Conflict-Free Coloring for Intervals: from Offline to Online

[extended abstract]

Amotz Bar-Noy*
Computer and Information Science Department
Brooklyn College
2900 Bedford Avenue
Brooklyn, NY, 11210

Panagiotis Cheilaris†
The Graduate Center
City University of New York
365 Fifth Avenue
New York, NY, 10016

Shakhar Smorodinsky‡
Courant Institute for Mathematical Sciences
New York University
251 Mercer Street
New York, NY, 10012

ABSTRACT

This paper studies deterministic algorithms for a frequency assignment problem in cellular networks. A cellular network consists of fixed-position base stations and moving agents. Each base station operates at a fixed frequency, and this allows an agent tuned at this frequency to communicate with the base station. Each agent has a specific range of communication (described as a geometric shape, e.g., a disc) that may contain one or several base stations. To avoid interference, the goal is to assign frequencies to base stations such that for any range, there exists a base station in the range with a frequency that is not reused by some other base station in the range. The base station with this unique (in the range) frequency serves the aforementioned range. Since using many frequencies is expensive, the optimization goal is to use as few frequencies as possible. The problem can be modeled as a special coloring problem for hypergraphs. Base stations are the vertices, ranges are the hyperedges, and colors (frequencies) must be assigned to vertices following the conflict-free property: In every hyperedge there is a color that occurs exactly once.

We concentrate on the special case where the n base stations lie on the real line and ranges are the $n(n+1)/2$ nonempty subsets of consecutive points. This problem is called conflict-free coloring for intervals. We introduce a hierarchy of four models for the above problem: (i) the static model, where the complete hypergraph is given and all vertices are colored simultaneously, (ii) the dynamic offline model, where the vertices appear in some order and the conflict-free property has to be maintained at all times, (iii) the online absolute positions model where the order is revealed in an online fashion and the final hypergraph and positions are known, and (iv) the online relative positions model where there is no knowledge about the final hypergraph and the final positions of vertices.

In the case of intervals, the hierarchy is strict. In the dynamic offline model, we give a deterministic algorithm that uses at most $\log_{3/2} n+1$ colors and exhibit inputs that force any algorithm to use at least $2\log_3 n + 1$ colors. For the online absolute positions model, we give two deterministic algorithms that use at most $2\lfloor \log_2(n+1) \rfloor$ and $3\lceil \log_3 n \rceil$ colors, respectively. To the best of our knowledge, these are the first $O(\log n)$ deterministic online algorithms, in a non-trivial model. In the online relative positions model, we resolve an open problem by showing a tight analysis on the number of colors used by the natural greedy online algorithm, that at each step uses the smallest color possible. In the case of conflict-free coloring only with respect to intervals that contain either of the two extreme points, we show a strong separation between static and dynamic models and we provide tight bounds for all four models up to an additive term of two.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of algorithms and problem complexity

General Terms

Algorithms, Theory

Keywords

Online algorithms, cellular networks, frequency assignment, conflict free, coloring

*Email: amotz@sci.brooklyn.cuny.edu

†Email: philaris@sci.brooklyn.cuny.edu

‡Email: shakhar@cims.nyu.edu

Work on this paper was supported by the NSF Mathematical Sciences Postdoctoral Fellowship award 0402492.

SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA.

1. INTRODUCTION

A *vertex coloring* of a graph $G(V, E)$ is a function $\chi\colon V \to \mathbb{N}^+$ such that for every edge $\{v_1, v_2\} \in E$: $\chi(v_1) \neq \chi(v_2)$. A *hypergraph* $G(V, E)$ is a generalization of a graph for which *hyperedges* can be arbitrary-sized non-empty subsets of V. There are several ways to define vertex coloring in hypergraphs: On one extreme, it is required that for every hyperedge, not all colors are the same (there are at least two colors); on the other extreme, it is required that for every edge, no color is repeated (all the colors are different). In between these two extremes, there is another possible generalization: A vertex coloring χ of hypergraph $G(V, E)$ is called *conflict-free* if in every hyperedge e there is a vertex whose color is unique among all other colors in the hyperedge. Formally, $\forall e \in E\colon \exists v \in e\colon \forall v' \in e\colon v' \neq v \to \chi(v') \neq \chi(v)$.

Conflict-free coloring models frequency assignment for cellular networks. A cellular network consists of two kinds of nodes: *base stations* and *mobile agents*. Base stations have fixed positions and provide the backbone of the network; they are modeled by vertices in V. Mobile agents are the clients of the network and they are served by base stations. This is done as follows: Every base station has a fixed frequency; this is modeled by the coloring χ, i.e., colors represent frequencies. If an agent wants to establish a link with a base station it has to tune itself to this base station's frequency. Since agents are mobile, they can be in the range of many different base stations. The range of communication of every agent is modeled by a hyperedge $e \in E$, which is the set of base stations that can communicate with the agent. To avoid interference, the system must assign frequencies to base stations in the following way: For any range, there must be a base station in the range with a frequency that is not reused by some other base station in the range. This is modeled by the conflict-free property. One can solve the problem by assigning n different frequencies to the n base stations. However, using many frequencies is expensive, and therefore, a scheme that reuses frequencies, where possible, is preferable.

The study of conflict-free colorings was originated in the work of Even et al. [6] and Smorodinsky [12]. In addition to the practical motivation described above, this new coloring model has drawn much attention of researchers through its own theoretical interest and such colorings have been the focus of several recent papers (see, e.g., [6, 12, 10, 8, 7, 9, 5, 13, 1]).

