Kinetic Conflict-Free Coloring*

Mark de Berg† Tim Leijssen† Marcel Roeloffzen‡

Abstract

A conflict-free coloring, or CF-coloring for short, of a set P of points in the plane with respect to disks is a coloring of the points of P with the following property: for any disk D containing at least one point of P there is a point $p \in P \cap D$ so that no other point $q \in P \cap D$ has the same color as p. In this paper we study the problem of maintaining such a CF-coloring when the points in P move. We present two methods for this and evaluate the maximum number of colors used as well as the number of recolorings, both in theory and experimentally.

1 Introduction

Wireless communication is commonplace in many applications, varying from mobile consumer devices like cell-phones to specialized networks for autonomous robots. For clear wireless communication it is important to avoid interference. In general interference-free communication is achieved by using different frequencies for different communication channels. However, the number of available frequency ranges is limited so it is impossible to assign a unique frequency to every communication channel. A more clever frequency-assignment scheme is therefore needed. Finding a proper assignment of frequencies is often modeled as the problem of computing a so-called *conflict-free coloring*, or *CF-coloring* for short, where different colors represent different frequencies.

In this paper we look at the problem of finding a CF-coloring with respect to disks for a set P of points in the plane. That is, we want to assign colors to the points of P so that every disk D containing at least one point of P contains a unique color. More formally, we want to assign a color $color(p)$ to each point $p \in P$ so that the following property holds: for every disk D containing at least one point of P, there is a point $p \in P \cap D$ so that for any point $q \in P \cap D \backslash \{p\}$ we have $color(p) \neq color(q)$. It is convenient to identify the i-th color with the integer i. The goal is then to minimize the maximum color used in the CF-coloring.

*MdB is supported by the Netherlands Organisation for Scientific Research (NWO) and MR is supported by the SMART project at Tohoku University.

†Department of Computer Science, TU Eindhoven, the Netherlands. *mdberg@win.tue.nl*

‡Graduate School of Information Sciences, Tohoku University, Japan. *marcel@dais.is.tohoku.ac.jp*

The problem was introduced by Even *et al.* [5] who provide a framework for finding CF-colorings with respect to disks and several other types of regions. Their algorithm works by finding independent sets on a sequence of Delaunay triangulations of subsets of P and uses $O(\log n)$ colors, which is asymptotically optimal [5, 7]. Several variants of the problem have also been studied. For example, Har-Peled *et al.* [6] provide a probabilistic algorithm that works for simple regions with low union-complexity, as well as several algorithms for special cases such as axis-aligned rectangles. They also study k-CF-coloring, which is a relaxation of a regular CF-coloring. In a k-CF-coloring each region need not contain a unique color, but instead a color that occurs between 1 and k times in that region. There are also several results for online CF-coloring, where points are added incrementally and must be assigned a color when added without knowledge of points to come [2, 4]. Additional background can be found in a recent survey by Smorodinsky [8].

All these papers assume the transmitters are stationary and frequencies are assigned exactly once. This may however not always be the case. For mobile RFID scanners or networks of autonomous robots, for example, the frequency assignment may need to change as the transmitters move. Unfortunately changing the frequency of a transmitter is often undesirable as it may interrupt current communications with that transmitter. This leads us to study the problem maintaining a conflict-free coloring in the so called *Kinetic Data Structures (KDS)* framework.

Following the KDS framework as introduced by Basch *et al.* [3] we assume that we know the trajectories of the points (at least in the short term). The goal is now to maintain a CF-coloring of the points as they move along these trajectories. In frequency assignment, changing the frequency of a transmitter is usually expensive. Therefore, we focus on keeping the number of recolorings low while using $O(\log n)$ colors. We provide two algorithms for this, both based on the algorithm by Even *et al.* [5]. For both algorithms we provide a theoretical analysis and an experimental analysis of the maximum number of colors used and of the number of recolorings.

2 The algorithms

First we describe the algorithm by Even *et al.* [5] for static points in more detail, as our kinetic algo-

rithms are based on this. The algorithm is recursive and starts by computing the Delaunay triangulation $\mathcal{DT}(P_1)$ of the complete point set $P_1 := P$. The next step is to find a large independent set $I_1 \subset P_1$ in $\mathcal{DT}(P_1)$. All points in I_1 receive color 1. On the set $P_2 := P_1 \backslash I_1$ a new Delaunay triangulation and independent set I_2 are computed, then the points in I_2 receive color 2, and the algorithm proceeds with $P_3 := P_2 \backslash I_2$. This process is repeated until all points are colored. If we compute the independent set in each $\mathcal{DT}(P_i)$ in a greedy manner, by considering the points in order of increasing degree, we can guarantee that $|I_i| \geqslant |P_i|/6$, which implies that the algorithm finishes after $O(\log n)$ rounds. The proof that this procedure produces a conflict-free coloring was given—in a more general setting—by Even *et al.* [5].

