Worst-Case Efficient Dynamic Geometric Independent Set

Jean Cardinal
Université libre de Bruxelles (ULB), Brussels, Belgium

John Iacono
Université libre de Bruxelles (ULB), Brussels, Belgium

Grigorios Koumoutsos
Université libre de Bruxelles (ULB), Brussels, Belgium

Abstract

We consider the problem of maintaining an approximate maximum independent set of geometric objects under insertions and deletions. We present a data structure that maintains a constant-factor approximate maximum independent set for broad classes of fat objects in d dimensions, where d is assumed to be a constant, in sublinear *worst-case* update time. This gives the first results for dynamic independent set in a wide variety of geometric settings, such as disks, fat polygons, and their high-dimensional equivalents.

Our result is obtained via a two-level approach. First, we develop a dynamic data structure which stores all objects and provides an approximate independent set when queried, with output-sensitive running time. We show that via standard methods such a structure can be used to obtain a dynamic algorithm with *amortized* update time bounds. Then, to obtain worst-case update time algorithms, we develop a generic deamortization scheme that with each insertion/deletion keeps (i) the update time bounded and (ii) the number of changes in the independent set constant. We show that such a scheme is applicable to fat objects by showing an appropriate generalization of a separator theorem.

Interestingly, we show that our deamortization scheme is also necessary in order to obtain worst-case update bounds: If for a class of objects our scheme is not applicable, then no constant-factor approximation with sublinear worst-case update time is possible. We show that such a lower bound applies even for seemingly simple classes of geometric objects including axis-aligned rectangles in the plane.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Data structures design and analysis

Keywords and phrases Maximum independent set, deamortization, approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.25

Related Version *Full Version*: `https://arxiv.org/abs/2108.08050`

Funding *John Iacono*: Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.
Grigorios Koumoutsos: Supported by the Fonds de la Recherche Scientifique-FNRS under a "Scientific Collaborator" grant.

Acknowledgements We would like to thank Timothy Chan and Qizheng He for pointing out an error in the previous version of this paper.

1 Introduction

Dynamic algorithms for classic NP-hard optimization problems is a quite active research area and major progress has been achieved over the last years (see [1, 9–11, 27]). For such problems, typically improved performance can be achieved when their *geometric* versions are considered, i.e., when the input has a certain geometric structure. As a result, a very recent line of research considers dynamic algorithms for well-known algorithmic problems in the geometric setting [3, 12, 18, 21, 29]. In this work, we continue this investigation by considering

the geometric version of the Maximum Independent Set problem; some of our results improve upon the previous (very recent) work of Henzinger et al. [29] and Bhore et al. [12] and some others provide the first dynamic data structures for various geometric instances.

Static Independent Set

In the maximum independent set problem, we are given a graph $G = (V, E)$ and we aim to produce a subset $I \subseteq V$ of maximum cardinality, such that no two vertices in I are adjacent. This is one of the most well-studied algorithmic problems. It is well-known to be NP-complete and hard to approximate: no polynomial time algorithm can achieve an approximation factor $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless P=NP [28, 36].

(Static) Geometric Independent Set

In the geometric version of the maximum independent set problem, the graph G is the intersection graph of a set V of geometric objects: each vertex corresponds to an object, and there is an edge between two vertices if and only if the corresponding objects intersect.

Besides the theoretical interest, the study of independent sets of geometric objects such as axis-aligned squares, rectangles or disks has been motivated by applications in various areas such as VLSI design [30], map labeling [4, 35] and data mining [8, 31].

For most classes of geometric objects, the problem remains NP-hard. A notable exception is the case where all objects are intervals on the real line (the well-known *interval scheduling* problem), where the optimal solution can be computed in polynomial time using a folklore greedy algorithm [25]. However, even the simplest 2d-generalization to axis-aligned unit squares is NP-hard [24]. As a result, most of the related work has focused on developing PTASs for certain versions of geometric independent set. This has been achieved for various types of objects such as squares, hypercubes, fat objects and pseudodisks [15, 16, 23, 30].

A substantially harder setting is the when the geometric objects are arbitrary (axis-aligned) rectangles in the plane: Despite the intense interest (see [2, 20]), no PTAS is known. Until very recently the best known approximation factor was $O(\log \log n)$ [13, 14]; in 2021, a $O(1)$-approximation was claimed [33].

Dynamic Geometric Independent Set

In the dynamic version of geometric independent set, V is a class of geometric objects (for instance squares in the plane) and we maintain a set $S \subseteq V$ of *active* objects. Objects of V may be inserted in S and deleted from S over time and the goal is to maintain a maximum independent set[1] of S, while keeping the time to implement the changes (the *update time*) sublinear on the size of the input.

Previous Work

Most previous work dates from 2020 onwards. The only exception is the work of Gavruskin et al. [26] who considered the very special case of intervals where no interval is fully contained in any other interval. The study of this area was revitalized by a paper of Henzinger, Neumann and Wiese in SoCG'20 [29]. They presented deterministic dynamic algorithms with approximation ratio $(1 + \epsilon)$ for intervals and $(1 + \epsilon) \cdot 2^d$ for d-dimensional hypercubes,

[1] This problem should not be confused by the related, but very different, *maximal* independent set problem, where the goal is to maintain an inclusionwise maximal independent set subject to updates on the edges (and not the vertices). This problem has been studied in the dynamic setting, with a remarkable recent progress after a sequence of breakthrough results [5–7, 19].

under the assumption that all objects are contained in $[0, N]^d$ and each of their edges has at least unit length. Their update time was polylogarithmic in both n and N, but N could be unbounded in n. Furthermore, they showed that no $(1+\epsilon)$-approximation scheme with sublinear update time is possible, unless the Exponential Time Hypothesis fails.