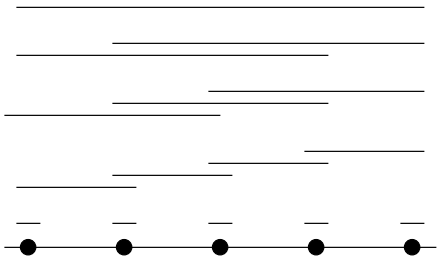


Figure 1: Points on a line and intervals

The paper [7] considered the special case of the problem where the hypergraph is defined as follows: Vertices are identified by points that lie on a line and E consists of all subsets of V defined by intervals intersecting at least one vertex. A line with n points has $n(n+1)/2$ such subsets (for every $i \in \{1, \ldots, n\}$, there are $n - i + 1$ different subsets containing i points). For $n = 5$, these subsets are shown in figure 1. We call these subsets intervals because for our problem, two intervals are equivalent if they contain the same vertices. We represent colorings by listing the colors of points from left to right in a *string*. For example, for the points in figure 1 ($n = 5$), 12312 is a conflict-free coloring, whereas 12123 is not.

Conflict-free coloring for intervals is important because it can model assignment of frequencies in networks where the agents' movement is approximately unidimensional, e.g., the cellular network that covers a single long road and has to serve agents that move along this road. Also, conflict-free coloring for intervals plays a role in conflict-free coloring for more complicated range spaces (see [6]).

The *static* version of the problem, where the n points are to be colored simultaneously, is solved in [6]. For $n = 2^k - 1$, the coloring χ^k is defined recursively as follows: $\chi^k = 1$ and $\chi^{k+1} = \chi^k \circ (k+1) \circ \chi^k$ (where $\circ$ is the concatenation operator for strings). Coloring χ^k is conflict-free, and for n with $n < 2^k - 1$ the prefix of length n of χ^k is conflict-free. This paper also shows that this coloring with $1 + \lfloor \lg n \rfloor$ colors is the best possible.

The problem becomes more interesting when the vertices are given online by an adversary. Namely, at every given time step $t \in \{1, \ldots, n\}$, a new vertex $v_t \in V$ is given and the algorithm must assign v_t a color such that the coloring is a conflict-free coloring of the hypergraph that is induced by the vertices $V_t = \{v_1, \ldots, v_t\}$. Once v_t is assigned a color, that color cannot be changed in the future. This is an online setting, so the algorithm has no knowledge of how vertices will be given in the future. For this version of the problem, in the case of intervals, Fiat et al. [7] provide several algorithms. Their randomized algorithm uses $O(\log n \log\log n)$ colors with high probability. Their deterministic algorithm uses $O(\log^2 n)$ colors in the worst case. That algorithm requires $\Omega(log^2 n)$ colors on some inputs. Recently, randomized algorithms that use $O(\log n)$ colors with high probability have been introduced in a slightly weaker adversary model ([4, 2]). In the weaker model, the adversary has to commit on a specific input sequence before revealing the first vertex to the algorithm. In the terminology of [11, 3], this is an *oblivious* adversary.

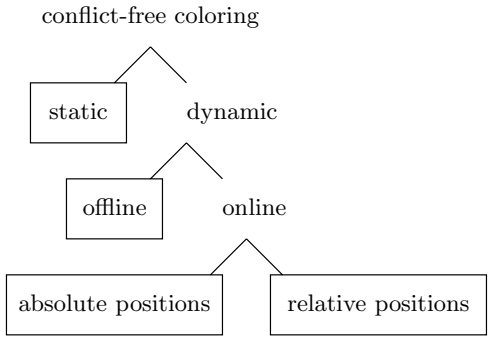


Figure 2: Models for conflict-free coloring for intervals

Our contribution. We introduce a *hierarchy* of four models for the above conflict-free coloring problem for hypergraphs:

(i) static, (ii) dynamic offline, (iii) dynamic online with absolute positions, and (iv) dynamic online with relative positions. Below we define these four models. The relationship among them is shown in figure 2.

- In the *static model*, the complete hypergraph G is given, and a conflict-free coloring for G must be found by the algorithm.

In dynamic models, a *sequence* $\{G_t\}_{t=1}^n$ of hypergraphs (with $G = G_n$) is given where G_t has t vertices and, for $t > 1$, G_{t-1} is an induced subhypergraph of G_t; for every t a conflict-free coloring for G_t must be found that, for $t > 1$, extends the coloring of G_{t-1} (i.e., the algorithm can not change colors of vertices). Alternatively, the input is a permutation of the vertices of the final hypergraph, and G_t, the hypergraph to be colored at every time step, is the subhypergraph of G induced by the first t vertices in the permutation.

- In the *dynamic offline model*, the complete sequence $\{G_t\}_{t=1}^n$ is given.

In dynamic online models, the sequence $\{G_t\}_{t=1}^n$ is revealed incrementally, at discrete time steps $t = 1, \ldots, n$, i.e., at time t, G_t is given, and a color for the new vertex v_t must be found without knowledge of future $G_{t'}$, where $t' > t$.

- In the *dynamic online with absolute positions model*, in addition to $\{G_t\}_{t=1}^n$ being revealed incrementally, the final $G_n = G$ is given from the start as a *vertex-labeled* hypergraph and for each time t, G_t is also given as a vertex-labeled hypergraph, with the induced subhypergraph isomorphism between G_t and G_n *preserving* the labels. This means that the algorithm knows for every new vertex v_t where it is going to lie (i.e., its 'absolute' position) in the final hypergraph G.

- In the *dynamic online with relative positions model*, no information about the final hypergraph G is given (not even its size n). The only information might be the structure of the final hypergraph (for example, if we are coloring points on a line with respect to intervals). Thus, for every new vertex v_t, we only know its 'relative' position with respect to already inserted points, by means of the information in G_t.

Related to applications, revealing absolute positions is not unnatural in many cases: One can think of all base stations being at fixed positions, which are known to the algorithm in advance. This means that the algorithm is aware of the final hypergraph that models the situation where all base stations are activated. In the start, no base station is activated. Base stations are constructed or activated one by one, in some order, in response to increasing network traffic. Every new station has to be given a color by the algorithm, such that the conflict-free property is maintained.

