Dynamic Independent Set of Disks (and Hypercubes) Made Easier

Sujoy Bhore[*] Timothy M. Chan[†]

Abstract

Maintaining an approximate maximum independent set of a dynamic collection of objects has been studied extensively in the past few years. Recently, Bhore, Nöllenburg, Tóth, and Wulms (SoCG 2024) showed that for (unweighted) disks in the plane, it is possible to maintain $O(1)$-factor approximate solution in polylogarithmic amortized update time. In this work, we provide a *much simpler* dynamic $O(1)$-approximation algorithm, which at the same time improves the number of logarithmic factors in the previous update time bound.

Along the way, we also obtain simpler and faster algorithms for dynamic independent set for axis-aligned hypercubes as well as static independent set for fat objects in any constant dimension. All the above results also hold for the minimum piercing set problem for the same classes of objects.

1 Introduction

Let S be a collection of objects in $\mathbb{R}^d$. The geometric maximum independent set (MIS) problem asks for a maximum-cardinality subset $I \subseteq S$ of *independent* (i.e., pairwise-disjoint) objects. MIS is a fundamental geometric optimization problem and among the most well-studied problems in computational geometry. Besides its theoretical importance and connection to other fundamental optimization problems, e.g., piercing set, vertex cover, etc., MIS has a wide range of applications in scheduling [BYHN+06], VLSI design [HM85], map labeling [AvKS98], data mining [KMP98, BDMR01a], and many others.

MIS is already NP-hard for a set of unit disks in the plane [CCJ90]. However, for near-equal-sized fat objects for any constant d, Hochbaum and Maass's shifted grid method yields a PTAS [HM85]. Moreover, for fat objects of arbitrary sizes, a simple greedy algorithm computes an $O(1)$-approximation [EKNS00] in $O(n^2)$ time (although with appropriate geometric data structures, the running time can be improved to near linear in $\mathbb{R}^2$ [EKNS00], or subquadratic in $\mathbb{R}^d$). Later, several PTASs with running time $n^{O(1/\varepsilon^d)}$ or $n^{O(1/\varepsilon^{d-1})}$ have been found, e.g., via shifted quadtrees, geometric separators, or local search [EJS05, Cha03, CH12]. The case of arbitrary rectangles[1] in $\mathbb{R}^2$ has also been extensively studied. After a long line of work [AvKS98, BDMR01b, CC09, AHW19, CE16], Mitchell [Mit22] obtained a polynomial-time $O(1)$-approximation algorithm for rectangles. Subsequently, Gálvez et al. [GKM+21] improved the approximation factor to close to 2. Very recently, Bhore and Chan [BC25] designed near-linear ($O(n^{1+\delta})$) time constant factor approximation algorithms for rectangles in $\mathbb{R}^2$ as well as for arbitrary-sized fat objects in $\mathbb{R}^d$ for any constant d (including balls and hypercubes).

Dynamic MIS. In dynamic settings, objects are inserted into or deleted from the collection S over time. The objective is to achieve (almost) the same approximation ratio as in the static case while keeping the update time as small as possible. In recent years, the dynamic version of the problem has received considerable attention. We briefly summarize the key prior results:

- For intervals in $\mathbb{R}^1$, Henzinger, Neumann, and Wiese [HNW20] gave the first fully dynamic $(1+\varepsilon)$-approximation algorithm with $O(\log^2 n \log^2 U)$ amortized update time (ignoring dependencies on ε),

[*]Department of Computer Science & Engineering, Indian Institute of Technology Bombay, Mumbai, India. Email: sujoy@cse.iitb.ac.in
[†]Department of Computer Science, University of Illinois at Urbana-Champaign. Email: tmc@illinois.edu.
Work supported in part by NSF Grant CCF-2224271.
[1]All rectangles, boxes, squares, and hypercubes are axis-aligned by default in this paper

assuming that all intervals are in $[0, U]$ and have minimum length at least 1. Bhore, Cardinal, Iacono, and Koumoutsos [BCIK21] improved the update time to $O(\log n)$ and made the bound worst-case. (The weighted setting of the problem was also considered by Henzinger et al. [HNW20], and subsequently by Compton et al. [CMR23].)

- For squares in $\mathbb{R}^2$ and more generally hypercubes in $\mathbb{R}^d$ for any constant d, Henzinger et al. [HNW20] gave a dynamic $O(1)$-approximation algorithm with $O(\log^{2d+1} n \log^{2d+1} U)$ amortized update time, again assuming that all objects are in $[0, U]^d$ and have minimum side length at least 1. Bhore et al. [BCIK21] improved the update time to $O(\log^{2d+1} n)$ while keeping approximation ratio $O(1)$.

  Note that it is not possible to maintain a $(1 + \varepsilon)$-approximation in sublinear time for a sufficiently small ε, even in the case of unit squares (or unit disks) in $\mathbb{R}^2$, since the static problem has a lower bound of $n^{\Omega(1/\varepsilon)}$ under ETH [Mar07].

- For disks in $\mathbb{R}^2$, Bhore, Nöllenburg, Tóth, and Wulms [BNTW24] recently presented the first dynamic $O(1)$-approximation algorithm with polylogarithmic expected amortized update time. Their algorithm is quite involved (the full paper is 37 pages long), and it requires a data structure for dynamic lower envelopes of surfaces in $\mathbb{R}^3$ by Kaplan et al. [KMR+20] (which generalizes Chan's data structure for dynamic lower envelopes of planes in $\mathbb{R}^3$ [Cha10, Cha20]). The update time of the data structure is close to $O(\log^9 n)$.