Next we adapt the static algorithm to a kinetic setting where points move along known constant-complexity curves. From the above it is clear that as long as none of the Delaunay triangulations change, then the coloring remains conflict-free. However, as the points move the Delaunay triangulation can change, namely when four points that form a quadrangle in the Delaunay triangulation become co-circular. When this happens a diagonal of the quadrangle flips, which may invalidate the independent set. Under known motions these co-circularities are not difficult to detect and kinetic data structures exist that maintain a Delaunay triangulation and support insertion and deletion of vertices [1]. Therefore we focus on repairing the independent sets when such a flip occurs. We propose two different methods, a greedy method and a lazy method.

In the following we describe what the two methods do when a flip occurs between vertices within a quadrangle $Q := Q(p, q, r, s)$, where the edge pr is replaced by qs. The quadrangle Q may exist in the Delaunay triangulation of multiple consecutive levels, but only in the last level i where it occurs can one or more of the points p, q, r, s be in the independent set. This follows from the fact that a point in the independent set of level i cannot occur in any level $j > i$. We denote this level by $lev(Q)$ and use k to denote the total number of colors used before recovering.

Lazy updates. In the lazy approach we check if the new edge connects two points in the independent set, thus making the set no longer independent. If so, we recolor one of the two conflicting vertices to a new color. As a result the independent set of each layer will slowly become smaller compared to the total number of points in that layer. When the independent set becomes too small, we completely recompute the structure from that layer onward.

In more detail, let $Q := Q(p, q, r, s)$ denote the quadrangle within which the flip occurred, where edge pr is replaced by qs. Let $i := lev(Q)$. When the edge pr flips to edge qs, we check if q and s are both in the independent set I_i. If so, we remove one of the two, say q, from I_i, give it a new color $k+1$ and add it to all sets P_j for $i < j \leqslant k+1$. Note that any two points that were independent in $\mathcal{DT}(P_j)$ are still independent in $\mathcal{DT}(P_j \cup \{q\})$. Since we reduced I_i in size we check if it has become too small. More precisely, we check if $|I_i| < |P_i|/12$, and if so we recolor P_i entirely by running the static algorithm on P_i. We call this a *reset* at level i. If we do not do a reset at level i, we check if $|I_j| < |P_j|/12$ for some $j > i$. (This is needed since we added q to all sets P_j for $i < j \leqslant k+1$.) If this is the case for some levels j, we do a reset at the smallest such level j.

Lemma 1 *Maintaining a CF-coloring with lazy updates ensures that we use at most $O(\log n)$ colors at any time.*

Proof. Each update guarantees that the sets I_i are independent. Following the proof of Even *et al.* [5] this is enough to enough to show the coloring remain conflict-free. Since we guarantee that each independent set I_i contains at least $|P_i|/12$ points at any time, $O(\log n)$ colors are used. □

Our goal is to guarantee a CF-coloring using as few recolorings as possible. Therefore, we now study the number of recolorings that this method makes. It is easy to see that a single flip in a Delaunay triangulation may cause $\Theta(n)$ recolorings: if a flip triggers a reset of the first level, then all points may be recolored. However, this does not happen often since our greedy independent-set computation guarantees that initially $|I_i| \geqslant |P_i|/6$, whereas we won't reset until $|I_i| < |P_i|/12$. Most events—that is, flips in the Delaunay triangulation—do not cause many recolorings and we can in fact prove that amortized only $O(\log n)$ recolorings happen per event.

Lemma 2 *Each event triggers $O(\log n)$ recolorings amortized.*

Proof. We prove this using an accounting scheme, where each point p has a wallet $W_i(p)$ with a certain amount of money, for every level i where it occurs. To recolor a point it must pay €1. Thus, when doing a reset at level i, all points in P_i must pay €1. Next we show that in each event we have to spend only $O(\log n)$ euro to guarantee we can pay for all recolorings. We keep as an invariant that any point p in every level i where it occurs has at least

$$1 - \frac{|I_i| - |P_i|/12}{|P_i|/12} = 2 - \frac{12|I_i|}{|P_i|} \text{ euros}$$

in its wallet $W_i(p)$. Note that since $|I_i| \geqslant |P_i|/6$ after a reset at level i, the points need not have any money

immediately after a reset. Moreover, a reset occurs when $|I_i| < |P_i|/12$, so then each point has at least €1 and can pay for its recoloring.