In the case of intervals, the four models produce a *strict hierarchy* of models, in the sense that an adversary in a higher model has more power and an algorithm for a higher model works also for a lower model. A summary of results for deterministic algorithms for conflict-free coloring with respect to intervals is given in table 1. All the online algorithms considered so far in the literature work in the *relative positions* model. Our main technical results concern two deterministic algorithms that use $2\lfloor \lg(n+1) \rfloor$ and $3\lceil \log_3 n \rceil \approx 1.89 \lg n$ colors, respectively, in the slightly changed online model of *absolute positions*. We also exhibit sequences of length $n = 3^k$ that need at least $1 + 2\log_3 n \approx 1.26 \lg n$ colors in any dynamic model, and then we outline an algorithm that uses at most $1 + \log_{3/2} n \approx 1.71 \lg n$ colors in the dynamic offline model. In the relative positions model, we resolve an open problem posed in [7]: We give a tight analysis on the performance of the *fully greedy online algorithm* (FG), and prove that it uses $\lfloor n/2 \rfloor + 1$ colors in the worst case.

Finally, we discuss coloring with respect to a specific subset of all intervals. One interesting case is coloring with respect to *rays* (or *halflines*), namely the subset of intervals that contain either of the two extreme points, for which we show a strong separation between static and dynamic offline models, in the sense that in the static model three colors suffice for any n, whereas in the dynamic offline model $\lfloor \lg n \rfloor + 1$ colors might be needed. On the other hand, $\lfloor \lg(n-2) \rfloor + 3$ colors are enough even in the relative positions model.

Paper organization. In section 2, we introduce two ways to describe input to dynamic and online algorithms in the case of intervals. In section 3, we consider the dynamic offline model. In section 4, we give $O(\log n)$ algorithms in the absolute positions model. In section 5, we analyze the worst-case behavior of the fully greedy algorithm. In section 6, we study conflict-free coloring with respect to rays. In section 7, we discuss some of the results and mention open problems.

2. PRELIMINARIES

We show two ways to represent inputs for dynamic models, in the intervals case. We will be using them in subsequent sections.

In the *relative positions model*, the sequence of points inserted can be described by the position in which each new point is inserted, relative to previously inserted points. If $i-1$ points have already been inserted, the i-th point can be inserted in any of i positions described by an integer in the range $[0, i-1]$: 0 is for the new point in the start of the sequence (before any other point), and $k > 0$ is for the new point immediately after the k-th already inserted point.

An insertion sequence of length n is represented by a string of n integers, σ, where $0 \leq \sigma_{(i)} \leq i-1$. If we consider insertion sequences of the same length n ordered lexicographically, then the first and last elements in that order are: $s_n^{\text{first}} = [0, 0, 0, \ldots, 0]$, $s_n^{\text{last}} = [0, 1, 2, \ldots, n-1]$. In the relative positions online model, an insertion sequence is revealed from left to right, one by one, to the online algorithm. There are $n!$ possible insertion sequences of length n.

In the *absolute positions model*, initially the algorithm knows the total number of points to be inserted. Then, for each new point the absolute position of that point in the final sequence is revealed to the algorithm. The absolute position can be any number in $\{1, \ldots, n\}$ which has not appeared before. Thus, the input to the algorithm is a permutation $\pi \in \mathbf{S}_n$ that is revealed one by one, from left to right.

In the dynamic *offline* setting, the input can be given in either absolute or relative positions, because the two representations are easily convertible to each other if the *whole*

model	lower bound	upper bound
dynamic online, relative positions	$1+2\log_3 n$ [this paper]	$O(\log^2 n)$ [7]
dynamic online, absolute positions	$1+2\log_3 n$ [this paper]	$3\lceil\log_3 n\rceil$ [this paper]
dynamic offline	$1+2\log_3 n$ [this paper]	$1+\log_{3/2} n$ [this paper]
static	$1+\lfloor\log_2 n\rfloor$ [6]	$1+\lfloor\log_2 n\rfloor$ [6]

Table 1: Number of colors used in deterministic algorithms for intervals

sequence is known. For example, the insertion sequence $\sigma = 00121$ corresponds to the permutation $\pi = 51342$, which means the first point inserted is at the 5^{th} absolute position (rightmost), the second point inserted is at the 1^{st} absolute position (leftmost), and so on.

3. DYNAMIC OFFLINE MODEL

Lower bound. For every k, we exhibit an insertion sequence π^k of absolute positions that has length $n = 3^k$ and needs $2k+1 = 2\log_3 n + 1$ colors to be conflict-free colored.

Definition 1. Given a string π of numbers and $x \in \mathbb{N}$, we define the string $(\pi + x)$ by adding x to each element of π, i.e., $(\pi + x)_{(i)} = \pi_{(i)} + x$, for $i \in \{1, \ldots, \text{len}(\pi)\}$.

The insertion sequences are defined recursively as follows:

$$\pi^1 = 132 \quad \text{and} \quad \pi^{k+1} = \pi^k \circ (\pi^k + 2\cdot 3^k) \circ (\pi^k + 3^k)$$

For example, $\pi^2 = 132798465$, or in relative positions notation $\sigma^2 = 011344344$.

PROPOSITION 1. *Input π^k needs at least $2k+1$ colors in the dynamic model.*

PROOF OUTLINE. The proof is by induction. The inductive step is based on the fact that π^k consists of three shifted π^{k-1} insertion orders that correspond to the leftmost 3^{k-1}, middle 3^{k-1}, and rightmost 3^{k-1} points. Each of these three parts of the coloring has to use at least $2k-1$ colors (by the inductive hypothesis). However, maintaining the conflict-free property for intervals that span more than one of the three parts needs $2k+1$ colors in total: It can be proved that after the insertion of the first $2\cdot 3^{k-1}$ points of π^k at least $2k$ colors have to be used, and after the insertion of all points, one additional new color has to be used, which brings the total number of colors to $2k+1$. □

Upper bound. The dynamic offline case can be viewed as a static problem, because dynamically coloring the sequence $\{G_t\}_{t=1}^n$ is equivalent to statically coloring the hypergraph $G(V, \bigcup_{t=1}^n E_t)$, where E_t is the hyperedge set of G_t. In [6], a general framework for conflict-free coloring is presented: The authors provide an algorithm (Algorithm 1 in the paper) that colors the points in iterations. At the ℓ-th iteration, some points are colored with color ℓ and these colored points are not considered in the subsequent iterations. It is not hard to prove the following:

PROPOSITION 2. *If the algorithm in [6] is applied to a hypergraph $G(V, \bigcup_{t=1}^n E_t)$ arising from intervals, then in each iteration of the algorithm at least a third of the remaining points is colored. As a result, the algorithm uses at most $\log_{3/2} n + 1$ colors.*

4. ABSOLUTE POSITIONS MODEL

In this section we present two algorithms that use $O(\log n)$ colors in the absolute positions model. Both of the algorithms use recursion, but in a different way, to choose points that will have a unique color for each interval. In a sense, a point p with a unique color in an interval acts as a *separator*: Points to the left of p and points to the right of p can be colored independently, and colors can be freely reused. Choice of the right separators means that a few colors are used.