- For rectangles in $\mathbb{R}^2$, Henzinger et al. [HNW20] gave a dynamic $O(\log U)$-approximation algorithm with $O(\log^2 n \log^5 U)$ amortized update time. Bhore and Chan [BC25] recently obtained a dynamic algorithm with an improved approximation ratio of $O(1)$ and with $O(n^\delta)$ amortized update time for an arbitrarily small constant $\delta > 0$ (the approximation ratio is polynomial in $1/\delta$).

- For arbitrary fat objects in $\mathbb{R}^d$, Cardinal, Iacono, and Koumoutsos [CIK21] designed dynamic $O(1)$-approximation algorithms with sublinear worst-case update time, but the exponent in the update bound converges to 1 as d increases. Bhore and Chan [BC25] recently gave a substantially faster dynamic $O(1)$-approximation algorithm with $O(n^\delta)$ amortized update time for any constant $\delta > 0$. Note that in the specific case of disks in $\mathbb{R}^2$, this update time bound is worse than the polylogarithmic bound of Bhore et al. [BNTW24], although Bhore and Chan's algorithm is simple and more general (it also works for the weighted setting).

Main new result. Our main result is a new dynamic $O(1)$-approximation algorithm for MIS for the case of disks in $\mathbb{R}^2$ with polylogarithmic amortized update time (Theorem 4.1). We are able to describe our entire algorithm and analysis in under 5 pages, thus significantly simplifying Bhore et al.'s previous work [BNTW24]. At the same time, we also (unintentionally) improve their update time, from about $O(\log^9 n)$ to $O(\log^6 n)$ (and avoiding randomization as well). In particular, we are able to replace their use of Kaplan et al.'s data structure for dynamic lower envelope of surfaces in $\mathbb{R}^3$ [KMR+20], with Chan's original data structure for lower envelope of planes in $\mathbb{R}^3$ [Cha20], which requires fewer of logarithmic factors.

More precisely, the data structure we maintain is just a version of the well-known *shifted balanced quadtree* [AMN+98, Cha98], where each node is augmented with an instance of Chan's lower-envelope data structure. We show that given this augmented balanced quadtree, one can recompute an $O(1)$-approximation of the MIS in output-sensitive time by a simple recursive algorithm—the key idea to ensure $O(1)$-approximation (instead of $O(\log n)$-approximation) is to skip over long paths of "degree-1" nodes in the recursion. Combined with a standard trick of lazy updating and periodic rebuilding (namely, we don't need to recompute a solution till after about δ OPT updates,[2] since OPT changes by at most 1 per update), we obtain polylogarithmic amortized update time.

We should mention that all these ideas individually have appeared in some form or another in prior work. For example, Bhore et al.'s algorithm [BNTW24] also used some (less conventional) variant of quadtrees ("nonatrees") and shifting, and they also dealt with the same issue of skipping over paths of "degree-1" nodes, but in a far more complicated way—the complication arose partly because they did not incorporate the lazy updating trick (dynamization of their method required maintaining multiple invariants at the tree nodes about

[2]We use OPT to denote the optimal value for the optimization problem at hand.

"obstacle cells", "barrier clearance disks", etc., which we completely bypass). On the other hand, the lazy updating trick itself has been used in many previous dynamic algorithms, e.g., for geometric MIS [CIK21], and geometric piercing, vertex cover, matching, etc. [AHRS24, BC25]. Our contribution lies in how we distill the essential ideas and put them together in a way that strives for simplicity and elegance.

We have not worked out or tried to optimize the precise constant in the approximation ratio, to keep the presentation simple, although Bhore et al. [BNTW24] have not either, and we believe the constant should be similar in magnitude to theirs (or perhaps smaller because our algorithm and analysis are simpler).

Other new results. Our approach is not limited to MIS for disks, but generalizes to other classes of fat objects (with time bounds depending on the class), and it leads to a number of other interesting consequences:

- For hypercubes in $\mathbb{R}^d$ for any constant d, we obtain a dynamic $O(1)$-approximation algorithm for MIS with $O(\log n)$ amortized update time (Theorem 4.2), which substantially reduces the number of logarithmic factors in the previous methods by Henzinger et al. [HNW20] and Bhore et al. [BCIK21], from $O(d)$ all the way to one! (To be fair, the constant in our approximation ratio is worse than Henzinger et al.'s [HNW20]. Ours is $O(1)^d$, whereas Henzinger et al.'s approached 2^d, the approximation ratio of the standard static greedy algorithm.)

- As a byproduct, we also obtain a new *static* $O(1)$-approximation algorithm for arbitrary fat objects in $\mathbb{R}^d$ running in $O(n \log n)$ time (Theorem 4.3). This is faster than the standard greedy algorithm [EKNS00]. It is also slightly faster than Bhore and Chan's recent $O(n^{1+\delta})$-time algorithm [BC25]. (To be fair, Bhore and Chan's algorithm generalized to the weighted setting, whereas our approach does not. They also gave a faster, Monte Carlo randomized variant of their algorithm with $O(n \operatorname{polylog} n)$ running time, but this variant can only compute an approximation of the optimal value and not the independent set itself.)

