
Proceedings Paper:
Bosek, B., Disser, Y., Feldmann, A.E. et al. (2020) Recoloring interval graphs with limited recourse budget. In: Albers, S., (ed.) Leibniz International Proceedings in Informatics, LIPIcs. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020), 22-24 Jun 2020, Tórshavn, Faroe Islands. Schloss Dagstuhl--Leibniz-Zentrum für Informatik, 17:1-17:23. ISBN: 9783959771504. ISSN: 1868-8969.

https://doi.org/10.4230/LIPIcs.SWAT.2020.17

Recoloring Interval Graphs with Limited Recourse Budget

Bartłomiej Bosek
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University in Kraków, Poland
bosek@tcs.uj.edu.pl

Yann Disser
Department of Mathematics, TU Darmstadt, Germany
disser@mathematik.tu-darmstadt.de

Andreas Emil Feldmann
Department of Applied Mathematics, Charles University in Prague, Czech Republic
https://sites.google.com/site/aefeldmann
Andreas.Feldmann@mff.cuni.cz

Jakub Pawlewicz
University of Warsaw, Poland
pan@mimuw.edu.pl

Anna Zych-Pawlewicz
University of Warsaw, Poland
anka@mimuw.edu.pl

Abstract

We consider the problem of coloring an interval graph dynamically. Intervals arrive one after the other and have to be colored immediately such that no two intervals of the same color overlap. In each step only a limited number of intervals may be recolored to maintain a proper coloring (thus interpolating between the well-studied online and offline settings). The number of allowed recolorings per step is the so-called *recourse budget*. Our main aim is to prove both upper and lower bounds on the required recourse budget for interval graphs, given a bound on the allowed number of colors.

For general interval graphs with n vertices and chromatic number k it is known that some recoloring is needed even if we have $2k$ colors available. We give an algorithm that maintains a $2k$-coloring with an amortized recourse budget of $\mathcal{O}(\log n)$. For maintaining a k-coloring with $k \leq n$, we give an amortized upper bound of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, and a lower bound of $\Omega(k)$ for $k \in \mathcal{O}(\sqrt{n})$, which can be as large as $\Omega(\sqrt{n})$.

For unit interval graphs it is known that some recoloring is needed even if we have $k+1$ colors available. We give an algorithm that maintains a $(k+1)$-coloring with at most $\mathcal{O}(k^2)$ recolorings per step in the worst case. We also give a lower bound of $\Omega(\log n)$ on the amortized recourse budget needed to maintain a k-coloring.

Additionally, for general interval graphs we show that if one does not insist on maintaining an explicit coloring, one can have a k-coloring algorithm which does not incur a factor of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ in the running time. For this we provide a data structure, which allows for adding intervals in $\mathcal{O}(k^2 \log^3 n)$ amortized time per update and querying for the color of a particular interval in $\mathcal{O}(\log n)$ time. Between any two updates, the data structure answers consistently with some optimal coloring. The data structure maintains the coloring implicitly, so the notion of recourse budget does not apply to it.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Online algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases Colouring, Dynamic Algorithms, Recourse Budget, Interval Graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.17

Funding

Bartłomiej Bosek: National Science Centre of Poland, project number 2017/26/D/ST6/00264.

Yann Disser: 'Excellence Initiative' of the German Federal and State Governments, and Graduate School CE at TU Darmstadt.

Andreas Emil Feldmann: Czech Science Foundation GAČR (grant #17-10090Y), and Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

Anna Zych-Pawlewicz: National Science Centre of Poland, project number 2017/26/D/ST6/00264.

1 Introduction

Graph coloring is one of the most prominent disciplines within graph theory, with plenty of variants, applications, and deep connections to theoretical computer science. A *proper k-coloring* of a graph, for a positive integer k, is an assignment of colors in $\{1, \dots, k\}$ to the vertices of the graph in such a way that no two adjacent vertices share a color. The chromatic number of the graph is the smallest integer k for which a proper k-coloring exists. In general, it is NP-hard [17, 36] to approximate the chromatic number of an n-vertex graph to within a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$. The literature offers many results for restricted graph classes.

In this paper, we consider the class of interval graphs, for which a linear time greedy algorithm achieves the optimum coloring [26]. Our main interest is in a dynamic setting, where intervals arrive one at a time, and one needs to maintain the coloring after each interval addition. We mainly study how many vertex recolorings are needed to maintain a reasonable coloring. The number of changes one needs to introduce to the maintained solution (in our case vertex recolorings) upon an update is referred to as *recourse bound* or *recourse budget* in the literature. A recourse budget of zero coincides with the online setting, where the algorithm's decisions are irrevocable. The online model is natural for many problems [14, 24, 27, 29] and has been widely studied, very often revealing pessimistic lower bounds. It is natural to ask if the situation improves if one allows a limited recourse budget. This model has been successfully applied to a variety of problems, including spanning tree and Steiner tree variants, bipartite matchings, and coloring [2, 3, 9, 10, 11, 16, 23, 21]. The proposed algorithms could often be efficiently implemented [3, 9, 21].

