Dynamic Conflict-Free Colorings in the Plane[*]

Mark de Berg[†1] and Aleksandar Markovic[‡2]

1 TU Eindhoven, Eindhoven, The Netherlands
mdberg@win.tue.nl

2 TU Eindhoven, Eindhoven, The Netherlands
a.markovic@tue.nl

Abstract

We study dynamic conflict-free colorings in the plane, where the goal is to maintain a conflict-free coloring (CF-coloring for short) under insertions and deletions.

- First we consider CF-colorings of a set S of unit squares with respect to points. Our method maintains a CF-coloring that uses $O(\log n)$ colors at any time, where n is the current number of squares in S, at the cost of only $O(\log n)$ recolorings per insertion or deletion We generalize the method to rectangles whose sides have lengths in the range $[1, c]$, where c is a fixed constant. Here the number of used colors becomes $O(\log^2 n)$. The method also extends to arbitrary rectangles whose coordinates come from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors. The number of recolorings for both methods stays in $O(\log n)$.
- We then present a general framework to maintain a CF-coloring under insertions for sets of objects that admit a unimax coloring with a small number of colors in the static case. As an application we show how to maintain a CF-coloring with $O(\log^3 n)$ colors for disks (or other objects with linear union complexity) with respect to points at the cost of $O(\log n)$ recolorings per insertion. We extend the framework to the fully-dynamic case when the static unimax coloring admits weak deletions. As an application we show how to maintain a CF-coloring with $O(\sqrt{n}\log^2 n)$ colors for points with respect to rectangles, at the cost of $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

These are the first results on fully-dynamic CF-colorings in the plane, and the first results for semi-dynamic CF-colorings for non-congruent objects.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Conflict-free colorings, Dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.27

1 Introduction

Consider a set of base stations in the plane that can be used for mobile communication. To ensure a good coverage, the base stations are typically positioned in such a way that the communication ranges of different base stations overlap. However, if a user is within range of several base stations using the same frequency, then interference occurs and the communication is lost. Therefore, we want to assign frequencies to the base stations such that any user within range of at least one base station, is also within range of at least one

[*] A full version of the paper is available at [12], https://arxiv.org/abs/1709.10466.
[†] MdB is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.
[‡] AM is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

base station using an interference-free frequency. The easy solution would be to give all stations a different frequency. However, this is undesirable as the set of available frequencies is limited. The question then arises: how many different frequencies are needed to ensure that any user that is within range of at least one base station has an interference-free base station at his disposal? Motivated by this and other applications, Even *et al.* [14] and Smorodinsky [17] introduced the notion of *conflict-free colorings* or *CF-colorings* for short. Here the ranges of the base stations are modeled as regions (disks, or other objects) in the plane, and frequencies are represented by colors. A CF-coloring is now defined as follows.

Let S be a set of objects in the plane. For a point $q \in \mathbb{R}^2$, let $S_q := \{S \in S | q \in S\}$ be the subset of objects containing q. A coloring $col : S \to \mathbb{N}$ of the objects in S – here we identify colors with non-negative integers – is said to be *conflict-free (with respect to points)* if for each point q with $S_q \neq \emptyset$ there is an object $S \in S_q$ whose color is unique among the objects in S_q. A CF-coloring is called *unimax* when the maximum color in S_q is unique.

We can also consider a dual version of planar CF-colorings. Here we are given a set S of points and a family $\mathcal{F}$ of geometric ranges, and the goal is to color the points in S such that any range from $\mathcal{F}$ containing a least one point, contains a point with a unique color.

Conflict-free colorings have received a lot of attention since they were introduced by Even *et al.* [14] and Smorodinsky [17]; see the overview paper by Smorodinsky [18], which surveys the work up to 2010. We review the work most relevant to our results.

Even *et al.* proved that it is always possible to CF-color a set of disks in the plane using $O(\log n)$ colors, and that $\Omega(\log n)$ colors are needed in the worst case. The authors extended the result to sets of translates of any given centrally symmetric polygon. Later, Har-Peled and Smorodinsky [15] further generalized the result to regions with near-linear union complexity. The dual version of the problem was also studied by Even *et al.* [14]; they showed it is possible to CF-color points using $O(\log n)$ colors with respect to disks, or with respect to scaled translations of a centrally symmetric convex polygon. Moreover, Ajwani *et al.* [1] showed how to CF-color points with respect to using $O(n^{0.382})$ colors.

Recall that CF-colorings correspond to interference-free frequency assignments in a cellular network. When a node in the network fails, the resulting assignment may no longer be interference-free. This leads to the study of *k-fault-tolerant* CF-colorings, where we want $\min(k, |S_q|)$ objects from S_q to have a unique color. In other words, a k-fault-tolerant CF-coloring allows the deletion of k objects without losing the conflict-free property. Cheilaris *et al.* [5] studied the 1-dimensional case, and presented a polynomial-time algorithm with approximation ratio $5 - \frac{2}{k}$ for the problem of finding a CF-coloring with a minimum number of colors. For $k = 1$ – that is, the regular CF-coloring – the algorithm gives a 2-approximation. Horev *et al.* [16] studied the 2-dimensional case and proved a $O(k \log n)$ bound for disks and, more generally, regions with near-linear union complexity.