Next we show we need to spend no more than $O(\log n)$ euro per event to maintain the invariant. Consider an event that causes point q to be removed from the independent set I_i and added to a new level P_{k+1}. The wallets for points in levels $j < i$ do not change as the set P_j and I_j do not change. In level i the size of P_i remains the same, but I_i becomes one smaller. Let I_i and P_i denote the independent set and complete set of points in level i before the event and I_i' and P_i' the same sets after the event. Before the event we know that each point $p \in P_i$ has at least $2 - 12|I_i|/|P_i|$ euro. After the event it should have at least $2 - 12|I_i'|/|P_i'| = 2 - 12(|I_i| - 1)/|P_i|$ euro. To ensure this we must give each point at most $12/|P_i|$ euro, so at most €12 in total for all points in level i.

The money to be payed to levels $j > i$ is calculated as follows. Before the event each point in level j has at least $2-12|I_j|/|P_j|$ euro and after the event it must have at least $2 - 12|I_j|/(|P_j| + 1)$ euro. This means we should pay each point in P_j (not including q)

$$\frac{12|I_j|}{|P_j|} - \frac{12|I_j|}{|P_j|+1} = \frac{12|I_j|}{|P_j|(|P_j|+1)} \leqslant \frac{12}{|P_j|+1} \text{ euro.}$$

Since we have to pay this to $|P_j|$ points, this costs us no more than €12 per level. Lastly we have not yet filled the wallet for the point q in all the levels it has been added to. However, in each level it requires at most €1. In total we spend no more than €13 per level, so $O(\log n)$ euro in total. □

Greedy updates. In the lazy approach we do $O(\log n)$ recolorings amortized per event, but in some updates we may have to recolor all points. We can try to avoid this worst-case behavior by keeping the independent sets large. To this end, at each event we not only remove a point from an independent set, but we also try to add new points. We maintain as an invariant that each independent set is a maximal independent set consisting only of points with degree less than 12. This produces an independent set I_i with $|I_i| \geqslant |P_i|/24$, as at least half the vertices have degree less than 12 and choosing one of these eliminates no more than 12 from becoming part of the independent set. Next we discuss the updates necessary to maintain this invariant.

Consider an event that is a flip within quadrangle $Q := Q(p, q, r, s)$, where the edge pr is replaced by edge qs. We denote by $lev^*(Q)$ the first level (that is, with smallest index i) where this flip occurs. When processing the event we start at $lev^*(Q)$ and then proceed to later levels. In each level we have the following situation. We have a current set P_i with Delaunay triangulation $\mathcal{DT}(P_i)$ and two (possibly empty) sets P_i^+ and P_i^-. The set P_i^+ contains points that are "pushed down" from the parent level because they are no longer in the independent set at that level; the set P_i^- contains points that are "pulled up" from the parent level because they have just been added to the independent set at the parent level. We then update the Delaunay triangulation by constructing $\mathcal{DT}((P_i \cup P_i^+) \setminus P_i^-)$, update the independent set I_i, and construct the sets P_{i+1}^+ and P_{i+1}^- by looking at the differences between old and new independent set I_i.

In more detail we proceed as follows. We start by computing $\mathcal{DT}((P_i \cup P_i^+) \setminus P_i^-)$. Note that for $i = lev^*(Q)$ we have $P_i^+ = P_i^- = \emptyset$, but the Delaunay triangulation still changes because of the flip in Q. Let P_i^* denote the set of points whose neighbor set has changed, including the set P_i^+ of new points. Now for the points in $P_i^* \cap I_i$ we test if they still have degree below 12, otherwise we remove them from I_i. Then we check if any two vertices remaining in the independent set are connected, and we remove one of them if needed. We repeat this until no more connected pairs exist, and we have obtained an independent set for the new Delaunay triangulation. Next we make the independent set maximal by greedily adding points with degree less than 12 that are not connected to any points of the independent set. The resulting set is the new independent set I_i. As already mentioned, we then construct the sets P_{i+1}^+ and P_{i+1}^- by looking at the differences between old and new independent set I_i. Note that the sets P_i^+ and P_i^- (and, hence, the set P_i^* of "affected points") may grow as we go down the levels. Unfortunately it seems hard to prevent this, which makes it difficult to bound the worst-case number of recolorings. Fortunately our experiments (see below) show that in practice the avalanche effect is limited and not too many recolorings occur per event.

3 Experimental results

We showed some basic theoretical bounds for the number of recolorings. However, especially for the greedy approach it seems unlikely that in practice we need many recolorings per step. We therefore implemented both algorithms in order to count the number of recolorings. For our input we use sets of points that move along straight lines within a bounding box. To avoid points going outside the bounding box, they bounce back when hitting the edge of bounding box. We ran our implementation of the two methods for sets of moving points ranging from 20 to 2000 points and measure the number of recolorings per step as well as the total number of colors used. For 100 points the results can be found in Fig. 1. The results with other amounts of points are summarized in Table 1 and are based on a run of 10,000 events—that is, flips in any of the Delaunay graphs.