Formally, we are interested in the following problem. We get a sequence of half-open intervals $\{[a_i, b_i)\}_{i=1}^{n}$, which defines a sequence of instances $\mathcal{I}_j = \{[a_i, b_i)\}_{i=1}^{j}$, where $\mathcal{I}_j$ differs from $\mathcal{I}_{j-1}$ by one interval. The instances may be interpreted as graphs, where the nodes are intervals and the edges connect intersecting intervals. The intervals arrive one at a time. After the j-th interval is revealed, the algorithm needs to compute a proper coloring C_j for the intersection graph of $\mathcal{I}_j$. We wish to minimize the recourse budget, which is the number of vertices with different colors in C_j and C_{j-1}. We also consider the special case of *unit* interval graphs, where each interval is of the form $b_i = a_i + 1$. For the sake of simplicity we assume that every instance $\mathcal{I}_j$ is k-colorable and k is known a priori, but it is not difficult to get rid of this assumption. Our results are summarized in Table 1 together with some known results from the literature for comparison. Unless stated otherwise, all the bounds in the table are amortized, i.e., they bound the average recourse budget per insertion.

For general interval graphs our first result shows that if we allow $\mathcal{O}(\log n)$ recolorings per interval insertion, we can improve the ratio of 3 of the online algorithm by Kierstead and Trotter [20] to 2. Since the ratio of 3 is best possible in the online setting, our result shows that only a modest number of recolorings are needed to obtain an improvement. If we allow a higher number of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ recolorings per update, we can even maintain an optimal solution. A trivial algorithm that recolors all intervals in each step has a recourse

Table 1 Our results for interval graphs (top) and unit interval graphs (bottom). All runtimes are amortized, if not otherwise stated.

		recourse budget	
	colors	upper bound	lower bound
general	$3k-2$	0 [20]	0
	$2k$	$\mathcal{O}(\log n)$ (Thm 5)	>0 [20]
	k	$\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ (Thm 10)	$\Omega(k)$ (Cor 21) for $k \in \mathcal{O}(\sqrt{n})$ $\Omega(\sqrt{n})$ (Cor 20) for $k \in \Theta(\sqrt{n})$ $\Omega(\log n)$ (Thm 1) for $k = 2$
unit interval	$2k-1$	0 [5]	0
	$k+1$	$\mathcal{O}(k^2)$ worst case (Thm 2)	>0 [5]
	k	$\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ (Thm 10)	$\Omega(\log n)$ (Thm 1)

budget of n, resulting in the bound $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ of Table 1. Note that this bound is non-trivial (i.e., smaller than n) for $k \in \mathcal{O}(\frac{\log n}{\log \log n})$. We complement these results with a lower bound for the budget of $\Omega(k)$, which can be as high as $\Omega(\sqrt{n})$ if k grows with n. We obtain another lower bound of $\Omega(\log n)$ for $k = 2$. The latter bound is even valid for unit interval graphs, for which we also show that if we allow a budget of $\mathcal{O}(k^2)$ recolorings (i.e., independent of n), we can maintain a solution using just one extra color compared to the optimum. Due to our lower bound of $\Omega(\log n)$ for maintaining an optimal coloring, it is clear that an extra color is necessary if we want to keep the budget constant for a constant k.

It is straightforward to see that our algorithms, except for the exact algorithm for general interval graphs that uses an amortized recourse budget of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, can be implemented efficiently. However, we can improve the exact algorithm significantly if we do not insist on maintaining an explicit coloring, i.e., if we do not require that the color of an interval can be retrieved in constant time. In Section 5 we provide a data structure, which allows for adding intervals in $\mathcal{O}(k^2 \log^3 n)$ amortized time per update and querying for the color of a particular interval in $\mathcal{O}(\log n)$ time. Between two updates the data structure answers queries consistently with some optimal coloring. The data structure maintains the coloring implicitly, so the notion of recourse budget does not apply to it.

1.1 Related work

Due to the inapproximability of the graph coloring problem, the positive results for dynamic coloring of general graphs are mostly of heuristic and experimental nature [25, 28, 30, 32, 35]. From the theoretical perspective, just recently there have been a few results concerning the recourse budget for coloring general graphs [2, 33] and dynamic general graph coloring with $\Delta + 1$ colors [4], where Δ is the maximum degree in the graph.