To increase coverage or capacity in a cellular network it may be necessary to increase the number of base stations. This led Fiat *et al.* [7] to study *online* CF-colorings. Here the objects to be CF-colored arrive over time, and as soon as an object appears it must receive a color which cannot be changed later on. For CF-coloring points with respect to intervals, they proposed a deterministic algorithm using $O(\log^2 n)$ colors as well as two randomized algorithms, one of which is using at most $O(\log n \log \log n)$ colors in expectation and always producing a valid coloring. Later, Chen *et al.* [6] improved the bound with an algorithm using an expected $O(\log n)$ colors.

Chen *et al.* [8] considered the problem of online CF-coloring of points with respect to geometric ranges. They showed that for ranges that are half-planes, unit disks, or bounded-size rectangles – i.e. rectangles whose heights and widths all lie in the range $[1, c]$, for some fixed constant c – there is an online CF-coloring using $O(\log n)$ colors with high probability.

For bounded-size rectangles they also presented a deterministic result using $O(\log^3 n)$ colors. Bar-Noy *et al.* [3] provided a general strategy for online CF-coloring of hypergraphs. Their method uses $O(k \log n)$ colors with high probability, where k is the so-called *degeneracy* of the hypergraph. Their method can for instance be applied for points with respect to half-planes using $O(\log n)$ colors, which implies [8] the same result for unit disks with respect to points. They also introduced a deterministic algorithm for points with respect to intervals in $\mathbb{R}^1$ if *recolorings* are allowed. Their method uses at most $n - \log n$ recolorings in total; they did not obtain a bound on the number of recolorings for an individual insertion. Note that the results for online colorings in $\mathbb{R}^2$ are rather limited: for the primal version of the problem – online CF-coloring objects with respect to points – there are essentially only results for unit disks or unit squares (where the problem is equivalent to the dual version of coloring points with respect to unit disks and unit squares, respectively). Moreover, most of the results are randomized.

De Berg *et al.* [11] introduced the *fully dynamic* variant of the CF-coloring problem, which generalizes and extends the fault-tolerant and online variants. Here the goal is to maintain a CF-coloring under insertions and deletions. It is easy to see that if we allow deletions and we do not recolor objects, we may need to give each object in S its own color. Using n colors is clearly undesirable. On the other hand, recoloring all objects after each update – using then the same number of colors as in the static case – is not desirable either. Thus the main question is which trade-offs can we get between the number of colors and the number of recolorings? De Berg *et al.* proved a lower bound on this trade-off for the 1-dimensional problem of CF-coloring intervals with respect to points. (For this case it is straightforward to give a static CF-coloring with only three colors.) Their lower bound implies that if we want $O(1/\varepsilon)$ colors, we must sometimes re-color $\Omega(\varepsilon n^{\varepsilon})$ intervals, and that if we allow only $O(1)$ recolorings we must use $\Omega(\log n/ \log\log n)$ colors in the worst case. They also presented a strategy that uses $O(\log n)$ colors at the cost of $O(\log n)$ recolorings. The main goal of our paper is to study fully dynamic CF-colorings for the 2-dimensional version of the problem.

Our contributions. In Section 2 we give an algorithm for CF-coloring unit squares using $O(\log n)$ colors and $O(\log n)$ recolorings per update. Note that $\Omega(\log n)$ is a lower bound on the number of colors for a CF-coloring of unit squares even in the static case, so the number of colors our fully dynamic method uses is asymptotically optimal. We also present an adaptation for bounded-size rectangles which uses $O(\log^2 n)$ colors. The method also extends to arbitrary rectangles whose coordinates come from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors. Both methods still use $O(\log n)$ recolorings per update. These constitute the first results on fully-dynamic CF-colorings in $\mathbb{R}^2$.

In Section 3, we give two general approaches that can be applied in many cases. The first uses a static coloring to solve insertions-only instances.

It can be applied in settings where the static version of the problem admits a unimax coloring with a small number of colors. The method can for example be used to maintain a CF-coloring for pseudodisks with $O(\log^3 n)$ colors and $O(\log n)$ recolorings per update, or to maintain a CF-coloring for fat regions. This is the first result for the semi-dynamic CF-coloring problem for such objects: previous online results for coloring objects with respect to points in $\mathbb{R}^2$ only applied to unit disks or unit squares. We extend the method to obtain a fully-dynamic solution, when the static solution allows what we call weak deletions. We can apply this technique for instance to CF-coloring points with respect to rectangles, using $O(\sqrt{n}\log^2 n)$ colors and $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

2 Dynamic CF-Colorings for Unit Squares and Rectangles

In this section we explain how to color unit squares with $O(\log n)$ colors and $O(\log n)$ recolorings per update. We then generalise this coloring to bounded-size rectangles, and to rectangles with coordinates from a fixed universe. We first explain our basic technique on so-called anchored rectangles.

2.1 A Subroutine: Maintaining a CF-coloring for Anchored Rectangles

We say that a rectangle r is *anchored* if its bottom-left vertex lies at the origin. Let S be a set of n anchored rectangles. We denote the x- and y-coordinate of the top-right vertex of a rectangle r by r_x and r_y, respectively. Our CF-coloring of S is based on an augmented red-black tree, as explained next.