- The same approach can also solve the *minimum piercing set (MPS)* problem: finding a minimum-cardinality set of points $Q \subset \mathbb{R}^d$ such that every object of S is pierced by (i.e., contains) at least one point of Q. MPS is closely related to MIS: it is easy to see that the *piercing number* (the size of the minimum piercing set) is always at least the *independence number* (the size of the maximum independent set); for fat objects, they are known to be within a constant factor of each other [EKNS00]. Our approach implies $O(1)$-approximation dynamic algorithms for MPS for disks in $\mathbb{R}^2$ with $O(\log^6 n)$ amortized update time and for hypercubes in $\mathbb{R}^d$ with $O(\log n)$ amortized update time. These are slightly faster than Bhore and Chan's recent dynamic MPS algorithms with $O(n^\delta)$ amortized update time [BC25] (and significantly faster than Agarwal, Har-Peled, Raychaudhury, and Sintos's previous dynamic MPS algorithm for squares in $\mathbb{R}^2$ with $O(n^{1/3} \operatorname{polylog} n)$ amortized update time [AHRS24]).

2 Preliminaries

We will describe our algorithm in a general setting, for families of objects satisfying the following properties:

- We assume that the objects are *fat*, in the following sense: for any hypercube B, there exist $O(1)$ points piercing all objects that intersect B and have L_∞-diameter at least $\operatorname{diam}_\infty(B)$. (Throughout the paper, $\operatorname{diam}_\infty(s)$ denotes the L_∞-diameter of an object s.) Disks, balls, and hypercubes fulfill this definition of fatness.

- We assume that each object has constant description complexity.

- We assume that there is an efficient data structure $\mathcal{DS}_0$ for storing a set Z of n objects, with $P_0(n)$ preprocessing time and $U_0(n)$ amortized update time, so that given any query box B, we can find an object of Z completely inside B (if exists), and an object of Z completely outside B (if exists), in $Q_0(n)$ time. (For disks, the "outside" case turns out to be more challenging than the "inside" case, as we will see later.)

Our data structure is an augmented version of a standard geometric data structure: shifted balanced quadtrees [Cha98] (related to the balanced-box decomposition trees of Arya et al. [AMN+98]). We review basic definitions below:

DEFINITION 1. *A* quadtree box *is a hypercube of the form* $[\frac{i_1}{2^\ell}, \frac{i_1+1}{2^\ell}) \times \cdots \times [\frac{i_d}{2^\ell}, \frac{i_d+1}{2^\ell})$ *for some integers* $i_1, \ldots, i_d, \ell$.

DEFINITION 2. *An object s is c_0*-good *if s is contained in a quadtree box B with* $\mathrm{diam}_\infty(B) \le c_0 \cdot \mathrm{diam}_\infty(s)$.

Goodness is desirable: good fat objects intersecting the boundary of a quadtree box B must have L_∞-diameter at least $\mathrm{diam}_\infty(B)/c_0$, and so can be pierced by $O(1)$ points (since ∂B can be covered by $O(1)$ hypercubes of L_∞-diameter $\mathrm{diam}_\infty(B)/c_0$).

Goodness can be guaranteed by shifting:

FACT 2.1. (SHIFTING LEMMA [CHA98, CHA03]) *Suppose d is even. Let $v_j = (\frac{j}{d+1}, \ldots, \frac{j}{d+1}) \in \mathbb{R}^d$. For every object $s \subset [0,1)^d$, there exists $j \in \{0, \ldots, d\}$ such that $s + v_j$ is $O(d)$-good.*

Quadtrees may generally have large height, but a balanced version with $O(\log n)$ height can be obtained by extending the shapes of cells (to allow for one "hole"), using the fact below:

DEFINITION 3. *A* quadtree cell *is defined as either a quadtree box or the difference of an "outer" quadtree box and an "inner" quadtree box.*

FACT 2.2. ([AMN$^+$98]) *Given $n > 1$ points inside a quadtree cell γ, we can decompose γ into $O(1)$ disjoint quadtree subcells, such that each subcell contains at most αn points for some constant $\alpha < 1$, in $O(n)$ time.*[3]

Proof. (Sketch) For completeness, we include a quick proof sketch. Let B_0 be a minimal quadtree box that contains at least $\frac{1}{2^d+1}n$ points (this corresponds to a "centroid" in the quadtree; see [AMN$^+$98] on how to find B_0 in $O(n)$ time). At most $\frac{2^d}{2^d+1}n$ points are outside B_0. Furthermore, B_0 can be covered by 2^d quadtree subboxes, each of which contains at most $\frac{1}{2^d+1}n$ points by minimality of B_0. Thus, at most $\frac{2^d}{2^d+1}n$ points are inside B_0. So, ∂B_0 subdivides γ into two cells each with at most αn points with $\alpha := \frac{2^d}{2^d+1}$.

However, one of the cells may have more than one hole if B_0 and the inner box of γ are disjoint. To fix this, we can take the smallest quadtree box B containing B_0 and γ, and subdivide B into 2^d quadtree subboxes. Then ∂B_0 together with the boundaries of these 2^d subboxes subdivide γ into at most $2^d + 1$ quadtree cells (each of which indeed has at most one hole). ☐

For a given point set P, we can build a tree by applying Fact 2.2 and recursively building a subtree for the points inside each quadtree subcell, until each leaf cell contains one point. This tree is referred to as the *balanced quadtree* for P, which indeed has height $O(\log n)$.