The first algorithm chooses the separators in each level and uses the recursion to color the independent sections to the left and to the right of separators. The second algorithm adopts the opposite approach of coloring two thirds of the points with a greedy non-recursive scheme in each level and of coloring the separators by using the recursion. We opt to show both approaches because we feel that both methodologies are instructive and we hope to get better results with either one.

4.1 An asymptotically $2\log_2 n$ algorithm

We present a recursive algorithm in the absolute positions model, that uses $2\lfloor\lg(n-1)\rfloor + 2$ colors to color any input of size n. The core of the algorithm is a way to color any permutation π of absolute positions of points with $b = b(n) = 2|\text{bin}(n-1)|$ colors[1], such that the two maximum colors $u_b^L = b-1$ and $u_b^R = b$ occur uniquely and 'relatively close' to the center of the coloring. Color u_b^L appears to the left of the center and color u_b^R appears to the right. 'Relatively close' means that these unique colors both appear somewhere in the $n/2$ central points of the coloring (each at most $n/4$ from the center point). We denote the maximum number of points the algorithm can color with b colors by $\hat{n}(b) = 2^{b/2}$.

Formally, given $l \le \hat{n}(b(n))/2$, $r \le \hat{n}(b(n))/2$, the following intervals are defined, in a $l + r = n$ length coloring:

- *leftmost interval*: from 1 to $\lfloor l/2\rfloor$
- *left middle interval*: from $\lfloor l/2\rfloor + 1$ to l
- *right middle interval*: from $l+1$ to $l + \lceil r/2\rceil$
- *rightmost interval*: from $l + \lceil r/2\rceil + 1$ to $l + r$

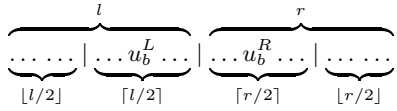
(l and r can also be 0 or 1, in which case, some intervals are empty). The leftmost and the left middle intervals comprise the *left part* of the coloring and the right middle and rightmost intervals comprise the *right part* of the coloring. The size of the intervals is shown below:

$$\overbrace{\underbrace{\text{leftmost}}_{\lfloor l/2\rfloor} \mid \underbrace{\text{left middle}}_{\lceil l/2\rceil}}^{l} \mid \overbrace{\underbrace{\text{right middle}}_{\lceil r/2\rceil} \mid \underbrace{\text{rightmost}}_{\lfloor r/2\rfloor}}^{r}$$

[1]For $x \in \mathbb{N}^+$, $|\text{bin}(x)|$ is the length of the binary representation of x, which is $1 + \lfloor\lg x\rfloor$.

We remark that there is a slight preference to the middle intervals (use of the ceiling, instead of the floor function). This will prove helpful later in the proof of correctness of the algorithm.

The algorithm colors any permutation π of length n, in a way such that at most b colors are used and if $l \geq 1$, then u_b^L occurs uniquely, somewhere in the left middle interval and if $r \geq 1$, then u_b^R occurs uniquely, somewhere in the right middle interval:



This is achieved by reserving color u_b^L for the first point that appears in the left middle interval and never reusing it, and by reserving u_b^R for the first point that appears in the right middle interval and never reusing it.

Initially, if the algorithm is given n points, in order to use $b = b(n)$ colors, it has to partition the points into two parts such that $l \leq \hat{n}(b)/2$ and $r \leq \hat{n}(b)/2$, so, one possible choice is $l = \lfloor n/2 \rfloor$ and $r = \lceil n/2 \rceil$, which tries to balance the sizes of the two parts. The left part of the coloring is colored by a subroutine L and the right part by a subroutine R. Also, the insertion of the first point in a middle interval determines the way recursion will be applied to the remaining points in this middle interval: Points to the left of this first inserted point are colored with a recursive call to R and points to the right with a recursive call to L (notice the contrast between position of points and subroutine applied to them). The recursion is shown in detail in figure 3 (L_x signifies use of subroutine L with x colors; similarly for R_x).

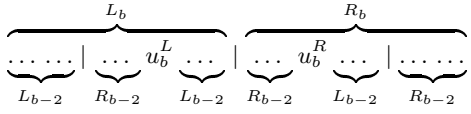


Figure 3: The recursion in a coloring

The algorithm is named 2CU (from 2 Central Unique colors) and is given in figure 4. A property of the algorithm is that the coloring of the two parts (left and right) are *independent* of each other.

Correctness of the algorithm. Since a coloring is just a concatenation of a part that is colored by an L_b subroutine, followed by a part that is colored by an R_b subroutine, it is enough to prove the following proposition.

PROPOSITION 3. *If L_b is applied on any l points with $0 \leq l \leq \hat{n}(b)/2$, it gives a legal coloring C_b^L at all times (i.e., in the dynamic online sense). If R_b is applied on any r points with $0 \leq r \leq \hat{n}(b)/2$, it gives a legal coloring C_b^R at all times (i.e., in the dynamic online sense). Moreover, any partial C_b^L can be concatenated with any partial C_b^R to give a legal coloring $C_b^L \circ C_b^R$, at all times.*

PROOF. By induction on the number of colors used at most (i.e., b colors).