Subsequently Bhore et al. [12] obtained the first results for dynamic geometric independent set without any assumption on the input. For intervals, they presented a dynamic $(1+\epsilon)$-approximation with logarithmic update time; this was recently simplified and improved by Compton et al. [21]. For squares, [12] presented a randomized algorithm with expected approximation ratio roughly 2^{12} (generalizing to roughly 2^{2d+5} for d-dimensional hypercubes) with amortized update time $O(\log^5 n)$ (generalizing to $O(\log^{2d+1})$ for hypercubes).

Our Results

In this work we obtain the first dynamic algorithms for fat objects (formally defined in Section 2) in fixed dimension d with sublinear worst-case update time. The precise bounds on the approximation ratio and the update time depend on the fatness constant and the running time for worst-case range query structures for each family of objects. However, the results follow from a generic approach, applicable to all those classes of objects.

▶ **Theorem 1.** *There exists a deterministic data structure for maintaining a $O(1)$-approximate independent set of a collection of fat objects, with sublinear worst-case update time, that reports $O(1)$ changes in the independent set per update. In particular, it achieves the following approximation ratios and worst-case update times, for any constant $0 < \epsilon \leq 1/4$:*

- *$(4+\epsilon)$-approximation with $O(n^{3/4})$ update time for axis-aligned squares in the plane.*
- *$(5+\epsilon)$-approximation with $\tilde{O}(n^{3/4})$ update time for disks in the plane*[2]
- *$(2^d+\epsilon)$-approximation with $O(n^{1-\frac{1}{2d}})$ update time for d-dimensional hypercubes.*
- *$O(1)$-approximation with $O(n^{1-\frac{1}{2dk}})$ update time for fat objects which are unions of k (hyper)rectangles*
- *$O(1)$-approximation with $\tilde{O}(n^{1-\frac{1}{d(d+1)}})$ update time for fat simplices in d dimensions.*
- *$O(1)$-approximation with $\tilde{O}(n^{1-\frac{1}{d(d+1)k}})$ update time for fat objects which are unions of k simplices in d dimensions.*
- *$O(1)$-approximation with $\tilde{O}(n^{1-\frac{1}{d+2}})$ update time for balls in d dimensions.*

This result gives the first dynamic algorithms with sublinear update time for all the aforementioned classes of objects, apart from d-dimensional hypercubes. Moreover, for hypercubes our result is the first with sublinear *worst-case update time bounds* without assumptions on the input; it also achieves the same approximation ratio as [29], which is the best known for any dynamic setting.

In fact, it seems hard to significantly improve our result on any aspect: First, for the approximation factor, as mentioned above, Henzinger, Neumann and Wiese [29] proved that (under the Exponential Time Hypothesis) one cannot maintain a $(1+\epsilon)$-approximate maximum independent set of hypercubes in $d \geq 2$ dimensions with update time $n^{O((1/\epsilon)^{1-\delta})}$ for any $\delta > 0$; therefore the only potential improvement in the approximation ratio is by small constant factors. Moreover, the update time we obtain is essentially the time required for detecting intersections between objects in a range query data structure. Fatness of the objects is a sensible condition for achieving such results: we prove that for (nonfat) rectangles, ellipses, or arbitrary polygons, no dynamic approximation in sublinear worst-case update time is possible. Finally, we emphasize the remarkable additional property that the number of changes reported per update is always constant.

[2] Throughout this paper, $\tilde{O}$ suppresses polylogarithmic factors.

To obtain the result of Theorem 1 we develop a generic method to obtain dynamic independent set algorithms and show how to apply it for fat objects. Our method uses a combination of several components. The first is a data structure which stores all objects, supports insertions and deletions in sublinear $f(n)$ time and, when queried, returns a β-approximate independent set with output-sensitive running time. The second main component is a generic transformation of such a data structure into a dynamic algorithm with approximation factor $\beta + \epsilon$ and *amortized* update time $O(f(n))$. This is done by periodically updating the solution using the data structure. Then, we apply a generic *deamortization* scheme, involving what we call a *MIX* function, a generic way to switch "smoothly" from one solution to another. We design such a MIX function for fat objects using geometric separators.

2 Overview

Notation

In what follows, V is a class of geometric objects, such as squares in the plane, and we consider a finite set $S \subseteq V$. A subset of S is a *maximum independent set (MIS)* of S if all its objects are pairwise disjoint and it has maximum size over all sets with this property. We use OPT to denote the size of a maximum independent set. We use n to denote $|S|$, unless otherwise specified. We say that I is a β-approximate MIS if its size is at least $\beta\,\mathrm{OPT}$, for a constant $0 < \beta \leq 1$ [3].

The problem we study is to maintain an approximate MIS set I under a sequence of insertions and deletions in S, which we call generically an *update*, starting from an empty set. This can be expressed as implementing a single operation $\Delta = \textsc{Update}(u)$, where u is the object to be inserted or deleted, and Δ, the *update set*, is the set of objects that change in the approximate MIS I, presented as the symmetric difference from the previous set. In general we will use subscripts to express variables' states after the indicated update, and unsubscripted versions for the current state. Using the operator $\oplus$ to denote the symmetric difference, we therefore have $S_i := \oplus_{j=1}^{i}\{u_j\}$ and $I_i := \oplus_{j=1}^{i}\Delta_j$. We say that Update is a β-approximate MIS algorithm if I_i is always a β-approximate MIS of S_i. We adopt the convention that the update set Δ_i is always returned explicitly and thus the running time of an update u_i is at least the size of the returned update set, $|\Delta_i|$.

We begin with a simple yet crucial observation about MIS.

▶ **Fact 2.** *The size of a MIS can change by at most one with every update:* $|\,\mathrm{OPT}_i - \mathrm{OPT}_{i-1}\,| \leq 1$.