Barba et al. [2] devise two complementary algorithms for the regime of adding and removing edges. For any $d > 0$, the first (resp. second) algorithm maintains a $k(d+1)$-coloring (resp. $k(d+1)n^{1/d}$-coloring) of a k-colorable graph and recolors at most $(d+1)n^{1/d}$ (resp. d) vertices per update, where updates include edge and vertex additions and removals. The authors also show that the first trade-off is essentially tight, and the bad example is a tree. So if one insists on a constant approximation ratio, one must incur polynomial recourse budget for every class of graphs that contains trees. The symmetry between these trade-offs may make it tempting to believe that the second trade-off is also tight. However, Solomon

and Wein [33] show, that in the regime of adding and removing edges, there is a deterministic algorithm for maintaining an $\mathcal{O}(\frac{k}{d}\log^3 n)$-coloring with $\mathcal{O}(d)$ recolorings per update step for any $d \in \mathcal{O}(\log n)$. They also show that a randomized algorithm performs slightly better. Solomon and Wein additionally consider bounded arboricity graphs, for which, using their result on the recourse budget, they provide an efficient dynamic algorithm maintaining an $\mathcal{O}(\alpha \log^2 n)$-coloring with polyloglog amortized time per update. Bhattacharya et al. [4] studied the problem of efficient dynamic coloring when the maximum degree of the dynamic graph remains bounded by Δ at all times. They present a randomized (resp. deterministic) algorithm for maintaining a $(\Delta+1)$-coloring (resp. $\Delta(1+o(1))$-coloring) with amortized $\mathcal{O}(\log \Delta)$ (resp. polylog(Δ)) update time.

To the best of our knowledge, no dynamic algorithms for the class of interval graphs have been proposed in the literature. Our motivation for studying this class of graphs in the incremental regime stems from the rich literature on the problem of online poset coloring. Schmerl asked whether an effective online chain partitioning algorithm exists, and this was answered in the affirmative by Kierstead in [18]. His algorithm uses at most $(5^w-1)/4$ chains on posets of width w. Szemerédi proved a quadratic lower bound of $\binom{w+1}{2}$ (see [5, 19] for a proof). In [7], Bosek and Krawczyk provide an online algorithm that partitions posets of width w into at most $w^{13\log_2 w}$ chains. This yields the first subexponential upper bound for the online chain partitioning problem. In [6] Bosek et al. improve this to $w^{6.5\log_2 w+7}$ with a shorter proof. Very recently, in [8] Bosek and Krawczyk present an online algorithm that partitions posets of width w into $w^{O(\log\log w)}$ chains. At this point, the problem of whether there is an online algorithm using polynomially many chains is still open.

The problem of online interval poset chain coloring is equivalent to the problem we are studying with the recoloring budget limited to zero. It has been extensively studied in many different variants [1, 5, 12, 20]. A well-known theorem of Kierstead and Trotter [20], translated to our setting, states that a $(3k-2)$-coloring of k-colorable graph can be maintained online and this is the best we can do if we do not allow recolorings. A folklore result [5] states that for unit interval graphs a $(2k-1)$-coloring of k-colorable graph can be maintained online and this is also tight. It is natural to wonder how many recolorings we need when the approximation ratio is going from 3 down to 1 for general interval graphs, or from 2 down to 1 for unit interval graphs. This is the main question we aim to answer in this paper. Nevertheless, it is not hard to show that our $(k+1)$-coloring algorithm for unit interval graphs can be extended to a fully dynamic setting (allowing also interval removals). In particular, this gives one more non-trivial class of graphs where the lower bound of Barba et al. [2] does not apply.

2 Unit intervals

In this section we focus on the class of unit interval graphs. This class is equivalent with proper interval graphs, i.e., interval graphs where no interval is contained in another interval [31]. We show a lower-bound of $\Omega(\log n)$ for the recourse budget for maintaining an optimal coloring.

▶ **Theorem 1.** *Maintaining an optimum coloring of a 2-colorable unit interval graph requires an amortized recourse budget of* $\Omega(\log n)$.

Proof. We describe a 2-colorable interval graph that appears online in the form of recursively constructed gadgets. We start with with the gadget G_0 consisting of the two intersecting intervals $[0,1)$ and $[0.5,1.5)$ that can be 2-colored without recoloring. Obviously, G_0 admits a unique 2-coloring (up to renaming colors).

Now, for $i \in \{1, 2, \dots\}$, assume we have a recursive construction G_{i-1} that admits a unique 2-coloring (up to renaming colors), and that all intervals in this coloring fall into $[a, b)$ in one color and into $[a+0.5, b+0.5)$ in the other color. This means that, regarding 2-colorings, G_{i-1} behaves macroscopically exactly like two intervals of the form $[a, b), [a + 0.5, b + 0.5)$. To obtain G_i, we first introduce, one after the other, two G_{i-1} gadgets shifted so that they behave exactly like the pairs of intervals $[a, b), [a + 0.5, b + 0.5)$ and $[b + 1, c), [b + 1.5, c + 0.5)$, respectively. See Fig. 1 along with the following.