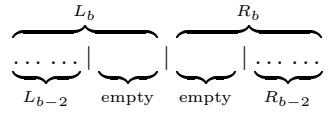
Base (b is at most 2): For $0 \leq l \leq 1$, $0 \leq r \leq 1$ the colorings are shown in table 2 and are correct.

Inductive step: Assume the hypothesis is true for $b-2$ colors, then we will prove it is true for b colors.

$l \setminus r$	0	1
0	ε	2
1	1	1 2

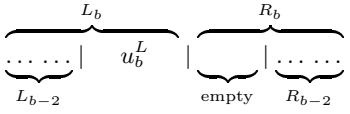
Table 2: Colorings that use less than or equal to 2 colors

As long as no point in the two middle intervals has appeared, the coloring is done only in the leftmost and the rightmost interval, using colors up to $b-2$:



By induction, the coloring up to that point is legal at all times, because each interval has size at most $\hat{n}(b)/4 = \hat{n}(b-2)$ (so $b-2$ colors suffice) and by the induction these colorings of the leftmost and rightmost interval can be concatenated to give a legal coloring.

Then, eventually, some point in the middle intervals is requested. W.l.o.g., assume it is in the left middle interval, so it gets color u_b^L:

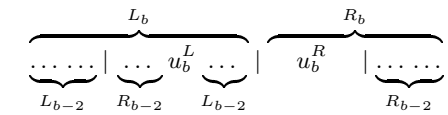


This u_b^L color will remain unique, so for proving correctness, from now on, we have to consider only intervals not containing u_b^L.

From now on, points in the left middle interval that are before the u_b^L-colored one are colored with subroutine R_{b-2} (see figure 3). The size of this interval where R_{b-2} is used is at most $\hat{n}(b)/4 = \hat{n}(b-2)$, so this interval is legally colored, by the inductive hypothesis. Also, when combined with the leftmost interval which is colored by L_{b-2} and which also has size at most $\hat{n}(b)/4 = \hat{n}(b-2)$, those two intervals concatenated together, are always legally colored.

For points to the right of the u_b^L-colored one, as long as no point in the right middle interval has been requested, either the L_{b-2} subroutine is used for points in the left middle interval (see figure 3), or the R_{b-2} subroutine is used for points in the rightmost interval. These two intervals are both of length at most $\hat{n}(b)/4 = \hat{n}(b-2)$, so by induction, are always legally colored, and their concatenation is legally colored.

However, as soon as the first point in the right middle interval is requested, it gets color u_b^R and this color will remain unique:



By the uniqueness of u_b^R, from now on we only have to consider intervals that do not contain this u_b^R-colored point.

In the right middle interval and to the left of the u_b^R-colored point, the R_{b-2} subroutine is used, which, combined with the L_{b-2} colored interval to the right of u_b^L, by induction, gives a legal coloring. Also the right middle interval points to the right of the u_b^R-colored point, are colored with the L_{b-2} subroutine, which combined with the R_{b-2} colored

```
coloring algorithm 2CU with b colors:
    break the n points in a left and a right part
      (of lengths l = ⌊n/2⌋, r = ⌈n/2⌉)
    for each point appearing online do
        if point is in left part
            color it with subroutine L and b colors
        if point is in right part
            color it with subroutine R and b colors

subroutine L with b colors:
    break the left part in a leftmost and a left middle interval
    if the point is in the leftmost interval
        color it with subroutine L and b − 2 colors
    if the point is in the left middle interval
        if the left middle interval is empty
            give color u_b^L = b − 1 to the point
        else (u_b^L has been used in this left middle interval)
            if point is to the left of u_b^L in the left middle interval
                color it with subroutine R and b − 2 colors
            if point is to the right of u_b^L in the left middle interval
                color it with subroutine L and b − 2 colors

subroutine R with b colors:
    break the right part in a right middle and a rightmost interval
    if the point is in the rightmost interval
        color it with subroutine R and b − 2 colors
    if the point is in the right middle interval
        if the right middle interval is empty
            give color u_b^R = b to the point
        else (u_b^R has been used in this right middle interval)
            if point is to the left of u_b^R in the right middle interval
                color it with subroutine R and b − 2 colors
            if point is to the right of u_b^R in the left middle interval
                color it with subroutine L and b − 2 colors
```

Figure 4: Algorithm 2CU

rightmost interval, by induction, is always legally colored (check figure 3; all mentioned intervals are of length at most $\hat{n}(b)/4 = \hat{n}(b-2)/2$). □

Slight improvements over the 2CU algorithm. As we mentioned, algorithm 2CU can color up to $2^{b/2}$ points if given b colors. Without any significant change to the algorithm, one can prove that if the algorithm is given b colors it also works correctly for all instances that have up to $2 \cdot 2^{b/2} - 2$ points (this is almost the double number of points). Equivalently, the algorithm can be made to use $2\lfloor \lg(n+1) \rfloor$ colors.

One can also use low numbered colors for the central uniquely colored points and use colors based on the size of the subintervals that are to be colored in the recursion (i.e., not always $b-2$, but maybe less, because the subinterval can be small). This can also lead to some decrease of colors in some instances, but it does not give significant improvement in worst case instances.

4.2 An asymptotically $3 \log_3 n$ algorithm

We sketch a recursive algorithm in the absolute positions model, that uses $3\lceil \log_3 n \rceil$ colors to color any input of size n. Asymptotically, this is $1.89 \lg n$, and thus the algorithm performs better than 2CU, which was given in the previous subsection. Triples of consecutive points play a major role in the algorithm and this is why we call it the 'triples' algorithm.

To prove the above bound, it suffices to show a method of conflict-free coloring any input of size 3^k with $3k$ colors, because, in that case, if $3^{k-1} < n \leq 3^k$ then the algorithm takes the n-sized input, attaches (in any insertion order) $3^k - n$ dummy points to the right of the n points, solves the 3^k-sized instance with the method to get a conflict-free coloring with $3k$ colors, and then it discards the colors of the dummy points to get a conflict-free coloring of the original n points.