To simplify the description we assume that the x-coordinates of the top-right vertices (and, similarly, their y-coordinates) are all distinct – extending the results to degenerate cases is straightforward. We store S in a red-black tree $\mathcal{T}$ where r_x (the x-coordinate of the top-right vertex of r) serves as the key of the rectangles $r \in S$. It is convenient to work with a *leaf-oriented* red-black tree, where the keys are stored in the leaves of the tree and the internal nodes store splitting values.[1] We can assume without loss of generality that the splitting values lie strictly in between the keys.

For a node $v \in \mathcal{T}$, let $\mathcal{T}_v$ denote the subtree rooted at v and let $S(v)$ denote the set of rectangles stored in the leaves of $\mathcal{T}_v$. We augment $\mathcal{T}$ by storing a rectangle $r_{\max}(v)$ at every (leaf or internal) node of v, define as follows:

$$r_{\max}(v) := \text{the rectangle } r \in S(v) \text{ that maximizes } r_y.$$

Let *left*(v) and *right*(v) denote the left and right child, respectively, of an internal node v. Notice that $r_{\max}(v)$ is the rectangle whose top-right vertex has maximum y-value among $r_{\max}(\mathit{left}(v))$ and $r_{\max}(\mathit{right}(v))$, so $r_{\max}(v)$ can be found in $O(1)$ time from the information at v's children. Hence, we can maintain the extra information in $O(\log n)$ time per insertion and deletion [9].

Next we define our coloring function. To this end we define for each rectangle $r \in S$ a set $N(r)$ of nodes in $\mathcal{T}$, as follows.

$$N(r) := \{v \in \mathcal{T} : \ v \text{ is the leaf storing } r, \text{ or } v \text{ is an internal node with } r_{\max}(\mathit{right}(v)) = r\}.$$

Observe that $N(r)$ only contains nodes on the search path to the leaf storing r and that $N(r) \cap N(r') = \emptyset$ for any two rectangles $r, r' \in S$. Let $\text{height}(v)$ denote the height of $\mathcal{T}_v$. Thus $\text{height}(v) = 0$ when v is a leaf, and for non-leaf nodes v we have $\text{height}(v) = \max(\text{height}(\mathit{left}(v)), \text{height}(\mathit{right}(v)) + 1$. We now define the color of a rectangle $r \in S$ as $\mathit{col}(r) := \max_{v \in N(r)} \text{height}(v)$. Since $N(r)$ always contains at least one node, namely the leaf storing r, this is a well-defined coloring.

▶ **Lemma 1.** *The coloring defined above is conflict-free.*

▶ **Theorem 2.** *Let S be a set of anchored rectangles in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of rectangles in S.*

[1] Such a leaf-oriented red-black tree can be seen as a regular red-black tree on a set $X'(S)$ that contains a splitting value between any two consecutive keys. Hence, all the normal operations can be done in the standard way.

Proof. Consider the coloring method described above, which by Lemma 1 is conflict-free. Since red-black trees have height $O(\log n)$, the number of colors used is $O(\log n)$ as well.

Now consider an update on S. The augmented red-black tree can be updated in $O(\log n)$ time in a standard manner [9]. The color of a rectangle $r \in S$ can only change when (i) the set $N(r)$ changes, or (ii) the height of a node in $N(r)$ changes. We argue that this only happens for $O(\log n)$ rectangles. Consider an insertion; the argument for deletions is similar. In the first phase of the insertion algorithm for red-black trees [9] a new leaf is created for the rectangle to be inserted. This may change height(v) or $r_{\max}(v)$ only for nodes v on the path to this leaf, so it affects the color of $O(\log n)$ rectangles. In the second phase the balance is restored using $O(1)$ rotations. Each rotation changes height(v) or $r_{\max}(v)$ for only $O(\log n)$ nodes, so also here only $O(\log n)$ rectangles are affected. ◀

2.2 Maintaining a CF-Coloring for Unit Squares

Let S be a set of unit squares. We first assume that all squares in S contain the origin.

A naive way to use the result from the previous section is to partition each square $s \in S$ into four rectangular parts by cutting it along the x-axis and the y-axis. Note that the set of north-east rectangle parts (i.e., the parts to the north-east of the origin) are all anchored rectangles, so we can use the method described above to maintain a CF-coloring on them. The other part types (south-east, south-west, and north-west) can be treated similarly. Thus every square $s \in S$ receives four colors. If we now assign a final color to s that is the four-tuple consisting of those four colors, then we obtain a CF-coloring with $O(\log^4 n)$ colors. (This trick of using a "product color" was also used by, among others, Ajwani *et al.* [1].)

It is possible to improve this by using the following fact: the ordering of the x-coordinates of the top-right corners of the squares in S is the same as the ordering of their bottom-right (or bottom-left, or top-left) corners. This implies that instead of working with four different trees we can use the same tree structure for all part types. Moreover, even the extra information stored in the internal nodes is the same for the north-east and north-west parts, since the y-coordinates of the top-right and top-left vertices are the same. Similarly, the extra information for the south-east and south-west parts are the same. Therefore, we can modify the augmented red-black tree to store two squares per internal node instead of one:

- $s_{\max}(v) :=$ the square $s \in S(v)$ that maximizes s_y,
- $s_{\min}(v) :=$ the square $s \in S(v)$ that minimizes s_y.