Up to renaming colors, there are two ways of coloring the gadgets. If $[a, b)$ and $[b + 1, c)$ receive the same color, we introduce the additional intervals $[b + 0.25, b + 1.25)$ and $[c, c + 1)$. Otherwise, $[a, b)$ and $[b + 1.5, c + 0.5)$ receive the same color, and we introduce the additional intervals $[b, b + 1)$ and $[b + 0.5, b + 1.5)$. In both cases, there is no way of consistently coloring the new intervals without recoloring one of the two gadgets. Since the gadgets admit a unique 2-coloring up to renaming colors, we need to completely recolor one of them by changing the color of all of its intervals. Afterwards, G_i admits a unique 2-coloring (up to renaming colors), and all intervals fall into $[a, c + 0.5)$ in one color and $[a + 0.5, c + 1)$ in the other color. We can therefore proceed with the recursive construction.

The number of intervals n_i of G_i is given by $n_0 = 2$ and $n_i = 2n_{i-1} + 2$ for $i \in \{1, 2, \dots\}$, which yields $n_i = 2^{i+1} + \sum_{j=1}^{i} 2^j = 2^{i+2} - 2$. The number of recolorings required during the recursive construction of G_i is given by $r_0 = 0$ and $r_i = 2r_{i-1} + n_{i-1}$ for $i \in \{1, 2, \dots\}$, which yields $r_i = \sum_{j=1}^{i} 2^{i-j} n_{j-1} = \sum_{j=1}^{i} 2^{i-j}(2^{j+1} - 2) = i \cdot 2^{i+1} - 2\sum_{j=0}^{i-1} 2^j = i \cdot 2^{i+1} - 2^i + 1$. This means that, asymptotically, we have $n_i = \Theta(2^i)$ and the amortized number of required recolorings is $r_i/n_i = \Theta(i) = \Theta(\log(n_i))$. ◀

We now prove an upper bound of $\mathcal{O}(k^2)$ for the worst-case recourse budget, which holds if the algorithm can use one extra color. This is in contrast with the lower bound of Theorem 1, which is $\Omega(\log n)$ recourse budget per update for an exact algorithm. We note that our algorithm can also be made to work in the fully dynamic setting (allowing also interval removals) with the same bounds on the required recolorings.

Before we begin, we introduce some definitions. Let $\mathcal{I} = \{[a_1, b_1), \dots [a_n, b_n)\}$ be a unit interval instance ordered by a_i. A left boundary $\xi_l(\mathcal{I})$ (respectively right boundary $\xi_r(\mathcal{I})$) is a set of intervals intersecting the largest integer smaller than b_1 (respectively the smallest integer larger or equal a_n). Note that $[a_1, b_1) \in \xi_l(\mathcal{I})$ and $[a_n, b_n) \in \xi_r(\mathcal{I})$. A circular arc graph is an intersection graph of (open) arcs lying on the same circle.

▶ **Theorem 2.** *There exists an algorithm which maintains a $(k+1)$-coloring of a k-colorable unit interval graph with $\mathcal{O}(k^2)$ worst case recourse budget per update.*

Proof. We partition the current instance $\mathcal{I}$ into smaller instances $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_m$ and separators between them. Each instance is of size at least lk (except for the last one, which may be smaller), and at most $2lk + k$ for $l = \max\{4, k + 1\}$. The reason for this particular choice of l will become apparent later. In the beginning there is just one instance $\mathcal{I}_1$. Whenever an instance $\mathcal{I}_i$ grows above size $2lk + k$, we pick a point p, such that there are at least lk intervals in $\mathcal{I}_i$ completely to the left and lk intervals completely to the right of p. This point

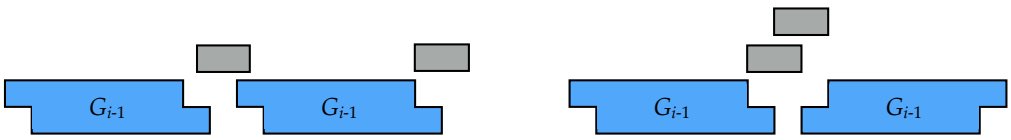

Figure 1 Illustration of the two cases in the recursive construction of G_i.

