d$, find a spanning tree that minimizes its *diameter* (i.e., the maximum distance over all pairs of points, where the "distance" between p and q refers to the sum of the edge lengths, measured in the Euclidean metric, along the path connecting p and q in the tree). As Ho *et al.* showed, the resulting tree turns out to have very low *link diameter* (every pair of points is connected by a path with at most three edges), and consequently, the problem reduces to an additively weighted version of the discrete 2-center problem, where the objective is to minimize $r_1 + r_2 + d(p_1, p_2)$.

We can use the same exhaustive-search algorithm, except that while considering a center candidate p_1, we assign each point $q \in P \setminus B$ an additive weight of $w_q = d(p_1, q)$ and maintain $\min_{q \in P} \max_{p \in P \setminus B}[d(p, q) + w_q]$ instead (using the method of Corollary 3.2). This results in the first subcubic time bound ($\widetilde{O}(n^{3-\frac{1}{(d+1)(\lceil d/2 \rceil+1)}})$) for the problem.

Greedy disk cover. Consider the following (NP-hard) geometric version of the set cover problem: given n points and n balls in $\mathbb{R}^d$, find the smallest number of balls that together cover all the points. Although various approximation algorithms have been proposed (e.g., see [7]), the most well-known is perhaps the greedy algorithm (which has a logarithmic approximation factor): choose the ball that covers the most points, remove the ball and all points inside it, and repeat. The naive implementation would require $O(n)$ time per iteration, for a total time of $O(n^2)$. By applying the fully dynamic Corollary 4.2, we can reduce the running time of the greedy algorithm to $O(n^{2-\frac{1}{d+1}})$. For unit balls, the time reduces to $O(n^{2-1/d})$.

Klee's measure problem for unit hypercubes. Klee's measure problem in $\mathbb{R}^d$ seeks the volume of n axis-parallel boxes. The fastest algorithm known is due to Overmars and Yap from 1991 [26] and runs in $O(n^{d/2}\log n)$ time. The algorithm basically exploits an orthogonal binary space partition of the $(d-2)$-faces of the boxes, and because such partitions have a worst-case lower bound of size $\Theta(n^{d/2})$ [12], improvement appears difficult in general.

Better bounds for the special case of unit hypercubes are however possible (e.g., see [18]), because the union of unit hypercubes has size $O(n^{\lfloor d/2\rfloor})$ only [6]. For example, for $d=3$, we can afford to construct the union explicitly [9] and therefore find the volume in $O(n\log n)$ time. In higher dimensions, assuming that there is an algorithm $\mathcal{A}_d$ to construct the union and decompose the interior/exterior into disjoint boxes in $\widetilde{O}(n^{\lfloor d/2\rfloor})$ time (for $d>3$, we are unable to find an explicit reference to such an algorithm), we can solve Klee's problem for unit hypercubes in $\widetilde{O}(n^{\lfloor d/2\rfloor})$ time—an improvement over Overmars and Yap's bound for odd dimensions d.

An interesting question involves the case of unit hypercubes in *even* dimensions $d>2$. By a standard space sweep, the 4-d Klee's problem reduces to the off-line maintenance of the volume of a dynamic 3-d union, and by Theorem 6.1, can therefore be solved in $\widetilde{O}(n^{3/2})$ time. This is surprising considering that the union itself may have quadratic size in $\mathbb{R}^4$. A similar approach works for higher even dimensions and yields a time bound of $\widetilde{O}(n^{\lceil d/2\rceil-1+\frac{1}{\lceil d/2\rceil}})$, assuming the existence of algorithm $\mathcal{A}_{d-1}$ (the simple calculations are left for the interested readers to verify).

Hypercubes of possibly different sizes can have unions of complexity $\Theta(n^{\lceil d/2\rceil})$ [6]. Assuming the existence of an algorithm to construct and decompose the union in near-optimal time, we can also obtain a similar algorithm, in this case with running time $\widetilde{O}(n^{\lfloor d/2\rfloor+\frac{1}{\lfloor d/2\rfloor+1}})$, which is an improvement in odd dimensions $d\geq 5$.

Of course, a solution for hypercubes implies a solution for *fat* axis-parallel boxes, where the edge lengths of a box differ by at most a constant factor. (Unlike Overmars and Yap's method, though, our method does not solve the related problem of computing the *depth* in an arrangement of boxes [18].)

9 Conclusion

We have shown that a number of basic geometric problems have nontrivial (*sublinear*) dynamization results under the *semi-online* model. More important than the specific results themselves, however, are our general strategies for attacking different categories of problems; these strategies can serve as helpful design models to tackle further problems in dynamic computational geometry.

Of course, the ultimate wish is to have *fully* dynamic, *polylogarithmic* algorithms, but at present this appears to be beyond our grasp for any of the problems discussed here. We hope that our results will inspire more work in this challenging area.