Next we modify our coloring function. Therefore we first redefine the set $N(s)$ of nodes for each square $s \in S$:

$$N(s) := \{\text{the leaf storing } s\} \cup N_{\text{NE}}(s) \cup N_{\text{SE}}(s) \cup N_{\text{SW}}(s) \cup N_{\text{NW}}(s),$$

where

- $N_{\text{NE}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\max}(\mathit{right}(v)) = s\}$,
- $N_{\text{SE}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\min}(\mathit{right}(v)) = s\}$,
- $N_{\text{SW}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\min}(\mathit{left}(v)) = s\}$,
- $N_{\text{NW}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\max}(\mathit{left}(v)) = s\}$.

The coloring is as follows. We now allow four colors per height-value, namely for height-value h we give colors $4h + j$ for $j \in \{0, 1, 2, 3\}$. These colors essentially correspond to the colors we would give out for the four part types. The color of a square s is now defined as

$$\mathit{col}(s) := \begin{cases} 0 & \text{if } \max\limits_{v \in N(s)} \text{height}(v) = 0 \text{ (s is only stored at a leaf)}, \\ 4 \cdot \max\limits_{v \in N(s)} \text{height}(v) + j & \text{if } \max\limits_{v \in N(s)} \text{height}(v) > 0, \end{cases}$$

where

$$j := \begin{cases} 0 & \text{if height}(s) = \max_{v \in N_{\mathrm{NE}}(s)} \text{height}(v), \\ 1 & \text{if the condition for } j = 0 \text{ does not apply and height}(s) = \max_{v \in N_{\mathrm{SE}}(s)} \text{height}(v), \\ 2 & \text{if the conditions for } j = 0, 1 \text{ do not apply and height}(s) = \max_{v \in N_{\mathrm{SW}}(s)} \text{height}(v), \\ 3 & \text{otherwise (we now must have height}(s) = \max_{v \in N_{\mathrm{NW}}(s)} \text{height}(v)). \end{cases}$$

The following lemma can be proven in exactly the same was ay Lemma 1. The only addition is that, when considering a set $S(v)$, we need to make a distinction depending on in which quadrant the query point q lies. If it lies in the north-east quadrant we can follow the proof verbatim, and the other cases are symmetric.

▶ **Lemma 3.** *The coloring defined above is conflict-free.*

It remains to remove the restriction that all squares contain the origin. To this end we use a grid-based method, similar to the one used by, e.g., Chen *et al.* [8]. Consider the integer grid, and assign each square in S to the grid point it contains; if a square contains multiple grid points, we assign it to the lexicographically smallest one. Thus we create for each grid point (i, j) a set $S(i, j)$ of squares that all contain the point (i, j). We maintain a CF-coloring for each such set using the method described above. Note that a square in $S(i, j)$ can only intersect squares in $S(i', j')$ when (i', j') is one of the eight neighboring grid points of (i, j). Hence, when $i' = i \mod 2$ and $j' = j \mod 2$ we can re-use the same color set, and so we only need four color sets of $O(\log n)$ colors each.

▶ **Theorem 4.** *Let S be a set of unit squares in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of squares in S.*

2.3 Maintaining a CF-Coloring for Bounded-Size Rectangles

Let S be a set of *bounded-size rectangles*: rectangles whose widths and heights are between 1 and c for some fixed constant c. Note that in practice, two different base stations have roughly the same coverage, hence it makes sense to assume the ratio is bounded by some constant c.

First consider the case where all rectangles in S contain the origin. Here, the x-ordering of the top-right corners of the rectangles may be different from the x-ordering of the top-left corners as we no longer use unit squares. Therefore the trees for the east (that is, north-east and south-east) parts no longer have the same structure. Note that the x-ordering of the top-left and bottom-left corners are the same, hence only one tree suffices for the east parts, and the same holds for the west parts. Hence, we build and maintain two separate trees, one for the east parts of the rectangles and one for the west parts. In the east tree we only work with the sets $N_{\mathrm{NE}}(s)$ and $N_{\mathrm{SE}}(s)$, and in the west tree we only work with $N_{\mathrm{SW}}(s)$ and $N_{\mathrm{NW}}(s)$; for the rest the structures and colorings are defined in the same as before. We then use the product coloring to obtain our bound: we give each rectangle a pair of colors – one coming from the east tree, one coming from the west tree – resulting in $O(\log^2 n)$ different color pairs.

To remove the restriction that each rectangle contains the origin we use the same grid-based approach as for unit squares. The only difference is that a rectangle in a set $S(i, j)$ can now intersect rectangles from up to $(1 + 2c)^2 - 1$ sets $S(i', j')$, namely with $i - c \leqslant i' \leqslant i + c$ and $j - c \leqslant j' \leqslant j + c$. Since c is a fixed constant, we still need only $O(1)$ color sets.

▶ **Theorem 5.** *Let S be a set of bounded-size rectangles in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log^2 n)$ colors using $O(\log n)$ recolorings per insertion and deletion, where n is the current number of rectangles in S.*

2.4 Maintaining a CF-Coloring for Rectangles with Coordinates from a Fixed Universe

The solution can also be extended to rectangles of arbitrary sizes, if their coordinates come from a fixed universe $U := \{0, \ldots, N-1\}$ of size N. Again, from a practical point of view it makes sense as in a city for instance, the places a base station can be created are limited.