For each object s, designate an arbitrary point inside s as the "center" of s. For a given set S of objects, we build a balanced quadtree T for the set of all centers of S. For each quadtree cell γ in T:

- Let S_γ denote the subset of all objects of S that are (completely) contained in γ.
- Let $Z_\gamma = S_\gamma - \bigcup_{\text{child } \gamma' \text{ of } \gamma} S_{\gamma'}$. Note that each object s belongs to Z_γ for exactly one cell γ of T (namely, γ corresponds to the lowest node for which $s \in S_\gamma$). We store each Z_γ in an instance of the data structure $\mathcal{DS}_0$.

We say that a cell γ is *empty* if $S_\gamma = \emptyset$. We don't explicitly store S_γ: we just maintain a flag to indicate whether it is empty, and if not, store an arbitrary single object in S_γ. This entire data structure will be referred to as the *augmented balanced quadtree* for S, and it can be built in $O(n \log n + P_0(n))$ time.[4]

LEMMA 2.1. *We can maintain an augmented balanced quadtree for n objects in a constant dimension, under insertions and deletions in $O(U_0(n) + \frac{P_0(n)}{n} \log n + \log^2 n)$ amortized time.*

[3]As in a number of previous work, we assume a model of computation that supports taking the bitwise exclusive-or of two numbers and computing the most significant bit of a number in constant time.

[4]We assume that $P_0(n)/n$ is nondecreasing.

Proof. To insert/delete an object s, we find the unique cell γ with $s \in Z_\gamma$, by descending along a path of the tree T in $O(\log n)$ time; we can then update Z_γ and its corresponding data structure $\mathcal{DS}_0$ in $O(U_0(n))$ time. We can maintain the flags by updating along a path of the tree.

To ensure that balance is maintained after insertion/deletion, we can use a standard weight-balancing scheme: when the number of centers in a child cell exceeds $\alpha + \delta$ times the number of centers in the parent cell (for a sufficiently small constant $\delta > 0$), we rebuild the entire data structure at the parent. The amortized cost of rebuilding is at most $O(\frac{n \log n + P_0(n)}{n})$ for the cells at one level of the tree; there are $O(\log n)$ levels. ∎

REMARK 1. In the appendix, we show that a more careful scheme to maintain balance can lower the $\log^2 n$ term to $\log n$. Also, if we assume input coordinates are integers bounded by U and we are happy with $\log U$ factors instead of $\log n$ factors, then the data structure can be simplified: we can just use the original quadtree, which has $O(\log U)$ height; there is no need for quadtree cells with holes, and no extra work to maintain balance.

3 Main Algorithm

We now show that given the augmented balanced quadtree, it is possible to compute an approximate maximum independent set *from scratch* in time sensitive to OPT. The quadtree naturally suggests a simple divide-and-conquer algorithm, but one could lose a logarithmic factor in the approximation ratio if not careful. The key idea is to skip over paths of "degree-1" nodes during recursion (as will be made precise below).

LEMMA 3.1. *Given an augmented balanced quadtree for n $O(1)$-good fat objects in any constant dimension d, we can compute an $O(1)$-approximation of the maximum independent set in $O(\mathrm{OPT} \cdot Q_0(n) \log^2 n)$ time.*

Proof. Given a nonempty cell γ, the following algorithm computes an independent set for S_γ:

```
INDEP-SET($\gamma$):
0.  let $\gamma_0 = \gamma$ and $i = 0$
1.  repeat {
2.     if $\gamma_i$ has at least two nonempty children $\gamma'$ then
3.        return $\bigcup_{\text{nonempty child } \gamma' \text{ of } \gamma_i}$ INDEP-SET($\gamma'$)
4.     if $\gamma_i$ has no nonempty child then
5.        return any single object of $S_\gamma$
6.     let $\gamma'$ be the unique nonempty child of $\gamma$
7.     if there is an object $s \in Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$ not intersecting $\gamma'$ then
8.        return INDEP-SET($\gamma'$) $\cup \{s\}$
9.     let $\gamma_{i+1} = \gamma'$ and increment $i$
    }
```

The algorithm generates a path $\gamma_0, \ldots, \gamma_i$, starting at the given node γ_0, and stopping when we have found a node γ_i with at least two nonempty children (line 2) or none (line 4), or when γ_i has one nonempty child γ' and there exists an object $s \in Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$ not intersecting γ' (line 7). The algorithm then recursively explores the nonempty child(ren) of γ_i (line 3 or 8). See Figure 1.

To analyze the algorithm, we define a new rooted tree $\widehat{T}$: if we arrive at line 3, we define the children of γ in $\widehat{T}$ to be all the nonempty children γ' of γ_i; if we arrive at line 5, we make γ a leaf in $\widehat{T}$ storing the single object returned; if we arrive at line 8, we define the children of γ in $\widehat{T}$ to be γ' and a leaf storing the object s. The output independent set I is precisely the objects stored in the leaves of this tree $\widehat{T}$. All internal nodes of $\widehat{T}$ have degree at least 2. Thus, the number of nodes in $\widehat{T}$ is at most twice the number of leaves, i.e., $2|I|$.