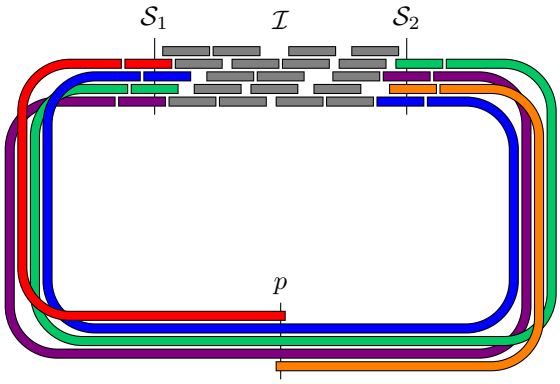

Figure 2 Illustration of the proof of Lemma 3 for $k = 5$.

partitions $\mathcal{I}_i$ in the desired way. We declare the intervals intersecting p to be a separator $\mathcal{S}_i$. At any point in time we maintain a partition of the current instance $\mathcal{I}$ into small instances and separators: $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{S}_1 \cup \mathcal{I}_2 \cup \mathcal{S}_2 \ldots \cup \mathcal{S}_{m-1} \cup \mathcal{I}_m$, where $m \in \Theta(n/(k^2))$. When adding a new interval, we will recolor the instance $\mathcal{I}_i$ into which the new interval falls, or separator $\mathcal{S}_i$ with neighboring instances if the new interval hits the integer point defining $\mathcal{S}_i$. The next lemma will be used to do this with at most $k+1$ colors without changing the colors of the neighboring separators (which are given by some boundary integer points).

▶ **Lemma 3.** *Let $\mathcal{I}$ be a k-colorable unit interval instance. If $|\mathcal{I}| \geq lk$ for $l = \max\{4, k+1\}$, then, for any fixed coloring on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ using colors from $[k]$, one can complete this coloring on $\mathcal{I}$ using colors from $[k+1]$.*

Proof. We first reduce the color completion problem from the lemma statement to the problem of coloring circular arc graphs. This reduction is shown in Figure 2. We draw the intervals of $\mathcal{I}$ as arcs on the north half of a circle, in a way that preserves the intersection relation. Let p be the south pole of the circle, i.e., the point extending the most to the south. For each pair of intervals $(I_1, I_2) \in \xi_l(\mathcal{I}) \times \xi_r(\mathcal{I})$ such that I_1 and I_2 are precolored with the same color, we stretch I_1 (respectively I_2) anticlockwise (respectively clockwise) so that they reach p and then glue them together to form the same arc. The remaining intervals of $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ are only stretched to reach (and intersect) p and are not glued with anything.

We now make use of the following lemma from [34], which allows us to color the obtained circular arc graph instance. We note that this theorem was also used in [15].

▶ **Lemma 4** ([34])**.** *Let G be a circular arc graph, $L(G)$ be the maximum number of arcs intersecting a common point on the circle, and $l(G)$ be the smallest number of intervals that cover the circle. If $l(G) \geq 5$ then $\left\lceil \frac{l(G)-1}{l(G)-2} L(G) \right\rceil$ colors suffice to color G and there is a linear time coloring algorithm.*

In order to apply Lemma 4, we need to consider quantities $L(G)$ and $l(G)$ for the instance G that we created. Before the transformation, since $\mathcal{I}$ is k-colorable, there are at most k intervals intersecting one point. After the transformation, if we cut out from the circle $[p-\epsilon, p+\epsilon]$ for some $\epsilon > 0$, we get a stretched instance $\mathcal{I}$. So for any point on the circle outside $[p-\epsilon, p+\epsilon]$ there are at most k arcs intersecting it. Within $[p-\epsilon, p+\epsilon]$ also at most k arcs intersect, since for every color used on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ there is precisely one arc intersecting p. So $L(G) \leq k$. Also, because $|\mathcal{I}| \geq lk$ and all intervals have unit length, the distance between $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ is at least l, and so the minimal number of intervals

needed to cover the circle is at least $l+1$, i.e., $l(G) \geq l+2$. Setting $l = \max\{4, k+1\}$ ensures $l(G) \geq 5$ so that the assumptions of Lemma 4 are satisfied and we ensure that $l(G) \geq k+2$. Due to Lemma 4, we can color $\mathcal{I}$ with a number of colors bounded by $\left\lceil \frac{l(G)-1}{l(G)-2} L(G) \right\rceil = \left\lceil (1+\frac{1}{l(G)-2}) L(G) \right\rceil \leq \lceil (1+\frac{1}{k})k \rceil = k+1$. Also, any intervals $I_1 \in \xi_l(\mathcal{I})$ and $I_2 \in \xi_r(\mathcal{I})$ are colored the same if and only if their precoloring is the same. Hence, we can permute colors in the obtained coloring so that it complies with the precoloring on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$. ◀

When a new interval I_{new} is added, it either fits into an instance $\mathcal{I}_i$ or it belongs to a separator $\mathcal{S}_j$. In the first case, we recolor $\mathcal{I}_i \cup \{I_{\text{new}}\}$ consistently with the current coloring on $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$. In the second case, we color the new interval I_{new} with the first color not used on $\mathcal{S}_j$ and recolor $\mathcal{I}_j$ and $\mathcal{I}_{j+1}$ consistently with the current coloring on $\mathcal{S}_{j-1}$, $\mathcal{S}_j$, and $\mathcal{S}_{j+1}$. What remains to be proved is that we can always recolor the chosen piece using $k+1$ colors. This follows directly from Lemma 3. ◀