If $n = 3^k$, points are colored in k levels that correspond to recursion call levels of the algorithm and each level uses three colors. At each level $\ell \in \{1, \ldots, k\}$, some of the points are colored and the rest are deferred for coloring at a higher level. More precisely at each level ℓ, with $\ell < k$, two thirds of the points are colored in that level and the rest (one third) are deferred. Thus, for each level $\ell < k$ of the recursion, out of the $3^{k+1-\ell}$ points that reach the level, $2 \cdot 3^{k-\ell}$ are colored in that level and $3^{k-\ell}$ are deferred for coloring in

recursion level	input size	points colored	points deferred	colors used
1	3^k	$2 \cdot 3^{k-1}$	3^{k-1}	1, 2, 3
...	...	...	...	...
ℓ	$3^{k+1-\ell}$	$2 \cdot 3^{k-\ell}$	$3^{k-\ell}$	$3\ell-2$, $3\ell-1$, 3ℓ
...	...	...	...	...
$k-1$	9	6	3	$3k-5$, $3k-4$, $3k-3$
k	3	3	0	$3k-2$, $3k-1$, $3k$

Table 3: Recursion levels of the triples algorithm

a higher level. The final level k is special because all three points that reach it are colored in that level. This situation is shown in table 3, where, by convention, level ℓ uses colors $3\ell-2$, $3\ell-1$, and 3ℓ.

Now, we describe how the algorithm decides at each level which points to color and which to defer. At each level ℓ, with $\ell < k$, the algorithm partitions the points in triples, according to their absolute positions: The three leftmost points are in the first triple, the second three leftmost points are in the second triple, and so on, until the final triple which contains the three rightmost points. *For every triple, the first point that is requested to be colored in the triple is deferred for coloring in a higher level, whereas the other two points are colored at level ℓ.* Also, for $\ell < k$, the input at level ℓ, which we denote by $\pi_{\langle \ell \rangle}$, induces an input $\pi_{\langle \ell+1 \rangle}$ at level $\ell+1$ as follows: The absolute positions of triples at level ℓ give the absolute positions of points and points in $\ell+1$ are requested in the same order as the first points of triples in ℓ. Initially, the input at level 1, i.e., $\pi_{\langle 1 \rangle}$, is set equal to the original input π. For example, consider the input $\pi = 923745618$, revealed to the online algorithm, one by one element, from left to right. In order to exhibit better how the algorithm runs, we take the inverse permutation of π, which maps absolute positions of points to the time they are requested:

$$\pi^{-1} = \pi^{-1}_{\langle 1 \rangle} = \ 8\mathbf{23}\ \mathbf{5}67\ 49\mathbf{1}$$

This is the input for level 1 (as denoted by the subscript) and we also have highlighted the first point requested in every triple. The above induces the following input for level 2: $\pi^{-1}_{\langle 2 \rangle} = 231$, or $\pi_{\langle 2 \rangle} = 312$.

Now, we explain how the algorithm colors points in a specific level. If a new point p is requested that is decided to be colored in level ℓ (i.e., not deferred for coloring in a higher level), the algorithm finds the set of all points P already colored at level ℓ with the following property: p' is in P, if there is no point deferred for coloring in a higher level between p and p'. It is not hard to see that there are at most three already inserted (and colored) points with this property. The algorithm chooses the color of p using a greedy coloring scheme, i.e., by choosing the minimum color possible among $3\ell-2$, $3\ell-1$, and 3ℓ, so that the interval containing $P \cup \{p\}$ remains conflict-free. The coloring at each level ℓ, corresponding to input $\pi_{\langle \ell \rangle}$, is denoted by $\chi_{\langle \ell \rangle}$; it is a partial coloring for $\ell < k$, because only two thirds of the points are colored.

The run of the algorithm on the example input mentioned above is shown in figure 5. The '$*$' denote points that are to be colored in a higher level and χ denotes the final coloring.

The correctness of the algorithm is immediate from the following result:

$\pi^{-1}_{\langle 1 \rangle} =$	8	2	3	5	6	7	4	9	1
$\chi_{\langle 1 \rangle} =$	1	$*$	1	$*$	1	3	2	1	$*$
$\pi^{-1}_{\langle 2 \rangle} =$		2		3					1
$\chi_{\langle 2 \rangle} =$		5		6					4
$\chi =$	1	5	1	6	1	3	2	1	4

Figure 5: A run of the triples algorithm

PROPOSITION 4. *At any time $t \in \{1, \ldots, n\}$, in any interval I of points, there is a point in I colored with a unique color in I. Moreover, a uniquely colored point can always be found among the points that were colored in the deepest recursive level of points in I.*

The following lemma is helpful in the proof of the above:

LEMMA 1. *Any conflict-free coloring of $x < 4$ points, can be extended to a conflict-free coloring of $x+1$ points, with at most three colors, for any position of the $x+1$-th point, by using the greedy coloring scheme.*

Proofs of proposition 4 and lemma 1 are not hard and are omitted in this short version of the paper.

5. RELATIVE POSITIONS MODEL

In this section, we analyze an algorithm in the relative positions model. The *fully greedy* algorithm (FG) for online conflict-free coloring for intervals, mentioned in [7], works as follows: For the next point to color, it chooses the minimum color that maintains the conflict-free coloring property. For example, the greedy algorithm colors insertion sequence $\sigma = 010322$ ($\pi = 251643$ in absolute positions) as follows: [.1....], [.1..2.], [21..2.], [21..23], [21.323], [214323]. We have the following tight result for FG:

THEOREM 1. *For $n \geq 2$, the maximum number of colors used by FG for inputs of length n is $\lceil n/2 \rceil + 1$.*

In order to establish the above result, in the following, we prove a lower bound for FG and a matching upper bound.