The condition in line 7 can be tested by querying the data structure $\mathcal{DS}_0$ for Z_{γ_j} (finding an object in Z_{γ_j} completely outside the outer box of γ' or completely inside the inner box of γ'), for each $j \in \{0, \ldots, i\}$. The cost for line 7 is thus $O(i \cdot Q_0(n)) \le O(Q_0(n) \log n)$. There are at most $O(\log n)$ iterations of the repeat loop. Thus, the total time is $O(Q_0(n) \log^2 n)$ times the number of nodes in $\widehat{T}$, i.e., $O(|I| \cdot Q_0(n) \log^2 n)$.

Next, we analyze the approximation factor. All objects in $Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_{i-1}}$ intersect γ_i (because otherwise the repeat loop would have terminated in the previous iteration), and none of these objects are contained in γ_i (by definition of the Z_γ's), so they all must intersect $\partial\gamma_i$. Furthermore, all objects in Z_{γ_i} must intersect $\bigcup_{\text{child } \gamma' \text{ of } \gamma_i} \partial\gamma'$.

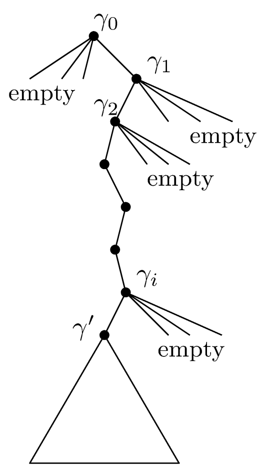

Figure 1: A path of "degree-1" nodes. If some object in $Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$ does not intersect γ', we recurse in γ'; otherwise, we extend the path.

Thus, by goodness and fatness, all objects in $Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_{i-1}} \cup Z_{\gamma_i}$ can be pierced by $O(1)$ points. So, the overall piercing number is at most $O(1)$ times the number of nodes of $\widehat{T}$, i.e., $O(1) \cdot |I|$. The independence number OPT is at most the piercing number and is thus at most $O(1) \cdot |I|$. ◻

Combining with the standard idea of lazy updating and periodic rebuilding [CIK21, AHRS24, BC25], we obtain a dynamic algorithm for independent set:

THEOREM 3.1. *For n fat objects in any constant dimension d, we can maintain an $O(1)$-approximation of the maximum independent set in $O(Q_0(n)\log^2 n + U_0(n) + \frac{P_0(n)}{n}\log n)$ amortized time.*

Proof. We assume that all objects are in $[0,1)^d$ (by rescaling) and are good. This is without loss of generality by the shifting lemma (Fact 2.1): for each of the $d+1$ shifts v_j ($j \in \{0,\ldots,d\}$), we can solve the problem for the good objects of $S + v_j$, and return the largest of the independent sets found (after shifting back by $-v_j$). The approximation ratio increases by a factor of $d+1 = O(1)$.

We divide the update sequence into phases. At the beginning of each phase, we rerun the algorithm in Lemma 3.1, to compute an $O(1)$-approximation in $O(\mathrm{OPT}\cdot Q_0(n)\log^2 n)$ time. Let I be the returned independent set. For the next $\delta|I|$ updates (for some sufficiently small constant $\delta > 0$), we do nothing to the solution. Since an update changes the optimal value by at most 1, this incurs an additive error of $O(\delta\,\mathrm{OPT})$, which we can tolerate. The amortized update time is $O(\frac{|I|\cdot Q_0(n)\log^2 n}{\delta|I|}) = O(Q_0(n)\log^2 n)$. After $\delta|I|$ updates, we start a new phase.

In addition, the augmented quadtree is updated after every insertion/deletion in $O(U_0(n) + \frac{P_0(n)}{n}\log n + \log^2 n)$ amortized time by Lemma 2.1. ◻

4 Applications

We now apply Theorem 3.1 to obtain dynamic independent set algorithms for specific families of objects, notably, disks in $\mathbb{R}^2$ and fat boxes (e.g., hypercubes) in $\mathbb{R}^d$, by providing the required data structure $\mathcal{DS}_0$ in these cases. We also discuss implications of the approach to static algorithms as well as to the piercing set problem.

4.1 Disks.

LEMMA 4.1. *There is a data structure $\mathcal{DS}_0$ for storing a set Z of n disks in $\mathbb{R}^2$, with $P_0(n) = O(n\log^4 n)$ preprocessing time and $U_0(n) = O(\log^6 n)$ amortized update time, so that we can find a disk of Z completely inside/outside a query axis-aligned rectangle in $Q_0(n) = O(\log^4 n)$ time.*

Proof. Finding a disk $s \in Z$ completely inside a rectangle B is equivalent to finding an $s \in Z$ whose minimum bounding rectangle is contained in B: this is a rectangle enclosure query, and reduces to orthogonal range searching in $\mathbb{R}^4$, after mapping rectangles in $\mathbb{R}^2$ to points in $\mathbb{R}^4$. So, the problem can be solved with $O(\log^4 n)$ update and query time by standard range trees [dBCvKO08] (or better, by more advanced techniques [CT18]).

Finding a disk $s \in Z$ completely outside a rectangle $B = [x_1, x_2) \times [y_1, y_2)$ is more challenging. Let (a_s, b_s) denote the center of s, and r_s denote its radius. We break into cases:

- Case 1: $(a_s, b_s) \in [x_1, x_2] \times [y_2, \infty)$. This case is equivalent to finding an $s \in Z$ with $(a_s, b_s - r_s) \in [x_1, x_2) \times [y_2, \infty)$. This reduces to orthogonal range searching in $\mathbb{R}^2$.
- Case 2: $(a_s, b_s) \in [x_2, \infty] \times [y_2, \infty)$. Under this constraint, s is completely outside B iff s does not contain the point (x_2, y_2), or equivalently, the lifted plane $\pi_s := \{(x, y, z) : z - 2a_s x - 2b_s y + a_s^2 + b_s^2 = r_s^2\} \subset \mathbb{R}^3$ is below the lifted point $(x_2, y_2, x_2^2 + y_2^2) \in \mathbb{R}^3$.