3 Low recourse budget for general interval graphs

In this section we focus on presenting the exact algorithm for arbitrary interval graphs with an amortized recourse budget of $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$. Before we move to that, let us mention the bounds for approximating the number of colors (maintaining a ck-coloring is referred to as c-approximation). The algorithm of Kierstead and Trotter [20] can be turned into a 2-approximation if we allow an amortized $\mathcal{O}(\log n)$ recourse budget. The proof of Theorem 5 can be divided into two lemmas that follow below.

▶ **Theorem 5.** *There is an algorithm maintaining a 2-approximate coloring of an interval graph with amortized recourse budget $\mathcal{O}(\log n)$.*

▶ **Lemma 6** ([20])**.** *There is an online algorithm which receives an interval graph G in an online way and produces a partition of G into subgraphs $P_1, \ldots, P_\omega$, where each P_i is a sum of disconnected paths and ω is a clique number of G.*

▶ **Lemma 7.** *There is an incremental algorithm which uses 2 colors on a sum of disconnected paths P with $n \log_2 n$ total changes, where n is a size of P.*

Proof of Lemma 6. While the algorithm receives next vertices, it tries to satisfy the following invariant.

- **(I)** For any $j \leq \omega$ each clique in $P_1 \cup P_2 \cup \ldots \cup P_j$, has size at most j.
- **(II)** For any $j \leq \omega$ and for any vertex $u \in P_j$ there is a clique in $P_1 \cup P_2 \cup \ldots \cup P_{j-1} \cup \{u\}$ of size j.

When new vertex v is presented, the algorithm finds the last j for which the invariant **(I)** does not hold plus one, i.e. algorithm finds $j_0 := \max\{j \in \mathbb{N} : \omega(P_1 \cup P_2 \cup \ldots \cup P_{j-1} \cup \{v\}) \geq j\}$. Than, it adds v to P_{j_0}, i.e., defines a new partition $P_1^+, \ldots, P_\omega^+$ of a new graph $G^+ = G \cup \{v\}$ in this way that $P_{j_0}^+ := P_{j_0} \cup \{v\}$ and $P_i^+ := P_i$ for $i \neq j_0$. The invariant **(I)** for P_j^+'s is trivially satisfied. The number $j_0 - 1$ is too small, i.e., there is a clique $K \in P_1 \cup P_2 \cup \ldots \cup P_{j_0-1} \cup \{v\}$ of size j_0, which contains the newly presented vertex v. Exactly this clique $K \subseteq P_1^+ \cup \ldots \cup P_{j_0-1}^+ \cup \{v\}$ is a witness for the invariant **(II)** for v. Moreover, the number j_0 is defined so that it will never be greater than the clique number of the graph G.

To understand why each P_i is a sum of disconnected paths, let's consider the interval representation $\mathcal{I}$ of graph G. It means that $\mathcal{I}$ is a family of closed intervals in $\mathbb{R}$. Moreover, for each $j \leq \omega$ let's define $\mathcal{I}_j$ as a family of intervals corresponding to the vertices of P_j. First, we note the following claim.

▷ Claim 8. There is no interval in $\mathcal{I}_j$ which is covered by the rest of the intervals from $\mathcal{I}_j$.

Proof. For the contradiction let's assume that there are different intervals $I_0, I_1, \ldots, I_t \in \mathcal{I}_j$ such that $I_0 \subseteq I_1 \cup \ldots \cup I_t$. Let's $K \subseteq P_1 \cup \ldots \cup P_j$ be a clique for I_0 from invariant **(II)**. Each clique in the interval representation can be identified with some real number that belongs to all intervals corresponding to elements from that clique. Let's $r \in \mathbb{R}$ be such a number corresponding to the clique K. Then $r \in I_1 \cup \ldots \cup I_t$ and in consequence $r \in I_s$ for some $s \leq t$. If vertex v_s corresponds to the interval I_s then $K \cup \{v_s\} \subseteq P_1 \cup \ldots \cup P_j$ forms a clique of size $j+1$ which contradicts the invariant **(I)**. ◁

The above claim directly implies the following statement.

▷ Claim 9. Each vertex in P_j has at most two neighbours in P_j.