To this end we construct a balanced tree $\mathcal{T}_x$ over the universe U, and we associate each rectangle $r = [r_{x,1}, r_{x,2}] \times [r_{y,1}, r_{y,2}]$ to the highest node v in $\mathcal{T}_x$ whose x-value $x(v)$ is contained in $[r_{x,1}, r_{x,2}]$. Let $S(v)$ be the set of objects associated to v. For each node $v \in \mathcal{T}_x$ we construct a balanced tree $\mathcal{T}_y(v)$ over the universe, and we associate each rectangle $r \in S(v)$ to the highest node w in $\mathcal{T}_y(v)$ whose y-value $y(w)$ is contained in $[r_{y,1}, r_{y,2}]$. (In other words, we are constructing a 2-level interval tree [10] on the rectangles, using the universe to provide the skeleton of the tree. The reason for using a skeleton tree is that otherwise we have to maintain balance under insertions and deletions, which is hard to do while ensuring worst-case bounds on the number of recolorings.) Let $S(w)$ be the set of objects associated to a node w in any second-level tree $\mathcal{T}_y(v)$. All rectangles in $S(w)$ have a point in common, namely the point $(x(v), y(w))$. Therefore we can proceed as in the previous section, and maintain a CF-coloring on $S(w)$ with a color set of size $O(\log^2 n)$, using $O(\log n)$ recolorings per insertions and deletion.

Note that for any two nodes w, w' at the same level in a tree $\mathcal{T}_y(v)$, any two rectangles $r \in S(w)$ and $r' \in S(w')$ are disjoint. Hence, over all nodes $w \in \mathcal{T}_x(v)$ we only need $O(\log N)$ different color sets. Similarly, for any two nodes v, v' of $\mathcal{T}_x$ at the same level, any two rectangles $r \in S(v)$ and $r' \in S(v')$ are disjoint. Hence, the total number of color sets we need is $O(\log^2 N)$. This leads to the following result.

▶ **Theorem 6.** *Let S be a set of rectangles in the plane, whose coordinates come from a fixed universe of size N. Then it is possible to maintain a CF-coloring on S with $O(\log^2 N \log^2 n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of rectangles in S.*

▶ Remark. Instead of assuming a skeleton tree and working with a fixed skeleton for our 2-level interval tree, we can also use randomized search trees. Then, assuming the adversary doing the insertions and deletions is oblivious of our structure and coloring, the tree is expected to be balanced at any point in time. Hence, we obtain $O(\log^4 n)$ colors in expectation, at the cost of $O(\log n)$ recolorings (worst-case) per update.

3 A General Technique

In this section we present a general technique to obtain a dynamic CF-coloring scheme in cases where there exists a static unimax coloring. (Recall that a unimax coloring is a CF-coloring where for any point q the object from S_q with the maximum color is unique.) Our technique results in a dynamic CF-coloring that uses $O(\gamma_{\mathrm{um}}(n) \log^2 n)$ colors, where $\gamma_{\mathrm{um}}(n)$ is the number of colors used in the static unimax coloring, at the cost of $O(\log n)$ recolorings per update. We first describe our technique for the case of insertions only. Then we extend the technique to the fully-dynamic setting, for the case where the unimax coloring allows for so-called weak deletions.

We remark that even though we describe our technique in the geometric setting in the plane, the techniques provided in this section can be applied in the abstract hypergraph setting as well.

3.1 An Insertion-Only Solution

Let S be a set of objects in the plane and assume that S can be colored in a unimax fashion using $\gamma_{\text{um}}(n)$ colors, where γ_{um} is a non-decreasing function. Here it does not matter if S is a set of geometric objects that we want to CF-color with respect to points, or a set of points that we want to CF-color with respect to a family of geometric ranges. For concreteness we refer to the elements from S as objects.

Our technique to maintain a CF-coloring under insertions of objects into S is based on the *logarithmic method* [4], which is also used to make static data structures semi-dynamic. Thus at any point in time we have $\lceil \log n \rceil + 1$ sets S_i such that each set S_i, for $i = 0, \ldots, \lceil \log n \rceil$, is either empty or contains exactly 2^i objects. The idea is to give each set S_i its own color set, consisting of $\gamma_{\text{um}}(2^i)$ colors. Maintaining a CF-coloring under insertions such that the *amortized* number of recolorings is small, is easy (and it does not require the coloring to be unimax): when inserting a new object we find the first empty set S_i, and we put all objects in $S_0 \cup \cdots \cup S_{i-1}$ together with the new object into S_i. The challenge is to achieve a *worst-case* bound on the number of recolorings per insertion. Note that for the maintenance of data structures, it is known how to achieve worst-case bounds using the logarithmic method. The idea is to build the new data structure for S_i "in the background" and switch to the new structure when it is ready. For us this does not work, however, since we would still need many recolorings when we switch. Hence, we need a more careful approach.