  Chan [Cha10, Cha20] gave a data structure for dynamic lower envelopes of planes in $\mathbb{R}^3$, which maintains a set of n planes in $\mathbb{R}^3$, with $O(n \log^2 n)$ preprocessing time and $O(\log^4 n)$ amortized update time, so that we can find a plane below a query point in $O(\log^2 n)$ time. We combine with a standard *multi-level* data structure technique [AE99]: we first use a 2D range tree to pick out all s with $(a_s, b_s) \in [x_2, \infty) \times [y_2, \infty)$, expressed as a union of *canonical subsets*; for each canonical subset, we store Chan's data structure. All the time bounds increase by 2 logarithmic factors.
- All other cases are symmetric.

$\square$

THEOREM 4.1. *For n disks in $\mathbb{R}^2$, we can maintain an $O(1)$-approximation of the maximum independent set in $O(\log^6 n)$ amortized time.*

4.2 Fat boxes in $\mathbb{R}^d$. For boxes in any constant dimension, we can implement the data structure $\mathcal{DS}_0$ easily in polylogarithmic time by reduction to orthogonal range searching; this immediately implies polylogarithmic overall update time. However, we note an alternative implementation with much fewer logarithmic factors, by observing that the algorithm in the proof of Theorem 3.1 only requires a special case of $\mathcal{DS}_0$ with the following restrictions:

1. the query boxes B are all quadtree boxes, and
2. the objects all intersect the boundaries of $O(1)$ quadtree boxes (this is because the objects of each Z_{γ_i} all intersect the boundary of a child cell of γ_i).

LEMMA 4.2. *For a set Z of n axis-aligned boxes in $\mathbb{R}^d$, all intersecting the boundary of a fixed quadtree box B_0, there is a data structure $\mathcal{DS}_0$, with $P_0(n) = O(n)$ preprocessing time and $U_0(n) = O(\log n)$ update time, so that we can find a box of Z completely inside/outside a query quadtree box in $Q_0(n) = O(1)$ time.*

Proof. Finding a box $s \in Z$ completely inside a query quadtree box B: If B_0 is disjoint from B, then the answer does not exist. Thus, we may assume that B_0 is contained in B. Let ℓ_s denote the L_∞-diameter of the smallest quadtree box containing s. The problem is equivalent to finding an $s \in Z$ with $\ell_s \le \mathrm{diam}_\infty(B)$. This reduces to finding the box $s \in Z$ with the minimum ℓ_s, and can be solved using a standard binary heap.

Finding a box $s \in Z$ completely outside a query box $B = [x_1^-, x_1^+) \times \cdots \times [x_d^-, x_d^+)$: Let $s = [a_{s,1}^-, a_{s,1}^+) \times \cdots \times [a_{s,d}^-, a_{s,d}^+)$. This problem is equivalent to finding an $s \in Z$ such that $a_{s,j}^- \ge x_j^+$ or $a_{s,j}^+ \le x_j^-$ for some $j \in \{1, \ldots, d\}$. This reduces to finding the box $s \in Z$ with the maximum $a_{s,j}^-$, and finding the box $s \in Z$ with the minimum $a_{s,j}^+$, for every j, and can again be solved using heaps. $\square$

Applying Theorem 3.1 immediately yields $O(\log^2 n)$ amortized time for maintain an $O(1)$-approximation of the maximum independent set for fat boxes.

We can do still better by observing that line 7 in the algorithm from Theorem 3.1 can be implemented in $O(1)$ time instead of $O(Q_0(n) \log n) = O(\log n)$: As in the proof of Lemma 4.2, the answer to the query in line 7 can be determined from the minimum/maximum of certain values (ℓ_s, $a_{s,j}^+$, and $a_{s,j}^-$) over all $s \in Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$. We can look up the minimum/maximum values over all $s \in Z_{\gamma_i}$ from the heaps for Z_{γ_i}, and so can compute the minimum/maximum values over all $s \in Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$ in $O(1)$ time, when given the minimum/maximum values over all $s \in Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_{i-1}}$ computed from the previous iteration. The total running time for Lemma 3.1 is thus lowered from $O(\log^2 n)$ to $O(\log n)$. The overall update time for Theorem 3.1 is then lowered from $O(\log^2 n)$ to $O(\log n)$, if we use the refined version of Lemma 2.1 from the appendix.

THEOREM 4.2. *For n axis-aligned fat boxes (e.g., hypercubes) in any constant dimension, we can maintain an $O(1)$-approximation of the maximum independent set in $O(\log n)$ amortized time.*

REMARK 2. The approximation ratio of our algorithm is of the order of $O(d)^d$, but can be lowered to $O(1)^d$ if we switch to Bern's version of the shifting lemma [Ber93]; this version uses 2^d shifts but achieves goodness $O(1)$ independent of d.