Proof. Again, let's $\mathcal{I}_j$ be a family of interval corresponding to vertices from P_j. For the contradiction let's assume that v_0 has three neighbours v_1, v_2, v_3 which corresponds to the intervals I_0, I_1, I_2, I_3. At the beginning, notice that the sum $I_0 \cup I_1 \cup I_2 \cup I_3$ form also some interval in $\mathbb{R}$. Let's l and r be the left and the right endpoint of $I_0 \cup I_1 \cup I_2 \cup I_3$, respectively. One of the intervals I_1, I_2, I_3 does not contain any points of l, r. Without loss of generality let us assume that this interval is I_3. Then $I_3 \subseteq I_0 \cup I_1 \cup I_2$ which is contradictory to the previous claim. ◁

Finally, it is worth noting that interval graphs are also chordal, so they can not contain simple cycles. So, the only possibility is that P_j is a sum of disconnected paths. ◀

Proof of Lemma 7. When new vertex v is coming, it combines two paths. If neighbours of v have the same color then the algorithm colors vertex v on the other one. If neighbours of v have the different colors then the algorithm recolors the shortest path. The given vertex u was recolored when the length of the path containing u increased by at least twice. This causes the vertex u to be recolored at most $\log_2 n$ times. Which gives the total number of recoloring equal $n \log_2 n$. ◀

In the remainder of this section we show a k-coloring algorithm with $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ recourse budget. Both for the algorithm and the analysis we use the greedy algorithm for coloring interval graphs [26]. The greedy algorithm sorts intervals by their begin coordinates. It processes intervals in that order, and assigns the smallest available colour to the currently processed interval. This simple algorithm was proven optimal [26]. We are now ready to prove the main theorem of this section.

▶ **Theorem 10.** *There is an algorithm maintaining an optimum coloring of a k-chromatic interval graph with an amortized recourse budget of* $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$.

Proof. Note that a trivial algorithm, which recolors all intervals in each step has recourse budget n. We will show that there also is an algorithm with amortized budget $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, which proves the claim. This algorithm is directly implied by Lemma 11, which is proved next. Due to this lemma n interval insertions into an n-element instance can be executed with a total recourse budget of $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$. The implication is as follows. Imagine we make a total of m insertions. We break the insertion sequence into powers of 2: once we inserted 2^i intervals, we add 2^i more using $\mathcal{O}(k \cdot k! \cdot 2^i\sqrt{2^i})$ recolorings. Let s be such that $2^{s-1} < m \leq 2^s$. The total number of recolorings is bounded by $\sum_{i=1}^{s} \mathcal{O}(k \cdot k! \cdot 2^i\sqrt{m}) = \mathcal{O}(k \cdot k! \cdot \sqrt{m} \sum_{i=1}^{s} 2^i) = \mathcal{O}(2^{s+1} k \cdot k! \cdot \sqrt{m}) = \mathcal{O}(k \cdot k! \cdot m\sqrt{m})$. ◀

▶ **Lemma 11.** *There is an algorithm, which, given n intervals, maintains the exact coloring over the course of n interval insertions and recolors a total of $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$ intervals.*

Proof. We move on to presenting the algorithm, followed by the analysis. The idea is to maintain a partition of the dynamically changing instance $\mathcal{I}$ into l disjoint instances $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_l$. We maintain the invariant that the size of each instance is at most $2\lceil\sqrt{n}\rceil + 2k$, and that the size of each instance but the last one is at least $\lceil\sqrt{n}\rceil$. This invariant guarantees that $l \in \mathcal{O}(\sqrt{n})$. At the beginning, the algorithm starts with n intervals, so $|\mathcal{I}| = n$. Then, it is easy to find such a partition. Let $\mathcal{I} = \{[a_1, b_1), \ldots [a_n, b_n)\}$ be sorted by end coordinates. We let $x_1 = b_{\lceil\sqrt{n}\rceil}$ be the first separator point. It may happen that up to k intervals end in the same coordinate, so there are at most $\lceil\sqrt{n}\rceil + k$ intervals to the left of x_1. We remove intervals to the left and intersecting x_1 from $\mathcal{I}$ and continue in the same manner in order to find separating points $x_2 \ldots x_{l-1}$. We let $\mathcal{I}_i$ be the intervals contained between x_{i-1} and x_i, and we define separator $\mathcal{S}_i$ to be the set of all intervals intersecting x_i. Note that the separators are not necessarily disjoint, since intervals can span a long stretch in which many smaller intervals live.

Now consider the dynamically growing instance. If at any time some instance $\mathcal{I}_i$ grows to more than $2\lceil\sqrt{n}\rceil + 2k$, we split it into instances $\mathcal{I}'_i$ and $\mathcal{I}''_i$, both of size at least $\lceil\sqrt{n}\rceil$, since the separator takes away at most k intervals, and we possibly have to put $\lceil\sqrt{n}\rceil + k$ intervals into $\mathcal{I}'_i$. At this point $\mathcal{I}_i$ ceases to exist. This ensures that our size invariant remains satisfied at all times.