When moving all objects from $S_0 \cup \cdots \cup S_{i-1}$ (together with the new object) into S_i, we do not recolor them all at once but we do so over the next 2^i insertions. As long as we still need to recolor objects from S_i, we say that S_i is *in migration.* We need to take care that the coloring of a set that is in migration, where some objects still have the color from the set S_j they came from and others have already received their new color in S_i, is valid. For this we need to recolor the objects in a specific order, which requires the static coloring to be unimax as explained below. Another complication is that, because the objects in S_j that are being moved to S_i still have their own color, we have to be careful when we create a new set S_j. To avoid any problems, we need several color sets per set. Next we describe our scheme in detail.

As already mentioned, we have sets $S_0, \ldots, S_\ell$, where $\ell := \lceil \log n \rceil$. Each set can be in one of three states: *empty*, *full*, or *in migration.* For each i with $0 \leqslant i \leqslant \ell$ we have $\ell - i + 1$ color sets of size $\gamma_{\text{um}}(2^i)$ available, denoted by $\mathcal{C}(i,t)$ for $0 \leqslant t \leqslant \ell - i$. The insertion of an object s into S now proceeds as follows.

1. Let i be the smallest index such that S_i is empty. Note that i might be $\ell + 1$, in which case we introduce a new set and redefine ℓ. Note that this only happens when the number of objects reaches a power of 2.
2. Set $S_i := \{s\} \cup S_0 \cup \cdots \cup S_{i-1}$. Mark $S_0, \ldots, S_{i-1}$ as *empty*, and mark S_i as *in migration.*
3. Take an unused color set $\mathcal{C}(i,t)$ – we argue below that at least one color set $\mathcal{C}(i,t)$ with $0 \leqslant t \leqslant \ell - i$ is currently unused – and compute a unimax coloring of S_i using colors from $\mathcal{C}(i,t)$. We refer to the color from $\mathcal{C}(i,t)$ that an object in S_i receives as its *final color* (for the current migration). Except for the newly inserted object s, we do not recolor any objects to their final color in this step; they all keep their current colors.

4. For each set S_k in migration – this includes the set we just created in Step 3 – we recolor one object whose final color is different from its current color and whose final color is maximal among such objects. When multiple objects share that property, we arbitrarily choose one of them. If all objects in S_k now have their final color, we mark S_k as *full*.

▶ **Lemma 7.** *Suppose that when we insert an object s into S, the first empty set is S_i. Then the sets $S_0, \dots, S_{i-1}$ are full.*

Proof. Suppose for a contradiction that S_j, for some $0 \leqslant j < i$ is in migration. Consider the last time at which S_j was created – that is, the last time at which we inserted an object s' that caused the then-empty set S_j to be created and marked as in migration. Upon insertion of s', we already perform one recoloring in S_j. At that point all sets $S_0, \dots, S_{j-1}$ were marked empty and it takes $\sum_{t=0}^{j-1} 2^t = 2^j - 1$ additional insertions to fill them, giving us as many recolorings in S_j. Thus before we create any set S_i with $i > j$, we have already marked S_j as full. Since s' was the last object whose insertion created S_j, by the time we create S_i the set S_j must still be full – it cannot in the mean time have become empty and later be re-created (and thus be in migration). ◀

Next we show that in Step 3 we always have an unused color set at our disposal.

▶ **Lemma 8.** *When we create a new set S_i in Step 3, at least one of the color sets $\mathcal{C}(i,t)$ with $0 \leqslant t \leqslant \ell - i$ is currently unused.*

Proof. Consider a color set $\mathcal{C}(i,t)$. The reason we may not be able to use $\mathcal{C}(i,t)$ when we create S_i is that there is a set $S_{i'}$ with $i' > i$ that is currently in migration: the objects from a previous instance of S_i (that were put into $S_{i'}$ when we created $S_{i'}$) may not all have been recolored yet. By Lemma 7 this previous instance was full when it was put into $S_{i'}$ and so it only blocks a single color set, namely one for S_i. Thus the number of color sets $\mathcal{C}(i,t)$ currently in use is at most $\ell - i$. Since we have $\ell - i + 1$ such colors sets at our disposal, one must be unused. ◀

▶ **Theorem 9.** *Let $\mathcal{F}$ be a family of objects such that any subset of n objects from $\mathcal{F}$ admits a unimax coloring with $\gamma_{\text{um}}(n)$ colors, where $\gamma_{\text{um}}(n)$ is non-decreasing. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions, such that the number of used colors is $O(\gamma_{\text{um}}(n) \log^2 n)$ and the number of recolorings per insertion is at most $\lceil \log n \rceil$, where n is the current number of objects in S.*

Proof. The number of colors used is $\sum_{i=0}^{\ell} (\ell - i + 1)\gamma_{\text{um}}(2^i)$, where $\ell = \lceil \log n \rceil$. Since $\gamma_{\text{um}}(n)$ is non-decreasing, this is bounded by $O(\gamma_{\text{um}}(n) \log^2 n)$. The number of recolorings per insertion is at most one per set S_i, so at most $\lceil \log n \rceil$ in total. (The total number of sets is actually $\lceil \log n \rceil + 1$, but not all of them can be in migration.)

It remains to prove that the coloring is conflict-free. Consider a point $q \in \mathbb{R}^2$. (Here we use terminology from CF-coloring of objects with respect to points. In the dual version, q would be a range.) Let S_i be a set containing an object s with $q \in s$; if no such set exists there is nothing to prove.