4.3 Static case. In the static setting, we don't even need the data structure $\mathcal{DS}_0$. In line 7 of the algorithm in Lemma 3.1, we can just do linear search in $Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$; we can charge each object of $Z_{\gamma_0} \cup \cdots \cup Z_{\gamma_i}$ one unit to cover the cost. Each object is charged $O(\log n)$ times over the entire algorithm. Thus, the total running time is $O(n \log n)$. (The division into phases in the proof of Theorem 3.1 is not needed.)

THEOREM 4.3. *For n fat objects (with constant description complexity) in any constant dimension, we can compute an $O(1)$-approximation of the maximum independent set in $O(n \log n)$ time.*

4.4 Piercing. The same approach works for the minimum piercing set problem. The proof of Lemma 3.1 already indicates how to obtain $O(1) \cdot |I| \le O(1) \cdot \mathrm{OPT}$ piercing points from the algorithm. In the proof of Theorem 3.1, we can return the union of the piercing sets found for each shift; the approximation ratio again increases by a $d+1$ factor. And during a phase, we can lazily add one piercing point for each new object inserted, again resulting in $O(\delta\, \mathrm{OPT})$ additive error.

As a consequence, we get an $O(1)$-approximation dynamic algorithm for piercing for disks in $\mathbb{R}^2$ with $O(\log^6 n)$ amortized update time, or for fat boxes (e.g., hypercubes) in $\mathbb{R}^d$ with $O(\log^2 n)$ amortized update time. We also get an $O(1)$-approximation static algorithm for arbitrary fat objects running in $O(n \log n)$ time.

References

[AE99] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999.

[AHRS24] Pankaj K. Agarwal, Sariel Har-Peled, Rahul Raychaudhury, and Stavros Sintos. Fast approximation algorithms for piercing boxes by points. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4892–4908, 2024. doi:10.1137/1.9781611977912.174.

[AHW19] Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019. doi:10.1145/3326122.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998. doi:10.1145/293347.293348.

[AvKS98] Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.

[BC25] Sujoy Bhore and Timothy M. Chan. Fast static and dynamic approximation algorithms for geometric optimization problems: Piercing, independent set, vertex cover, and matching. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, to appear, 2025. arXiv:2407.20659.

[BCIK21] Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. In *Abstracts of 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games (TJDCG)*, 2021. arXiv:2007.08643.

[BDMR01a] Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Efficient approximation algorithms for tiling and packing problems with rectangles. *J. Algorithms*, 41(2):443–470, 2001. doi:10.1006/jagm.2001.1188.

[BDMR01b] Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 427–436, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365496.

[Ber93] Marshall W. Bern. Approximate closest-point queries in high dimensions. *Inf. Process. Lett.*, 45(2):95–99, 1993. doi:10.1016/0020-0190(93)90222-U.

[BNTW24] Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 19:1–19:16, 2024. doi:10.4230/LIPICS.SOCG.2024.19.

[BYHN$^+$06] Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36(1):1–15, 2006. doi:10.1137/S0097539703437843.

[CC09] Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 892–901, 2009. doi:10.1137/1.9781611973068.97.

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

[CE16] Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 820–829, 2016. doi:10.1109/FOCS.2016.92.

[CH12] Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012. doi:10.1007/S00454-012-9417-5.

[Cha98] Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998. doi:10.1007/PL00009390.

[Cha03] Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

[Cha10] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.

[Cha20] Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discret. Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/S00454-020-00229-5.

[CIK21] Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 25:1–25:15, 2021. doi:10.4230/LIPICS.ESA.2021.25.

[CK95] Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 263–272, 1995. doi:10.5555/313651.313705.

[CMR23] Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 45:1–45:16, 2023. doi:https://doi.org/10.4230/LIPIcs.ICALP.2023.45.

[CT18] Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. *J. Comput. Geom.*, 9(2):45–66, 2018. doi:10.20382/JOCG.V9I2A5.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. URL: https://www.worldcat.org/oclc/227584184.

[EJS05] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

[EKNS00] Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0.

[Fre93] Greg N. Frederickson. A data structure for dynamically maintaining rooted trees. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 175–184, 1993. doi:10.1006/jagm.1996.0835.

[GKM$^+$21] Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A $(2+\varepsilon)$-approximation algorithm for maximum independent set of rectangles. *CoRR*, abs/2106.00623, 2021. arXiv:2106.00623.

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

[HNW20] Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020. doi:10.4230/LIPICS.SOCG.2020.51.

[KMP98] Sanjeev Khanna, Shan Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Symposium on Discrete Algorithms (SODA'98)*, pages 384–393, 1998. doi:10.5555/314613.314768.

[KMR$^+$20] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

[Mar07] Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–348, 2007. doi:10.1109/FOCS.2007.26.

[Mit22] Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 339–350, 2022. doi:10.1109/FOCS52979.2021.00042.

A Appendix: Maintaining Balance

In this appendix, we note a refinement of Lemma 2.1 that lowers the $\log^2 n$ term to $\log n$ (which is needed in Theorem 4.2 but nowhere else). While dynamic versions of balanced quadtrees have been proposed before (e.g., using dynamic topology trees) [CK95, Fre93], our situation is trickier, as we are also maintaining auxiliary information at the tree nodes (namely, the Z_γ sets). When a subtree becomes out of balance, our idea is to rebuild some parts of the subtree but keep some parts unchanged, rather than rebuilding the entire subtree from scratch.