The graphs of Fig. 1 clearly show the difference between the two methods. The lazy method has a very

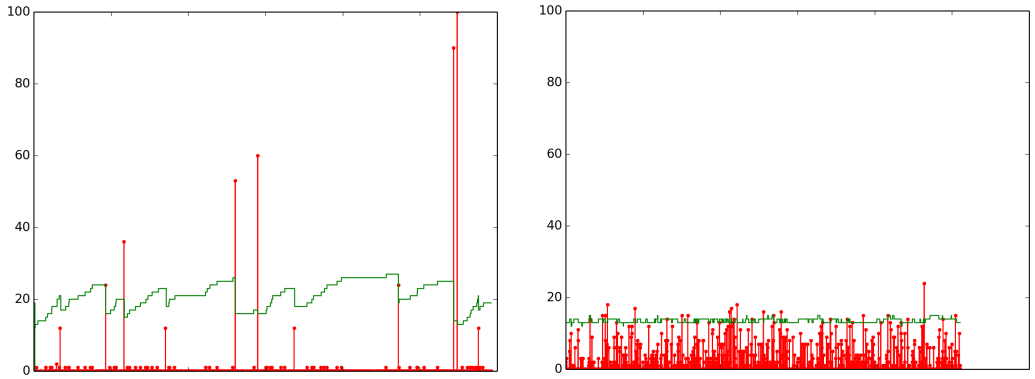

Figure 1: Number of recolorings indicated in red and total number of colors used in green for the lazy (left) and greedy (right) method. Both are measured from an instance with 100 moving points over 2500 events.

	Lazy Updates						Greedy Updates					
	#Colors				#Recolorings		#Colors				#Recolorings	
n	avg	max	$\frac{\text{avg}}{\log n}$	$\frac{\text{max}}{\log n}$	total	max	avg	max	$\frac{\text{avg}}{\log n}$	$\frac{\text{max}}{\log n}$	total	max
20	12.4	15	2.9	3.5	1232	20	8.2	10	1.9	2.3	6414	12
50	17.2	22	3.0	3.8	1569	50	11.2	14	2.0	2.5	8036	19
100	20.9	28	3.1	4.2	2017	100	13.5	16	2.0	2.4	9834	24
200	24.6	31	3.2	4.1	2275	200	15.9	18	2.1	2.4	12034	27
500	28.3	35	3.2	3.9	2879	337	19.1	21	2.1	2.3	15248	34
1000	31.3	38	3.1	3.8	4064	444	21.3	23	2.1	2.3	18406	62
2000	34.0	42	3.1	3.8	5659	696	23.5	25	2.1	2.3	22164	60

Table 1: Number of colors used and recolorings for n moving points using lazy and greedy updates.

low number of recolorings for most events, but several events with a many recolorings that correspond to resets at top levels of our data structure. This step-wise behavior is also found in the total number of colors used, which slowly goes up until a reset of one of the top levels is done. The greedy approach on the other hand does not have this spiky behavior, and instead many events cause between 5 and 20 recolorings, but rarely more than that. It also has the effect that the total number of colors appears very stable.

In Table 1 we see that in both cases the number of colors used is proportional to $\log n$. What is interesting though is that although the greedy method uses less colors, both on average and maximum, it requires many more recolorings on average. This shows that the neither of the two methods is strictly better than the other. If one can afford to do many recolorings every now and then, and one does not care about using a few more colors, then the lazy method is better as it will do fewer recolorings. On the other hand, if the goal is to keep the total number of colors very small or to avoid many recolorings at a single event, then the greedy method appears better.

References

[1] G. Albers, L.J. Guibas, J.S.B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *Int. J. Comput. Geometry Appl.* 8(3):365–380, 1998.

[2] A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Online conflict-free colorings for hypergraphs. *Journal Combinatorics, Probability and Computing* 19(4):493–526, 2010.

[3] J. Basch, L.J. Guibasand, J. Hershberger. Datastructures for mobile data. In *Proc. 8th ACM-SIAM Symp. Disc. Alg.*, pages 747–756, 1997.

[4] K. Chen, A. Fiat, H. Kaplan, M. Levy, J. Matoušek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, and U. Wagner. Online conflict-free coloring for intervals. *SIAM Journal on Computing* 36(5):1342–1359, 2006.

[5] G. Even, Z. Lotker, D. Ron, S. 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.

[6] S. Har-Peled and S. Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry* 34(1):47–70, 2005.

[7] J. Pach and G. Toth. Conflict-free colorings. In *Discrete and Computational Geometry - The Goodman-Pollack Festschrift* (S. Basu et al., eds.), Springer Verlag, Berlin, pages 665–671, 2003.

[8] S. Smorodinsky. Conflict-Free Coloring and its Applications. In *Geometry - Intuitive, Discrete, and Convex* (I. Bárány et al., eds.) Springer Verlag, Berlin, 2014.