Appendix

It is a well-known fact [19] that the minimum-area rectangle enclosing a set of planar points has one side flushed to the convex hull, but since we are unable to find a reference stating the analogous fact for the minimum-perimeter rectangle, we include a quick proof:

Parametrize the rectangle differently, in terms of variables $\xi,\eta,\omega_1,\omega_2,\zeta$:

$$\{(x,y)\mid\ \omega_1\leq \xi x+\eta y\leq \omega_1+\zeta,\ \ \omega_2\leq \eta x-\xi y\leq \omega_2+(1-\zeta)\}.$$

The perimeter is $2/\sqrt{\xi^2+\eta^2}$. The problem is to maximize $\xi^2+\eta^2$ subject to the constraints that the n given points lie in the rectangle—these constraints are linear in $\xi, \eta, \omega_1, \omega_2, \zeta$. To finish, observe that the maximum of a convex function over a polytope must be located at a vertex, here defined by five 5-d bounding hyperplanes, two of which are associated with a common side of the rectangle. □

Acknowledgements

I thank Der-Tsai Lee for (accidentally) mentioning the minimum-diameter spanning tree application to me.

References

[1] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[2] P. K. Agarwal and J. Matoušek, *On range searching with semi-algebraic sets*, *Discrete Comput. Geom.*, 11:393–416, 1994.

[3] P. K. Agarwal and J. Matoušek, *Dynamic half-space range reporting and its applications*, *Algorithmica*, 13:325–345, 1995.

[4] P. K. Agarwal, M. Sharir, and E. Welzl, *The discrete 2-center problem*, *Discrete Comput. Geom.*, 20:287–305, 1998.

[5] J. Bentley and J. Saxe, *Decomposable searching problems I: static-to-dynamic transformation*, *J. Algorithms*, 1:301–358, 1980.

[6] J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec, *Voronoi diagrams in higher dimensions under certain polyhedral distance functions*, *Discrete Comput. Geom.*, 19:473–484, 1998.

[7] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite VC-dimension*, *Discrete Comput. Geom.*, 14:263–279, 1995.

[8] T. M. Chan, *A fully dynamic algorithm for planar width*, in *Proc. 17th ACM Sympos. Comput. Geom.*, pages 172–176, 2001; *Discrete Comput. Geom.*, to appear.

[9] L. P. Chew, D. Dor, A. Efrat, and K. Kedem, *Geometric pattern matching in d-dimensional space*, *Discrete Comput. Geom.*, 21:257–274, 1999.

[10] Y.-J. Chiang and R. Tamassia, *Dynamic algorithms in computational geometry*, *Proc. of the IEEE*, 80:1412–1434, 1992.

[11] D. Dobkin and S. Suri, *Maintenance of geometric extrema*, *J. ACM*, 38:275–298, 1991.

[12] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, *Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles*, in *Proc. 17th ACM Sympos. Comput. Geom.*, pages 141–150, 2001.

[13] D. Eppstein, *Dynamic three-dimensional linear programming*, *ORSA J. Comput.*, 4:360–368, 1992.

[14] D. Eppstein, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, *Discrete Comput. Geom.*, 13:111–122, 1995.

[15] D. Eppstein, *Average case analysis of dynamic geometric optimization*, *Comput. Geom. Theory Appl.*, 6:45–68, 1996.

[16] D. Eppstein, *Incremental and decremental maintenance of planar width*, *J. Algorithms*, 37:570–577, 2000.

[17] D. Eppstein and J. Erickson, *Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions*, *Discrete Comput. Geom.*, 22:569–592, 1999.

[18] J. Erickson, *Klee's measure problem*, http://compgeom.cs.uiuc.edu/~jeffe/open/klee.html, 1998.

[19] H. Freeman and R. Shapira, *Determining the minimum-area encasing rectangle for an arbitrary closed curve*, *Commun. ACM*, 18:409–413, 1975.

[20] J.-M. Ho, D. T. Lee, C.-H. Chang, and C. K. Wong, *Minimum diameter spanning trees and related problems*, *SIAM J. Comput.*, 20:987–997, 1991.

[21] J. Matoušek, *Efficient partition trees*, *Discrete Comput. Geom.*, 8:315–334, 1992.

[22] J. Matoušek, *Reporting points in halfspaces*, *Comput. Geom. Theory Appl.*, 2:169–186, 1992.

[23] J. Matoušek, *Range searching with efficient hierarchical cuttings*, *Discrete Comput. Geom.*, 10:157–182, 1993.

[24] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, 1984.

[25] M. H. Overmars, *The Design of Dynamic Data Structures*, Lect. Notes in Comput. Sci., vol. 156, Springer-Verlag, Heidelberg, 1983.

[26] M. Overmars and C.-K. Yap, *New upper bounds in Klee's measure problem*, *SIAM J. Comput.*, 20:1034–1045, 1991.

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

[28] M. Smid, *Closest-point problems in computational geometry*, in Handbook of Computational Geometry J. Urrutia and J. Sack, eds., North-Holland, Amsterdam, 2000, pp. 877–935.