Note that this fact does not hold for the weighted version of the MIS problem. Also note that this does not say that it is possible to have an update algorithm with an update set Δ_i with cardinality always at most 1; this fact bounds how the size of a MIS can change after an update, and not the content. In fact, it is easy to produce examples where $|\Delta_i|$ must be 2, even for intervals (e.g. $u_1 = [1, 4]$, $u_2 = [2, 3]$, $u_3 = [1, 4]$). However, it does leave open the possibility to have an update operation returning constant-size update sets, which is something we will achieve for the classes of geometric objects we consider.

[3] Note that while stating the main result in Theorem 1, we used the convention that the approximation ratio is > 1. This is done mainly for aesthetical reasons, making the result easier to parse. From now on, we assume $\beta < 1$; it is easy to see that this is equivalent to a data structure achieving a $(1/\beta)$-approximation in the language of Theorem 1.

Overview

Our goal is thus to develop a general method for dynamic approximate MIS that has efficient worst-case running times and small update set sizes for various classes of geometric objects. We identify two ingredients that are needed:

β-dynamic independent set query structure (β-DISQS): This an abstract data type for maintaining a set S of n objects, in which one can insert or delete an object in time $f(n)$, and obtain a β-approximate MIS of the current set S in output-sensitive time $kf(n)$ if the set returned has size k.

MIX algorithm: A MIX algorithm receives two independent sets S_1 and S_2 whose sizes sums to n as input, and smoothly transitions from S_1 to S_2 by adding or removing one element at a time such that at all times the the intermediate sets are an independent sets of size at least $\min(|S_1|, |S_2|) - o(n)$. The running time of the MIX algorithm is said to be $\gamma(n)$ if the entire computation takes time $n\gamma(n)$.

The plan of the paper is as follows.

Section 3: Amortized Update Time

In this section, we prove that the first ingredient, the β-DISQS, is sufficient to obtain a $(\beta - \epsilon)$-approximate dynamic MIS algorithm with $O(f(n))$ amortized update time for any $\epsilon > 0$. This is presented as Theorem 4. We note that this is a bit of a "folklore" algorithm that essentially does nothing more than periodically querying the β-DISQS.

Section 4: Worst-case Update time

The second ingredient, MIX, is vital to deamortizing: In Section 4, we show that a DISQS and a MIX together are sufficient to produce a data structure which for any constant $\epsilon > 0$ maintains an approximate MIS of size at least $(\beta - \epsilon)\,\mathrm{OPT} - o(\mathrm{OPT})$, has a worst-case running time of $O(f(n) + \gamma(n) + \log n)$, and whose UPDATE operation returns a constant-sized update set; this is presented as Theorem 7. We also show in that the non-existence of a MIX function implies the impossibility of a approximate MIS data structure with sublinear update set size, hence sublinear worst-case running time, and that this impossibility holds for rectangles and other classes of nonfat objects (Theorem 8 and Lemma 9).

Given this generic scheme developed, to prove our main result, it remains to show that fat objects have a MIX function and a DISQS. This is the content of Sections 5 and 6.

Section 5: Existence of a MIX function for fat objects

We show in Lemma 13 that for fat objects in any constant dimension a MIX algorithm always exists with $\gamma(n) = O(\log n)$. This is achieved via geometric separators [34].

Section 6: Existence of β-DISQS for fat objects

Last, we show that obtaining a β-DISQS is possible for many types of fat objects using variants of standard range query data structures (kd-trees for orthogonal objects and partition trees for non-orthogonal objects such as disks, triangles or general polyhedra) with the running time $f(n)$ matching the query time of such structures. The result is based on a simple greedy algorithm for the MIS problem on fat objects [22, 32], yielding an approximation factor β that only depends on the fatness constant.

3 Dynamization

In this section we define formally the β-dynamic independent set query structure (β-DISQS) and show that its existence implies a dynamic independent set algorithm with approximation ratio $\beta - \epsilon$ and sublinear amortized update time.

▶ **Definition 3.** *A* β-dynamic independent set query structure (β-DISQS) *is a data structure that maintains a set* S *whose size is denoted as* n *and supports the following operations:*

- UPDATE(u)*: Insert or remove* u*, so that* S *becomes* $S \oplus u$*.*
- QUERY*: Returns a* β*-approximate MIS of* S

We say that the running time of a DISQS is $f(n)$ *if* UPDATE *takes time* $f(n)$ *and if* QUERY *returns a set of size* k *in time* $kf(n)$*. We require* $f(n)$ *to be sublinear.*

We now show that a β-DISQS is sufficient to give a approximate MIS data structure with an amortized running time, with a loss of only ϵ in the approximation factor for any $\epsilon > 0$. The intuition is simple, pass through the update operations to the DISQS and periodically replace the approximate MIS seen by the user by querying the DISQS. The only subtlety is to immediately remove any items from the approximate MIS that have been deleted in order to keep the approximate MIS valid. This simple transformation is likely folklore, but we work out the details in the full version of this paper for completeness.

▶ **Theorem 4.** *Given a* β*-DISQS with sublinear running time* $f(n)$ *for an independent set problem and a* $\epsilon > 0$ *there is a fully dynamic data structure to maintain a* $(1-\epsilon)\cdot\beta$*-approximate independent set that runs in* $O\left(\frac{1}{\epsilon}f(n) + \log n\right)$ *amortized time per operation.*

4 Deamortization

In this section we present our deamortization technique. In particular, we describe a procedure, which we call MIX, which if exists, is used to transform a β-DISQS into a deterministic dynamic algorithm for with worst-case update time guarantees and update set size bounds. We also show that if a MIX does not exist for an independent set problem, then no sublinear worst-case update time guarantees are possible.