If S_i is full then it has a unimax coloring using a color set $\mathcal{C}(i,t)$ not used by any other set S_j. Hence, there is an object containing q with a unique color.

Now suppose that S_i is in migration. We have two cases: (i) q is contained in an object from S_i that has already received its final color, (ii) all objects in S_i containing q still have their old color.

In case (i) the object containing q with the highest final color must have a unique color, because of the following easy-to-prove fact.

▶ **Fact.** *Consider any set A colored with a unimax coloring. Let z be an integer, and let $B \subseteq A$ be a subset that contains all objects of color greater than z, some objects of color z, and at most one other object. Then the coloring of B is unimax.*

This fact proves the statement above for case (i), because we recolor the objects in decreasing order of their colors and the coloring we are migrating to is a unimax coloring. (The "at most one other object" mentioned in the fact is needed because the object that caused the migration immediately receives its color, and this color needs not be the highest color.)

In case (ii), q is contained in an object from some old set S_j with $j < i$. At the time we created S_i this set S_j was CF-colored, and since we did not yet recolor any object from S_j that contains q – otherwise we are in case (i) – we conclude that q is contained in an object with a unique color. ◀

Application: Objects with Near-Linear Union Complexity. Har-Peled and Smorodinsky [15] proved that any family of objects with linear union complexity (for example disks, or pseudodisks) can be colored in a unimax fashion using $O(\log n)$ colors. In fact, their result is more general: if the union complexity is at most $n \cdot \beta(n)$ then the number of colors is $O(\beta(n) \log n)$. Note that for disks and pseudodisks we have $\beta(n) = O(1)$, for fat triangles we have $\beta(n) = O(\log^* n)$ [2] and for locally fat objects we have $\beta(n) = O(2^{O(\log^* n)})$ [2]. This directly implies the following result.

▶ **Corollary 10.** *Let $\mathcal{F}$ be a family of objects such that the union complexity of any subset of n objects from $\mathcal{F}$ is at most $n\beta(n)$. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions, such that the number of used colors is $O(\beta(n) \log^3 n)$ and the number of recolorings per insertion is $O(\log n)$, where n is the current number of objects in S.*

3.2 A Fully-Dynamic Solution

We now present a generalisation of our method to the fully-dynamic case. For lack of space we only give a brief sketch; details can be found in the full version [12]. As before, we assume we have a family $\mathcal{F}$ of objects such that any set of n objects from $\mathcal{F}$ can be unimax-colored with $\gamma_{\text{um}}(n)$ colors. We further assume that such a coloring admits *weak deletions*: once we have colored a given set S of n_0 objects using $\gamma_{\text{um}}(n_0)$ colors, we can delete objects from it using $r(n_0)$ recolorings per deletion such that the number of colors never exceeds $\gamma_{\text{um}}(n_0)$. The functions $\gamma_{\text{um}}(n)$ and $r(n)$ are assumed to be non-decreasing.

The insertions in this method are similar to those in the semi-dynamic solution, except that now a set S_i can be in four states: *empty*, *non-empty*, *in upwards migration*, and *in downwards migration*. It is worth pointing out that the upwards migrations are very similar to the migrations of the previous section, and that only S_ℓ (i.e., the last set) can be in downwards migration. The deletion procedure makes use of the weak deletions. Furthermore, since now a set S_i can have less than 2^i elements due to deletions, we need to make sure the number of set stays logarithmic. To that purpose, we make sure that the last set is at least half full. When this is no longer true, we combine the last three sets using a downwards migration, which is similar to the upwards migration. The details are fairly intricate and can be found in the full version [12]. We obtain the following theorem.

▶ **Theorem 11.** *Let $\mathcal{F}$ be a family of objects such that any subset of n objects from $\mathcal{F}$ admits a unimax coloring with $\gamma_{\text{um}}(n)$ colors and that allows weak deletions at the cost of*

$r(n)$ recolorings, where $\gamma_{\text{um}}(n)$ and $r(n)$ are non-decreasing. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions and deletions, such that the number of used colors is $O(\sum_{i=0}^{k}\gamma_{\text{um}}(2^i)\log n)$, where $k = \Theta(\log n)$. The number of recolorings per insertion is $O(\log n)$, and the number of recolorings per deletion is $O(r(8n)+1)$, where n is the current number of objects in S.

Application: Points with Respect to Rectangles. We now make use of Theorem 11 to maintain a CF-coloring of points with respect to rectangles. But first we present a simple technique to color points with respect to intervals in $\mathbb{R}^1$, which we use as a subroutine.

▶ **Lemma 12.** *We can maintain a unimax coloring of n points in $\mathbb{R}^1$ with respect to intervals under deletions, using $\lceil \log n_0 \rceil$ colors and at the cost of one recoloring per deletion. Here n_0 is the initial number of points.*

Proof. We start with a static unimax coloring of points with respect to intervals using $\lceil \log n_0 \rceil$ colors [18]. Recoloring after deleting a point p with color i is done as follows. If both neighbors of p have a higher color then we do nothing, otherwise we pick a neighbor with color smaller than i and recolor it to i. To prove the coloring stays unimax we only need to consider intervals I containing a neighbor of p. Now consider $I \cup \{p\}$. If before the deletion the maximum color was larger than i then that color is still present and unique. Otherwise i was the unique maximum color. Now either I contains a neighbor of p that was recolored to i, or no point in I was recolored; in both cases the maximum color in I is unique. ◀

Let now S be a set of points in the plane and $\mathcal{F}$ be the family of all rectangles in the plane. The following lemma shows how to perform weak deletions.