LEMMA A.1. *We can maintain an augmented balanced quadtree for n objects in a constant dimension, under insertions and deletions in $O(U_0(n) + \frac{P_0(n)}{n}\log n + \log n)$ amortized time.*

Proof. Let $\alpha < 1$, c, and $0 < \delta < 1 - \alpha$ be parameters (constants) to be set later. We maintain the invariant that each cell γ of the tree T has degree at most c, and that $|P_{\gamma'}| \le (\alpha+\delta)|P_\gamma|$ for every child γ' of γ, where P_γ denotes the set of centers inside γ. Define the *imbalance* of γ to be $\max_{\text{child } \gamma' \text{ of } \gamma}(|P_{\gamma'}| - \alpha|P_\gamma|)$.

Suppose the invariant is violated at a root cell γ, but is true at all descendants of γ. Assume that γ has degree at most C, for another parameter $C > c$. We describe a procedure to *fix* γ:

1. First pick a great-grandchild γ^* of γ maximizing $|P_{\gamma^*}|$. Then
$$\tfrac{1}{Cc^2}|P_\gamma| \;\le\; |P_{\gamma^*}| \;\le\; (\alpha+\delta)^2|P_\gamma|.$$

2. Let Γ be the set of all siblings of γ^*, all siblings of the parent of γ^*, and all siblings of the grandparent of γ^*, together with γ^* itself. Then Γ consists of at most $2(c-1) + (C-1) + 1 < C + 2c$ quadtree cells. In the following steps, we will rearrange the nodes in Γ, to make γ^* a child of γ.

3. Pick an arbitrary point in each cell in Γ, and let Q_Γ be the resulting point set. Find a quadtree box B_0 such that $|Q_\Gamma \cap B_0|, |Q_\Gamma \setminus B_0| \le \frac{2^d}{2^d+1}|Q_\Gamma|$, as in the proof of Fact 2.2.

4. Let Ψ be a set of quadtree boxes, which include B_0 and the inner and outer boxes of γ^*, so that $\bigcup_{B\in\Psi}\partial B$ subdivides γ into at most c_1 quadtree cells, for some constant c_1 (depending only on d); see the proof of Fact 2.2 on how to ensure each cell has at most one hole.

5. The boxes in Ψ may not necessarily "align" with the boundaries of Γ. To rectify this, we modify Ψ into a new set $\widehat{\Psi}$: for each box $B \in \Psi$, if ∂B is already part of $\bigcup_{\gamma'\in\Gamma}\partial\gamma'$, then we just add B to $\widehat{\Psi}$; otherwise, ∂B is contained in a unique $\gamma' \in \Gamma$ and we add the inner and outer boxes of γ' to $\widehat{\Psi}$.

6. Now, $\bigcup_{B\in\widehat{\Psi}}\partial B$ subdivides γ into at most $2c_1$ quadtree cells (each cell still has at most one hole); we make these cells the new children of γ. Note that γ^* is one of these children. For each new child γ' of γ that contains more than one cell of Γ, we define the children of γ' to be all cells $\gamma'' \in \Gamma$ contained in γ'. We can compute Z_γ and $Z_{\gamma'}$ by a linear scan in $O(|S_\gamma|) \le O(|P_\gamma|)$ time, and can also preprocess the corresponding data structures $\mathcal{DS}_0$ in $O(P_0(|P_\gamma|))$ time. Note that γ has degree at most $2c_1 \le c$ and each new child γ' of γ has degree at most $\frac{2^d}{2^d+1}(C+2c) \le C$, by setting $c := 2c_1$ and $C := 2^{d+1}c$.

   The invariant is now satisfied at γ; in fact, γ now has imbalance 0, since $|P_{\gamma^*}| \le (\alpha+\delta)^2|P_\gamma| \le \alpha|P_\gamma|$ and every other child γ' of γ has $|P_{\gamma'}| \le (1 - \frac{1}{Cc^2})|P_\gamma| \le \alpha|P_\gamma|$, by setting $\alpha := 1 - \frac{1}{Cc^2}$ and $\delta := \sqrt{\alpha} - \alpha$.

7. However, the invariant may be violated at some of the children of γ. For each child γ' of γ other than γ^*, we recursively fix γ'. We leave the subtree at γ^* unchanged.

The running time for fixing a cell γ with $|P_\gamma| = m$ satisfies the recurrence
$$T(m) \;\le\; \max_{m_i \text{ with } \sum_i m_i \le (1-1/(Cc^2))m} T(m_i) \;+\; O(m + P_0(m)),$$
which solves to $T(m) = O(m + P_0(m))$.

We handle each insertion and deletion in $O(\log n + U_0(n))$ time as in the proof of Lemma 2.1. Whenever the invariant is violated at some cell γ in T, we invoke the above procedure to fix γ in $O(|P_\gamma| + P_0(|P_\gamma|))$ time. As a result, the imbalance at γ decreases from $\delta|P_\gamma|$ to 0, while the imbalance at every other cell in T either decreases to 0 or stays the same. Thus, the total imbalance decreases by $\Omega(|P_\gamma|)$, and we can bound the fixing cost by the decrease in total imbalance times $O(1 + \frac{P_0(|P_\gamma|)}{|P_\gamma|})$. Since each insertion/deletion increases the total imbalance by at most $O(\log n)$, the amortized fixing cost is at most $O(\log n \cdot (1 + \frac{P_0(n)}{n}))$ per insertion/deletion. ∎