MIX function

We now define our main ingredient for deamortization, which essentially says that we can smoothly switch from one solution to another, by adding or removing one item at a time:

▶ **Definition 5** (MIX function)**.** *Given two solution sets* A *and* B*, let* $\mathrm{MIX}(A, B, i)$*, for* $i \in [0, |A| + |B|]$ *be a set where:*

- $\mathrm{MIX}(A, B, i)$ *is always a valid solution*
- $\mathrm{MIX}(A, B, 0) = A$
- $\mathrm{MIX}(A, B, |A| + |B|) = B$
- $|\mathrm{MIX}(A, B, i)| \geq \min(|A|, |B|) - \Gamma(|A| + |B|)$*, for some* $\Gamma(|A| + |B|) = o(|A| + |B|)$*.*
- $\mathrm{MIX}(A, B, i)$ *and* $\mathrm{MIX}(A, B, i + 1)$ *differ by one item.*

Given this purely combinatorial definition, we define a MIX algorithm as follows.

▶ **Definition 6.** *A MIX algorithm with running time* $\gamma(n)$ *is a data structure such that*

1. *It is initialized with* A*,* B*,* $i = 0$ *and it has a single operation* ADVANCE *which advances* i *and reports the single element in the symmetric difference* $\mathrm{MIX}(A, B, i) \oplus \mathrm{MIX}(A, B, i-1)$*.*
2. *The initialization plus* $|A|+|B|$ *calls to* ADVANCE *run in total time* $(|A|+|B|)\cdot\gamma(|A|+|B|)$*.*

The rest of this section is organized as follows. In 4.1 we show that given a β-DISQS and a MIX function for an independent set problem, we can produce a dynamic algorithm with worst-case update time guarantees and approximation ratio arbitrarily close to β. In 4.2, we show the necessity of MIX function; in other words, we show that if there does not exist a MIX function for an independent set problem, then no deterministic algorithms with worst-case update time guarantees exist.

4.1 Dynamic algorithm with worst-Case update time

We now present our main theorem. As we discuss next, the intuition expands upon that for Theorem 4, that in addition to periodically using the β-DISQS to get a new solution, the MIX slowly transitions between the two previous solutions reported by the β-DISQS. Our result is the following.

▶ **Theorem 7.** *Given a β-DISQS with running time $f(n)$ for an independent set problem, a $\gamma(n)$-time MIX algorithm with nondecreasing $\Gamma(n) = o(n)$, and an $0 < \epsilon < 1/4$, there is a fully dynamic data structure to maintain an independent set of size at least $(\beta - \epsilon)\,\mathrm{OPT} - o(\mathrm{OPT})$. The data structure runs in $O_{\epsilon,\beta}(f(n) + \gamma(n) + \log n)$ worst-case time per operation, where n is the current number of objects stored, and reports a $O_\epsilon(1)$ number of changes in the independent set per update.*

High-level description

We saw in the previous section how to obtain an amortized update time algorithm by splitting the update sequence into *rounds* and query the DISQS at the end of each round to recompute an approximate MIS. Let $\hat{I}_k$ be the independent set produced by the DISQS at the end of round k. At a high-level, the main task is to deamortize the computation of $\hat{I}_k$: we can not afford computing it during one step. Instead, we compute $\hat{I}_k$ gradually during round $k+1$, making sure that the running time per step is bounded. $\hat{I}_k$ is eventually computed by the end of round $k+1$. At this point, we would like to have $\hat{I}_k$ (discarding elements deleted during round $k+1$) as our output independent set; however this can not be done immediately, since $\hat{I}_k$ might be very different from $\hat{I}_{k-1}$. For that reason, the switch from $\hat{I}_{k-1}$ to $\hat{I}_k$ is done gradually using the MIX function during round $k+2$. After all, our algorithm uses $\hat{I}_k$ as its independent set at the end of round $k+2$.

It follows that the independent set reported to the user is a combination of DISQS queries 3 or 2 rounds in the past. We show that by appropriately choosing the lengths of the rounds depending on the sizes of the independent sets, this lag affects the approximation factor by an additive ϵ.

Proof of Theorem 7. We group the updates u_i into rounds, and use r_k to indicate that u_{r_k} is the last update of round k. Let $\hat{I}_k$ be the independent set output by the β-DISQS (if queried) at time r_k, i.e., at the end of kth round. The length of the kth round, $R_k = r_k - r_{k-1}$ is defined to be $R_k := \max\{1, \lfloor \epsilon \cdot |\hat{I}_{k-2}| \rfloor\}$ updates.

For convenience, we define the following functions for any $0 < \epsilon < 1/4$:

$$g(\epsilon) = \frac{1 + \sqrt{1 - 4\epsilon}}{2} \qquad h(\epsilon) = \frac{3 - \sqrt{1 - 4\epsilon}}{2} \qquad \phi(\epsilon) = \frac{16h^2(\epsilon)}{\epsilon \cdot \beta \cdot g^3(\epsilon)}$$

Note that $g(\epsilon) = 1 - \epsilon - O(\epsilon^2)$ and $h(\epsilon) = 1 + \epsilon + O(\epsilon^2)$. Note also that $g(\epsilon) \in (1/2, 1)$ and in particular $g(\epsilon) \to 1$ as $\epsilon \to 0$. Similarly, $h(\epsilon) \in (1, 3/2)$ and $h(\epsilon) \to 1$ as $\epsilon \to 0$.

Our Data Structures and Operations

Our overall structure contains a β-DISQS and a MIX algorithm. We also maintain the current active set of objects S_i and our approximate independent set I_i explicitly in a binary search tree; we refer to simply as S and I, each stored according to some total order on the objects[4]. We maintain the invariant that at the beginning of round k, the DISQS stores the set of objects $S_{r_{k-2}}$. Also, our structure stores $\hat{I}_{k-3}$ and $\hat{I}_{k-2}$.