▶ **Lemma 13.** *There is a conflict free coloring of n points with respect to rectangles using $O(\sqrt{n}\log n)$ colors that allows weak deletions at the cost of one recoloring per deletion.*

Proof. We first partition the point set into at most $\sqrt{n}$ subsets such that each set is monotone using Dilworth's theorem [13]. Then, each point set behaves exactly as points with respect to intervals in one dimension. We can then apply Lemma 12 to finish the proof. ◀

▶ **Corollary 14.** *Let S be a set of points in the plane and $\mathcal{F}$ be a family of rectangles. Then we can maintain a CF-coloring on S under insertions and deletions such that the number of used colors is $O(\sqrt{n}\log^2 n)$ and the number of recolorings per insertion is $O(\log n)$ and $O(1)$ per deletion, where n is the current number of objects in S.*

4 Concluding Remarks

We studied the maintenance of a CF-coloring under insertions and deletions, presenting the first fully-dynamic solution for objects in $\mathbb{R}^2$. We showed how to maintain a CF-coloring for unit squares and for bounded-size rectangles, with $O(\log n)$ resp. $O(\log^2 n)$ colors and $O(\log n)$ recolorings per update. The method extends to arbitrary rectangles with coordinates from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors and $O(\log n)$ recolorings per update. We also presented general techniques for the semi-dynamic (insertions-only) and the fully-dynamic case (insertions and deletions). Our insertions-only technique can be applied to objects with near-linear union complexity, giving for instance a CF-coloring of $O(\log^3 n)$ colors for pseudodisks using $O(\log n)$ recolorings per update. Our fully-dynamic solution applies to any class of object on which weak deletions are possible, giving for instanct a CF-coloring of $O(\sqrt{n}\log^2 n)$ colors for points with respect to rectangles at the cost of $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

References

1 Deepak Ajwani, Khaled M. Elbassioni, Sathish Govindarajan, and Saurabh Ray. Conflict-free coloring for rectangle ranges using $o(n^{.382})$ colors. *Discrete & Computational Geometry*, 48(1):39–52, 2012. doi:10.1007/s00454-012-9425-5.

2 Boris Aronov, Mark de Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM Journal on Computing*, 43:534–572, 2014.

3 Amotz Bar-Noy, Panagiotis Cheilaris, Svetlana Olonetsky, and Shakhar Smorodinsky. Online conflict-free colouring for hypergraphs. *Combinatorics, Probability & Computing*, 19(4):493–516, 2010. doi:10.1017/S0963548309990587.

4 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

5 Panagiotis Cheilaris, Luisa Gargano, Adele A. Rescigno, and Shakhar Smorodinsky. Strong conflict-free coloring for intervals. *Algorithmica*, 70(4):732–749, 2014. doi:10.1007/s00453-014-9929-x.

6 Ke Chen. How to play a coloring game against a color-blind adversary. In *Proceedings of the 22nd ACM Symposium on Computational Geometry*, pages 44–51, 2006. doi:10.1145/1137856.1137865.

7 Ke Chen, Amos Fiat, Haim Kaplan, Meital Levy, Jiří Matoušek, Elchanan Mossel, János Pach, Micha Sharir, Shakhar Smorodinsky, Uli Wagner, and Emo Welzl. Online conflict-free coloring for intervals. *SIAM Journal on Computing*, 36(5):1342–1359, 2007. doi:10.1137/S0097539704446682.

8 Ke Chen, Haim Kaplan, and Micha Sharir. Online conflict-free coloring for halfplanes, congruent disks, and axis-parallel rectangles. *ACM Transactions on Algorithms*, 5(2):16:1–16:24, 2009. doi:10.1145/1497290.1497292.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

10 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008.

11 Mark de Berg, Tim Leijsen, Aleksandar Markovic, André van Renssen, Marcel Roeloffzen, and Gerhard Woeginger. Dynamic and kinetic conflict-free coloring of intervals with respect to points. In *Proceedings of the 28th International Symposium on Algorithms and Computation*, 2017.

12 Mark de Berg and Aleksandar Markovic. Dynamic conflict-free colorings in the plane. *CoRR*, abs/1709.10466, 2017.

13 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: http://www.jstor.org/stable/1969503.

14 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003. doi:10.1137/S0097539702431840.

15 Sariel Har-Peled and Shakhar Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry*, 34(1):47–70, 2005. doi:10.1007/s00454-005-1162-6.

16 Elad Horev, Roi Krakovski, and Shakhar Smorodinsky. Conflict-free coloring made stronger. In *Proceedings ot the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 105–117, 2010. doi:10.1007/978-3-642-13731-0_11.

17 Shakhar Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, Tel-Aviv University, 2003.

18 Shakhar Smorodinsky. Conflict-free coloring and its applications. *CoRR*, abs/1005.3616, 2010. URL: `http://arxiv.org/abs/1005.3616`.