Lower bound for FG. There are sequences which force the FG algorithm to use $O(n)$ colors: We will prove that sequence $00(20)^i1$, of length $2i+3$, uses $i+3$ colors. We need the following lemma:

LEMMA 2. *Insertion sequence $00(20)^i$ uses $i+2$ colors and the two leftmost colors in the coloring are $i+2$, $i+1$. The third and fourth leftmost colors, in case $i > 0$ are i, $i+1$.*

PROOF OUTLINE. By induction. Base case ($i = 0$ and 1): 00 gives the coloring 21 and 0020 gives the coloring 3212. For the Inductive step: By induction, the coloring for $i > 0$ is:

$$c^i = \quad i+2 \quad i+1 \quad i \quad i+1 \quad \dots$$

The next insertion (at position 2) is between $i+1$ and i. It has to get color $i+2$, because if it would get another smaller color ($i+1$ is also impossible), then this color could also be used as the color occurring in the leftmost position of c^i. The coloring becomes:

$$i+2 \quad i+1 \quad i+2 \quad i \quad i+1 \quad \dots$$

The next insertion (at position 0) can not get a color among $i+2$, $i+1$, i. It can not get any other already used color, because then this color could also be used as the color occurring in the leftmost position of c^i. So, a new color has to be used and we get:

$$c^{i+1} = \quad i+3 \quad i+2 \quad i+1 \quad i+2 \quad \dots$$

□

LEMMA 3. *Insertion sequence* $00(20)^i1$ *uses* $i+3$ *colors.*

PROOF OUTLINE. We augment the coloring of $00(20)^i$ we have from lemma 2, which is:

$$i+2 \quad i+1 \quad \dots$$

The insertion (at relative position 1) between colors $i+2$ and $i+1$ has to get a new color, because any other color, except $i+2$ which is in any case impossible, would be chosen by the FG algorithm when coloring $00(20)^i$, in the last insertion. □

This establishes the following lower bound on the number of colors used by FG:

PROPOSITION 5. *For every* $n \geq 2$*, there are insertions sequences of length* n *that force the fully greedy algorithm to use* $\lceil n/2 \rceil + 1$ *colors.*

Upper bound for FG. In order to prove an upper bound on the number of colors used by the FG algorithm, we consider *uniquely occurring colors* in a coloring.

LEMMA 4. *In any FG coloring there are at most three distinct colors with the following property: each of these colors occurs exactly once.*

PROOF OUTLINE. Assume the three colors x, y, z that occur uniquely in a coloring by FG:

$$\dots x \dots y \dots z \dots$$

W.l.o.g., a new point can be inserted:

- either between y and z, in which case color x is eligible (gives a conflict-free coloring),
- or after z, in which both colors x and y are eligible.

In any case, FG will introduce no new color for the new point, since FG chooses the minimal eligible color. □

We remark that we can have 1, 2, or 3 uniquely occurring colors in a coloring by FG, as exhibited by the insertion sequence $\sigma = 011$ which is colored as 132 by FG.

PROPOSITION 6. *For* $n \geq 2$*: No insertion sequence of length* n *forces the fully greedy algorithm to use more than* $\lceil n/2 \rceil + 1$ *colors.*

PROOF. Consider a coloring by FG of n points. If k colors are used and u of them occur uniquely, then $k-u$ colors occur at least as duplicates, and $n \geq 2(k-u)+u$, which gives $2k \leq n+u$ and since k is integer:

$$k \leq \left\lfloor \frac{n+u}{2} \right\rfloor \leq \left\lfloor \frac{n+3}{2} \right\rfloor = \left\lceil \frac{n}{2} \right\rceil + 1$$

because $u \leq 3$ (by lemma 4). □

Theorem 1 follows immediately from propositions 5 and 6.

Remark 1. The upper bound technique for FG can also be applied to the *unique max* algorithm (UM), a simple algorithm that is used as a component in other more elaborate algorithms, including the $O(\log^2 n)$ algorithm of [7]. Our technique gives an upper bound of $\lceil n/2 \rceil + 2$ for the number of colors used by UM. However, in contrast to the tight analysis for FG, only insertion sequences that force UM to use $\Theta(\sqrt{n})$ colors have been found (see [7]).

6. COLORING WITH RESPECT TO A SUBSET OF THE SET OF ALL INTERVALS

We relax the conflict-free coloring problem of intervals as follows. Instead of the requirement that *all* intervals need to have a uniquely colored point, it is required that the conflict-free condition holds only for intervals in a specific *subset* of the set of all intervals. Examples are coloring with respect to all intervals of a specific length, say k, or all intervals of length up to k.

Another interesting case arises from the intervals that contain either of the two extreme points. Equivalently, these are intervals that are defined by halflines (infinite intervals), or *rays*. We therefore refer to the problem as conflict-free coloring with respect to *rays*. The motivation for considering this restricted subset comes from agents whose movement range is not strictly inside the line segment between the two extreme points. We also want to point out how different are the results and the gaps between models, related to the all intervals case.

For n points there are $2n-1$ ray defined intervals, of which n contain the leftmost point and are called *prefix* intervals and n contain the rightmost point and are called *suffix* intervals (the interval containing all points is both a prefix and a suffix interval).

In the static model, the coloring $133\dots332$ (i.e., color the extreme points with unique colors and use the same color for all non-extreme points) suffices for all n and uses three colors. It is not hard to see that three colors are required for $n \geq 4$ (for $n = 3$ the coloring 121 with two colors is a conflict-free coloring).

To analyze the problem in the dynamic models, we consider first coloring with respect to *prefix intervals* only (the suffix case has the same bounds, because it is symmetric). In the static model for prefixes, the coloring $122\dots22$ is a conflict-free coloring with 2 colors. Obviously, this coloring is optimal. In the dynamic models for prefixes, we will

first prove a lower bound of $1 + \lfloor \lg n \rfloor$ already for the dynamic offline model and then provide an algorithm using $2 + \lfloor \lg(n-1) \rfloor$ colors already in the relative position model.

PROPOSITION 7. *In the dynamic (offline) model, input $\sigma = 0^n$ needs $1 + \lfloor \lg n \rfloor$ colors to be conflict-free colored with respect to prefixes.*

PROOF OUTLINE. The i-th point inserted is always at the left of all previously inserted points and thus contributes i new intervals. In fact, by viewing the dynamic problem as a static problem (as we did in section 3), it can be proved that coloring 0^n with respect to prefixes is equivalent to coloring n points statically with respect to (all) intervals. Thus, at least $1 + \lfloor \lg n \rfloor$ colors are needed. □

We propose the following algorithm for coloring prefixes: The algorithm colors differently

(a) points that appear before all previously inserted points,

(b) points that appear after at least one previously inserted point.