To execute update operations in round k, the following are performed:

Running MIX slowly: During round k we use MIX to transition from $\hat{I}_{k-3}$ to $\hat{I}_{k-2}$. This is done by initializing MIX with $\hat{I}_{k-3}$ and $\hat{I}_{k-2}$ and repeatedly running the ADVANCE operation. After each update, we continue running MIX where we left off and continue until either $\phi(\epsilon) \cdot \gamma\left(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|\right)$ time has passed or if $\phi(\epsilon)$ calls to ADVANCE have been performed.

Operation archiving: All updates are placed into a queue Q as they arrive. This will ensure that the DISQS will take into account all updates of previous rounds. Moreover, if an update u_i deletes an element v of S, we wish to report it as being deleted. To do so we set a variable $\Delta_i^{\text{DEL}} = \{v\}$, and otherwise $\Delta_i^{\text{DEL}} = \emptyset$.

Interaction with the DISQS: During round k, we want to use the DISQS in order to compute the set $\hat{I}_{k-1}$. To do that, we first perform to the DISQS all updates of round $k-1$ one by one and remove them from Q. This way, DISQS stores the set $S_{r_{k-1}}$. Then, we perform a query to the DISQS, to get $\hat{I}_{k-1}$. In each update of the round $\left(1 + \frac{2h(\epsilon)}{\beta\epsilon}\right) f(n)$ time is spent on executing these operations.

Maintaining S: We store S in a binary search tree based on some total ordering of the objects. For each update u_i, we search for u_i in S and remove it if it is there, and add it if it is not, to maintain $S = S_i = S_{i-1} \oplus \{u_i\}$.

Output: After each update, we report the symmetric difference Δ between previous and current independent set to the user. Let Δ_i^{MIX} be the union of the items returned by the ADVANCE operation of the MIX algorithm during the execution of update u_i. We combine Δ_i^{MIX} and Δ_i^{DEL}, and before returning, we remove any items that would result in the insertion into I_i of items that are not in S.

Roadmap

We need to argue about (i) correctness, (ii) running time, (iii) approximation ratio and (iv) *feasibility* of our algorithm, i.e., that during each round the computation of MIX and DISQS have finished before the round ends. Due to space limitations, we provide a brief sketch here and defer the formal proofs to the full version.

Correctness

It is easy to see that the algorithm described above always outputs an independent set to the user: During current round t, the user always sees $\text{MIX}(\hat{I}_{t-3}, \hat{I}_{t-2}, j)$ for some j, with perhaps some items that have been deleted in round $t-1$ or the current round removed. MIX is by definition an independent set at every step as in any subset of it, so the user always sees an independent set.

[4] Instead of a tree one could use a hash table, which would remove the additive logarithmic term from the update time, at the expense of randomization. However this will not improve our overall result, since functions $\gamma(n)$ and $f(n)$ are at least logarithmic in our application (see sections 5,6); therefore the logarithm can be absorbed while keeping the result deterministic.

Bound on update set size

The update set returned is a subset of $\Delta_i^{\text{MIX}} \cup \Delta_i^{\text{DEL}}$, as insertions of Δ_i^{MIX} might be removed if the items are no longer in S. The size of Δ_i^{MIX} is at most $\phi(\epsilon)$ by construction; the set Δ_i^{DEL} contains at most one element. Therefore,

$$\Delta \leq \phi(\epsilon) + 1 = \frac{16h^2(\epsilon)}{\epsilon \cdot \beta \cdot g^3(\epsilon)} + 1 \leq \frac{16}{\epsilon \cdot \beta} \cdot \frac{(3/2)^2}{(1/2)^3} + 1 = O\left(\frac{1}{\epsilon \cdot \beta}\right) = O_{\epsilon,\beta}(1) \tag{1}$$

Running time

The formal proof is located in the full version. Briefly, MIX takes time $\phi(\epsilon) \cdot \gamma\left(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|\right)$. It is easy to show that $|\hat{I}_{k-3}| + |\hat{I}_{k-2}| \leq \frac{2\,\text{OPT}}{g^3(\epsilon)} \leq 16n$. Using that γ is sublinear, we get that $\phi(\epsilon) \cdot \gamma\left(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|\right) = O_\epsilon(\gamma(n))$. DISQS works for time $(1 + \frac{2h(\epsilon)}{\beta \cdot \epsilon})f(n) = O_{\epsilon,\beta}(f(n))$. Last, operations for checking if update elements are still in S require time $O(|\Delta_i^{\text{MIX}} \cup \Delta_i^{\text{DEL}}| \cdot \log n) = O_{\epsilon,\beta}(\log n)$, due to (1).

Approximation ratio and feasibility

The proofs of approximation ratio and feasibility are simple but technical, and deferred to the full version. At a high level, the main idea is that the size of each round is carefully chosen such that, combined with Fact 2, the approximation factor β of the DISQS worsens just by an additive ϵ term due to the delayed updates and allows the DISQS and MIX operation to complete during the round. ◀

4.2 Necessity of the MIX function

▶ **Theorem 8.** *Suppose for any n, there are independent sets A and B of size n such there is no MIX function with $\Gamma(n) \leq (1-\beta)n$ for all $A' \subseteq A$ and $B' \subseteq B$, where $|A'|, |B'| \geq \beta n$. Then there is no $(\beta+\epsilon)$-approximate dynamic MIS algorithm that reports at most $o(n)$ changes in the independent set per update, for any $\epsilon > 0$.*

Proof. Let $\delta(n) = o(n)$ and suppose there is a $(\beta - \epsilon)$-approximate MIS algorithm that reports $\delta(n)$ changes per update in the worst case. Insert A in to the data structure. Then insert B and delete A. This is $|A| + |B|$ update operations. At all times the independent set is at least $(\beta - \epsilon)n$, and there are at most $\delta(n)$ changes per update operation. We could transform this into a MIX function by taking the at most $\delta(n)$ changes from update and report each change one at a time, first deletes and then inserts; thus at each step of the resultant MIX, their independent set is at least $(\beta + \epsilon)n - \delta(n)$ which is at least βn for sufficiently large n ◀

We can apply this to the case of rectangles in the plane, where we show that with a non-trivial worst-case performance of $o(n)$ changes in the independent set per operation, it is impossible to have an β-approximate MIS for any $\beta > 0$.

▶ **Lemma 9.** *There is no MIX function for rectangles with $\Gamma(n) < n$. Thus from Theorem 8, for any $\beta > 0$, there is no β-approximate dynamic MIS algorithm that reports at most $o(n)$ changes in the independent set per update.*

Proof. This is equivalent to saying that there are sets of rectangles A and B such that for any MIX function, there is an i such that $\text{MIX}(A, B, i) = \emptyset$. Consider sets of rectangles A and B, each of size n, in the form of a grid such that A are horizontally thin and disjoint, B

are vertically thin and disjoint, and every rectangle of A intersects all rectangles of B. In a MIX function, starting from A, one can not add a single element from B until all elements of A have been removed. ◀

This construction works for any class of objects for such a generalized "hashtag" construction is possible, which includes any class of shapes which are connected and where for any rectangle, there is a shape in the class that has that rectangle as its minimum orthogonal bounding rectangle. This includes natural classes of shapes without fatness constraints, such as triangles, ellipses, polygons, etc.

5 A MIX algorithm for fat objects

Fat objects

There are many possible definitions of fat objects in Euclidean space, we use the following one from [15]. Define the center and size of an object to be the center and side length of one of its minimal enclosing hypercube.

▶ **Definition 10.** *A collection S of (connected) objects is f-fat, for a constant f, if for any size-r box R, there are f points such that every object that intersects R and has size at least r contains one of the chosen points.*

This implies that any box can only intersect f disjoint objects of size larger than the box. Throughout the whole section, f and the dimension d are considered to be constant.

We will develop and use a variant of the rectangle separator theorem of Smith and Wormald [34]. We first state the classic version, and then prove the variant we need. Our proofs are straightforward adaptations of those in [34].

▶ **Lemma 11** (Smith and Wormald [34])**.** *For any set S of disjoint squares objects in the plane, there is a separating rectangle R that such if we partition S into S^{IN}, S^{OUT} and S^{ON} based on whether each object lies entirely inside R, entirely outside R, or intersects R, $S^{\text{IN}} \leq \frac{2}{3}|S|$, $S^{\text{OUT}} \leq \frac{2}{3}|S|$ and $S^{ON} = O(\sqrt{|S|})$.*

What we need differs from this in that we have two sets of fat objects, in each set the objects are disjoint but intersection is allowed between the two sets, and we want to have the separator intersect with an order-square-root number of objects in each set. However we require the separator to be balanced with respect to the first set only; it is not possible to require balance with respect to both sets. We state the separator theorem here, the proof is in the full version.

▶ **Lemma 12.** *Let S_1 and S_2 be two sets of disjoint f-fat objects in d-dimensions. Let $n = |S_1| + |S_2|$. We can compute a hypercube s and sets $S_1^{\text{IN}}, S_2^{\text{IN}}, S_1^{\text{OUT}}, S_2^{\text{OUT}}$ with the following properties in time $O(d \cdot n) = O(n)$:*

- *The hypercube s intersects $O(n^{1-\frac{1}{d}})$ objects of $S_1 \cup S_2$.*
- $S_1^{\text{IN}} \subseteq S_1$, $S_2^{\text{IN}} \subseteq S_2$, $S_1^{\text{OUT}} \subseteq S_1$, $S_2^{\text{OUT}} \subseteq S_2$
- *All objects in S_1^{IN} and S_2^{IN} lie entirely inside s*
- *All objects in S_1^{OUT} and S_2^{OUT} lie entirely outside s*
- $|S_1^{\text{IN}}| \leq \frac{4^d-1}{4^d}|S_1|$
- $|S_1^{\text{OUT}}| \leq \frac{4^d-1}{4^d}|S_1|$

▶ **Lemma 13.** *Fat objects in constant dimension d have a MIX algorithm with running time $\gamma(n) = O(\log n)$: Given independent sets of fat objects S_1 and S_2, there is a MIX from S_1 to S_2 whose size is always at least $\min(|S_1|, |S_2|) - \Gamma(|S_1| + |S_2|)$, with $\Gamma(n) = O(n^{1-\frac{1}{d}} \log n) = o(n)$. The total running time of initializing the algorithm with S_1 and S_2 and performing all steps of MIX is $O((|S_1| + |S_2|) \cdot \log(|S_1| + |S_2|))$.*

Proof. We compute the separator s from Lemma 12. Let S_1^{IN}, S_1^{OUT}, S_1^s S_2^{IN}, S_2^{OUT}, S_2^s, denote partition of S_1 and S_2 into parts that are completely inside the separator, completely outside, and those that intersect the separator, respectively.

Main Idea. The main idea is the following: First, we will remove all objects of S_1^s. Then the remaining objects of S_1 would be either completely inside s or completely outside S. We will use recursively the MIX function in both sides to switch from S_1^{IN} to S_2^{IN} and from S_1^{OUT} to S_2^{OUT} respectively. At the end we will add S_2^s.

Applying the recursion carefully. Recall that our goal is twofold: First, minimize the running time of MIX, second, make sure that at each step the size of the current set is at large as possible. Formally, minimize both $\gamma(n)$ and $\Gamma(n)$, which should be definitely sublinear, and as small as possible. Note that towards the second goal, i.e., keeping the set size as high as possible at all times, the sets used to apply the first recursive call of MIX matter: should we start from switching from S_1^{IN} to S_2^{IN} or from S_1^{OUT} to S_2^{OUT}? We want to make the choice that leads to the largest independent set at the intermediate step when only one of the two recursive calls has been applied.