The first group of points contains points for which $\sigma_{(i)} = 0$, and the second group points for which $\sigma_{(i)} > 0$. Therefore, it is possible to distinguish between the two groups even in the relative positions model. Points in the first group are colored according to the static coloring for intervals: ...41213121. Points in the second group are all colored with the same color, which is different from the colors used in the first group. For example, input $\sigma = 010120020$ is colored as $131\star2\star\star1\star$, where '$\star$' is the color used for points in the second group.

From the above, it is not hard to derive the following result:

PROPOSITION 8. *For $n \geq 2$, there is an algorithm that conflict-free colors with respect to prefixes any insertion sequence σ (in the relative positions model) with at most $2 + \lfloor \lg(n-1) \rfloor$ colors.*

Finally, we use the upper bound for prefixes (and suffixes) to prove an upper bound for rays. We claim that for dynamically coloring with respect to rays, one more color than the prefix (or suffix) case suffices. The idea is to use a unique color for the first point p inserted, and then color independently points to the left of p from points to the right of p: color whatever is inserted to the left of p with respect to prefixes and whatever is inserted to the right of p with respect to suffixes. From the above, it is not hard to prove:

PROPOSITION 9. *For $n \geq 3$, there is an algorithm that conflict-free colors with respect to rays any insertion sequence σ (in the relative positions model) with at most $3 + \lfloor \lg(n-2) \rfloor$ colors.*

model	lower bound	upper bound
all dynamic	$1 + \lfloor \lg n \rfloor$	$3 + \lfloor \lg(n-2) \rfloor$
static	3	3

Table 4: Number of colors used in deterministic algorithms for rays ($n \geq 3$)

The above analysis gives a separation between static and dynamic models for coloring with respect to rays: The number of colors used is a logarithmic factor apart. All the results are shown in table 4. This is in contrast with the all-intervals case in which the separation result between static and dynamic offline model is weaker, just a constant factor apart, $1 + \lg n$ and $1 + \log_{3/2} n$ colors used, respectively.

7. DISCUSSION AND OPEN PROBLEMS

We introduced a hierarchy of models for conflict-free coloring ranging from a completely static model (weakest adversary model) to a fully online model (strongest adversary model). We concentrated on conflict-free coloring with respect to intervals. For this special case, we proposed deterministic algorithms for some of the models and gave upper bounds on their worst-case performance. We also provided lower bounds on the number of colors used in some models.

There are still gaps between lower and upper bounds. For example, in the dynamic offline model the lower bound is $1 + 2\log_3 n \approx 1.26 \lg n$, whereas the upper bound is $1 + \log_{3/2} n \approx 1.71 \lg n$, a constant factor apart. The situation is similar in the absolute positions model where the upper bound is approximately $1.89 \lg n$. The most important open problem is narrowing the gap between lower and upper bound in the relative positions model: $\Omega(\log n)$, and $O(\log^2 n)$, respectively, which are a logarithmic factor apart.

So far in the literature, only the static and the fully online (relative positions) models had been considered. If there is a gap on the number of colors used between these two extreme models, the hierarchy can help pin-point exactly where the 'jump' occurs, and thus give a better understanding of the problem. In the case of all-intervals, static uses $O(\log n)$ and the best known online deterministic algorithm $O(\log^2 n)$ colors, but this logarithmic factor 'jump' is not a result of the online model, because it occurs just between the absolute positions model and the fully online (relative positions) model. However, it can not always be the case that the jump occurs between these two models: As we have seen in the rays case, a logarithmic factor jump occurs between the static and the dynamic offline model.

Another open problem is removing the following possible shortcoming from the absolute position $O(\log n)$ algorithms: Both 2CU and the triples algorithms might use too many colors for the first requests. For example, in the triples algorithm, for final size of input $n = 3^k$, the adversary can request the first k points in such a way such that the algorithm uses k different colors.

Finally, the hierarchy of models is not constrained to problems for points on the real line. It can be used in conflict-free coloring for hypergraphs, in general. A possible use of the hierarchy would be to understand better conflict-free coloring problems in the plane.

Acknowledgements

We would like to thank János Pach and David Peleg for helpful discussions concerning the problems studied in this paper. We would also like to thank the reviewers for their comments.

8. REFERENCES

[1] Noga Alon and Shakhar Smorodinsky. Conflict-free colorings of shallow discs. To appear in *22nd Annual*

ACM Symposium on Computational Geometry, 2006.

[2] Amotz Bar-Noy, Panagiotis Cheilaris, and Shakhar Smorodinsky. Online conflict-free coloring for hypergraphs. Manuscript, 2006.

[3] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[4] Ke Chen. On how to play a coloring game against color-blind adversaries. To appear in *22nd Annual ACM Symposium on Computational Geometry*, 2006.

[5] Khaled Elbassioni and Nabil H. Mustafa. Conflict-free colorings for rectangle ranges. To appear in *23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006.

[6] 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*, 33:94–136, 2003. Also in *Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[7] Amos Fiat, Meital Levy, Jiří Matoušek, Elchanan Mossel, János Pach, Micha Sharir, Shakhar Smorodinsky, Uli Wagner, and Emo Welzl. Online conflict-free coloring for intervals. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 545–554, 2005.

[8] Sariel Har-Peled and Shakhar Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete and Computational Geometry*, 34:47–70, 2005.

[9] Haim Kaplan and Micha Sharir. Online CF coloring for halfplanes, congruent disks, and axis-parallel rectangles. Manuscript, 2004.

[10] János Pach and Géza Tóth. Conflict free colorings. In *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, pages 665–671. Springer Verlag, 2003.

[11] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 687–703, 1989.

[12] Shakhar Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, School of Computer Science, Tel-Aviv University, 2003.

[13] Shakhar Smorodinsky. On the chromatic number of some geometric hypergraphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.