We denote a and b the sides of separator (IN and OUT) so that $|S_1^a| + |S_2^b| \leq |S_1^b| + |S_2^a|$ holds. As $\min(w + x, y + z) \leq \max(w + z, x + y)$:

$$|S_1^b| + |S_2^a| = \max(|S_1^b| + |S_2^a|, |S_1^a| + |S_2^b|) \geq \min(|S_1^a| + |S_1^b|, |S_2^a| + |S_2^b|). \tag{2}$$

This way, at the intermediate step when we have mixed only one side, the set size will not be smaller than the beginning or the end of the mix operation.

Formal Description. The MIX function then proceeds as follows:

1. Start with S_1.
2. Remove the elements of S_1^s, one at a time, to give $S_1^a \cup S_1^b$.
3. Recursively MIX S_1^a to S_2^a.
4. Now we have $S_2^a \cup S_1^b$.
5. Recursively MIX S_1^b to S_2^b. At the end of this process we will have $S_2^b \cup S_2^b$.
6. Add the elements of S_2^s, one at a time.
7. We finish with $S_2^a \cup S_2^b \cup S_2^s = S_2$.

The base case is when one of the two sets is empty, and the MIX proceeds in the obvious way by deleting all elements of S_1 if S_2 is empty, or inserting all elements of S_2 if S_1 is empty. In such a case the lemma holds trivially as $\min(|S_1|, |S_2|)$ is zero.

We need to argue that at all times this process generates a set that is an independent set of the claimed size.

Always an independent set. In steps 1-2 we always have a subset of S_1, which is an independent set, the same holds in steps 6-7 with respect to S_2. In step 4, $S_2^a \cup S_1^b$ is independent as each of the sets are independent and all of the objects on each set are entirely on opposite sides of the separating rectangle s. Steps 3 and 5 hold by induction, and that the part we are recursively MIXing and the part that is unchanged are entirely on opposite sides of the separating rectangle.

Size bound. Let $\mathrm{MIX}_{\min}(S_1, S_2, n)$ be the smallest size of the independent set during the running of the MIX function from S_1 to S_2, and where n is an upper bound on $|S_1| + |S_2|$. Then we can directly express $\mathrm{MIX}_{\min}$ as a recurrence, taking the minimum of each step of the algorithm:

$$\mathrm{MIX}_{\min}(S_1, S_2, n) = \min(|S_1|, |S_1^a| + |S_1^b|, \mathrm{MIX}_{\min}(|S_1^a|, |S_2^a|, n) + |S_1^b|, |S_2^a| + |S_1^b|, \\ |S_2^a| + \mathrm{MIX}_{\min}(|S_1^b|, |S_2^b|, n), |S_2^a| + |S_2^b|, |S_2|)$$

We will prove that

$$\mathrm{MIX}(S_1, S_2, n) \geq \min(|S_1|, |S_2|) - (\log_{\frac{4^d}{4^d-1}} |S_1|) \cdot n^{1-\frac{1}{d}} \tag{3}$$

which implies the claim of this lemma. The details of this inductive proof are given in the full version. It follows exactly the intuition that (i) the only loss in the MIX function is the separators at each level of the recursion, (ii) there are logarithmic levels of the recursion and (iii) the size of the separators in each level are $O(n^{1-1/d})$. Some care must be taken, since the separators are only balanced for one of the two sets.

Running time. The running time is given by the recursion:

$$T(S_1, S_2) = \begin{cases} |S_1| & \text{if } S_2 = \emptyset \\ |S_2| & \text{if } S_1 = \emptyset \\ T(S_1^a, S_2^a) + T(S_1^b, S_2^b) + O(|S_1| + |S_2|) & \text{otherwise} \end{cases}$$

where the additive term in the last case is dominated by the time needed to compute the separator (Lemma 12) which is $O(d \cdot (|S_1| + |S_2|)) = O(|S_1| + |S_2|)$, since d is assumed to be constant.

Recall $|S_1^a|, |S_2^a| \leq \frac{4^d}{4^d-1} S_1$, S_1^a and S_1^b are disjoint subsets of S_1, and S_2^a and S_2^b are disjoint subsets of S_2. Hence the recursion depth is logarithmic in $|S_1|$ and each item from S_1 and S_2 is passed to at most one of the recursive terms.

Thus the running time is $O((|S_1| + |S_2|) \log |S_1|)$. As the running time is defined to be $(|S_1| + |S_2|) \cdot \gamma(|S_1| + |S_2|)$, we have $\gamma(|S_1| + |S_2|) \leq \log(|S_1| + |S_2|)$.

Remark. We note the effect of the running time of the separator algorithm (Lemma 12) on the running time of the MIX algorithm: If the running time was $O(n^c)$ for some constant c, then the additive term in the recursion would have increase to $O((|S_1| + |S_2|)^c)$, leading to $\gamma(n) = n^{c-1} \cdot \log n$; such a running time would be sublinear only for $c < 2$; here, by achieving $c = 1$, we get the fastest possible running time which implies the polylogarithmic running time for MIX for fat objects. ◀

6 DISQS for fat objects

In this section, we define DISQS for various classes of geometric objects. First observe that for intervals, a 1-DISQS with running time $O(\log n)$ follows from the classic greedy algorithm [25]. By storing the intervals in an augmented binary search tree, one can insert and delete intervals as well as answer queries of the form "What is the interval entirely to the right of x with the leftmost right endpoint?" As intervals are fat, this implies a $(1-\epsilon)$ approximate MIS algorithm with running time $O(\frac{1}{\epsilon} \log n)$. This in not new, in the past year a complicated structure via local exchanges appeared in [12] and soon after a much simpler method [21] using a local rebuilding was obtained; we obtain this as part of our more general scheme for fat objects.

We now focus on developing DISQS for fat objects. This involves two ideas. First, we use a simple greedy offline algorithm that computes an constant-approximate MIS for fat objects. Then we combine this algorithm with a range searching data structure to implement the greedy choice.

A greedy offline approximation algorithm

Given a collection of fat objects, we consider the independent set obtained by sorting them by size, from the smallest to the largest, and adding them greedily to the independent set, provided they do not intersect anything added previously. We refer to this algorithm as the *greedy* algorithm. It was considered in particular by Chan and Har-Peled [17], but was known in special cases before [22]. From the definition of f-fat objects, every successive object returned by the algorithm can intersect at most f disjoint objects that are larger. Hence this simple algorithm yields a constant-factor approximation algorithm for fat objects.

▶ **Lemma 14.** *For f-fat objects in dimension d, the greedy algorithm returns an $1/f$-approximate MIS.*

We need to implement this greedy method as the query of a DISQS; that is, it should support insertion and deletion of objects, and the running time of the greedy algorithm must be output-sensitive. This can be done with a slight variation of classic range intersection query structures, where we can insert and delete objects, mark or unmark objects that intersect a given query, and return the largest unmarked object. Thus each item returned by the greedy algorithm is reported after a constant number of range intersection query operations.

The DISQS data structures thus have running times that match those of the underlying range intersection query structures when the query ranges are from the same family of objects as the objects stored: $O(n^{1-\frac{1}{2d}})$ for hyperrectangles, using kd-trees, $\tilde{O}(n^{1-\frac{1}{d+2}})$ for disks and $\tilde{O}(n^{1-\frac{1}{d(d+1)}})$ for simplices, using partition trees; see the full version for more details.

Summary

The range intersection queries (details in the full version) combined with the greedy algorithm (Lemma 14) gives a DISQS whose running time depends on the range intersection queries, and whose approximation radio is the inverse of the fatness constant. From Lemma 13 fat objects have a MIX algorithm. Given the MIX, DISQS, and constant $\epsilon > 0$, Theorem 7 yields a dynamic MIS structure, with worst-case update time depending on the range intersection queries, approximation ratio within ϵ of that from the fatness, and with only a constant-size update set per operation. Putting all these pieces together yields Theorem 1.

References

1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 114–125, 2019. doi:10.1145/3313276.3316376.

2 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Foundations of Computer Science (FOCS)*, pages 400–409. IEEE, 2013. doi:10.1109/FOCS.2013.50.

3 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. In *36th International Symposium on Computational Geometry, SoCG*, pages 2:1–2:15, 2020. doi:10.4230/LIPIcs.SoCG.2020.2.

4 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, 1997.

5 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 815–826, 2018. doi:10.1145/3188745.3188922.

6 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1919–1936, 2019. doi:10.1137/1.9781611975482.116.

7 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 382–405, 2019. doi:10.1109/FOCS.2019.00032.

8 Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 427–436, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365496.

9 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 406–423, 2019. doi:10.1109/FOCS.2019.00033.

10 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2537–2549, 2021. doi:10.1137/1.9781611976465.150.

11 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1872–1885, 2019. doi:10.1137/1.9781611975482.113.

12 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. *CoRR*, abs/2007.08643, 2020. arXiv:2007.08643.

13 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 892–901. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496867.

14 Parinya Chalermsook and Bartosz Walczak. Coloring and maximum weight independent set of rectangles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 860–868, 2021. doi:10.1137/1.9781611976465.54.

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

16 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, pages 373–392, 2012.

17 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012.

18 Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. In *36th International Symposium on Computational Geometry, SoCG*, page In press, 2021.

19 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 370–381, 2019. doi:10.1109/FOCS.2019.00031.

20 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Foundations of Computer Science (FOCS)*, pages 820–829. IEEE, 2016. doi:10.1109/FOCS.2016.92.

21 Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. *CoRR*, abs/2012.15002, 2020. arXiv:2012.15002.

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

23 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, pages 1302–1323, 2005. doi:10.1137/S0097539702402676.

24 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, pages 133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

25 Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1(2):180–187, 1972. doi:10.1137/0201013.

26 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, pages 227–242, 2015. doi:10.1016/j.tcs.2014.09.046.

27 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 537–550, 2017. doi:10.1145/3055399.3055493.

28 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. doi:10.1007/BF02392825.

29 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *36th International Symposium on Computational Geometry, SoCG*, pages 51:1–51:14, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.

30 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, pages 130–136, 1985. doi:10.1145/2455.214106.

31 Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 384–393, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314768.

32 Madhav V. Marathe, H. Breu, Harry B. Hunt III, S. S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995. doi:10.1002/net.3230250205.

33 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. *CoRR*, abs/2101.00326, 2021. arXiv:2101.00326.

34 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.

35 Bram Verweij and Karen Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*, pages 426–437, 1999. doi:10.1007/3-540-48481-7_37.

36 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, pages 103–128, 2007. doi:10.4086/toc.2007.v003a006.

Revision Notice

This is a revised version of the eponymous paper appeared in the proceedings of ESA 2021 (LIPIcs, volume 204, https://www.dagstuhl.de/dagpub/978-3-95977-204-4, published in September, 2021), in which it was erroneously claimed that the update time for squares, hypercubes and unions of k fat hyperrectangles is polylogarithmic. This claim is withdrawn and replaced by the polynomial update time presented in this version. Reaching polylogarithmic worst-case update time bounds for these objects would require new ideas, distinct from those given in Section 6.

Dagstuhl Publishing – October 15, 2021.