In each step, the algorithm takes a new interval I_{new} as input. It uses a procedure `total-recolor`(i, j) as a subroutine. Procedure `total-recolor`(i, j) takes two numbers $i, j \in \{1, \ldots, l-1\}$, $i \leq j$ as parameters. It is an invariant that I_{new} is entirely contained in (x_{i-1}, x_j). The procedure recolors the new instance $\mathcal{I} \cup \{I_{\text{new}}\}$ in the following way. It leaves the current coloring as it is on $\mathcal{I}_1, \mathcal{S}_1, \mathcal{I}_2, \mathcal{S}_2, \ldots, \mathcal{I}_{i-1}, \mathcal{S}_{i-1}$. Starting with the current coloring on $\mathcal{S}_{i-1}$, it colors $\mathcal{I}_i \cup \mathcal{S}_i \cup \ldots \cup \mathcal{I}_j \cup \mathcal{S}_j \cup \{I_{\text{new}}\}$ greedily. The greedy coloring is consistent with the coloring of $\mathcal{S}_{i-1}$, but may not be consistent with the current coloring on $\mathcal{S}_j$. Nevertheless, we can permute the colors in order to obtain the new greedy coloring on $\mathcal{S}_j$. The procedure permutes the colors in the same way on the remaining part of the instance, i.e., for $\mathcal{I}_{j+1}, \mathcal{S}_{j+1}, \ldots, \mathcal{S}_{l-1}, \mathcal{I}_l$. Procedure `total-recolor`$(i, j)$ possibly recolors the whole graph, i.e, it triggers $\mathcal{O}(n)$ recolorings.

Having procedure `total-recolor`(i, j) at hand, the algorithm distinguishes two cases.

1. $I_{\text{new}} \in \mathcal{I}_j$ for some $\mathcal{I}_j$. In this case we try to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ with k colors in a way that is consistent with the current coloring on $\mathcal{S}_{j-1}$ and $\mathcal{S}_j$ (see the parameterized algorithm of Marx [22] for efficient implementation). There are two more cases now.
   a. It is possible to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ consistently with $\mathcal{S}_{j-1}$ and $\mathcal{S}_j$. In this case we perform $\mathcal{O}(\sqrt{n} + k)$ recolorings.
   b. It is impossible to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ in this way. In this case we call `total-recolor`(j, j).
2. I_{new} intersects some separation point. If $x_i, x_{i+1}, \ldots x_{j-1}$ are the x-coordinates of the separation points intersected by I_{new}, we call `total-recolor`(i, j).

As for the analysis of the above algorithm, the recoloring budget claimed in Lemma 11 follows from Lemma 12 and Lemma 13 below. Observe, that the only expensive operation we need to amortize for is `total-recolor`$(\cdot, \cdot)$, which performs $\mathcal{O}(n)$ recolorings. Due to Lemma 12, the total number of recolorings triggerred by `total-recolor`(i, j) for $i \neq j$ is $\mathcal{O}(k \cdot n\sqrt{n})$. Due to Lemma 13, the total number of recolorings triggered by `total-recolor`(i, i) on a particular instance $\mathcal{I}_i$ is $\mathcal{O}(k \cdot k! \cdot n)$. Observe, that the number of instances that

ever exist is $\mathcal{O}(\sqrt{n})$: the algorithm starts with n intervals, and for these initial intervals it creates $l \in \mathcal{O}(\sqrt{n})$ instances. Further on it creates at most $\mathcal{O}(\sqrt{n})$ more instances by splitting the existing ones. Summed over all instances that exist at some point of time this gives $\mathcal{O}(k \cdot k! \cdot \sqrt{n} \cdot n)$ recolorings. The total number of recolorings caused by case 1 a) of the algorithm is bounded by $\mathcal{O}(n(\sqrt{n} + k))$. The number of all recolorings the algorithm performs is hence bounded by $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$, as claimed. ◀

▶ **Lemma 12.** *The total number of calls to* `total-recolor`(i, j) *for any* $i \neq j$ *is in* $\mathcal{O}(k\sqrt{n})$.

Proof. The call to `total-recolor`(i, j) for $i \neq j$ is only made if I_{new} intersects some separator line. There are $\mathcal{O}(\sqrt{n})$ separator lines created by the algorithm, and at most k intervals may be added to each separator. This gives the claim of the lemma. ◀

▶ **Lemma 13.** *For every instance* $\mathcal{I}_j$ *the algorithm calls* `total-recolor`(j, j) *at most* $2k \cdot k!$ *times overall in step 1b).*

Proof. Fix $i \in [1, \ldots, l]$ and consider the pair of separators $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$. We say that $\mathcal{I}_i$ is reset when procedure `total-recolor`(j_1, j_2) is called with $j_1 \neq j_2$ for $j_1 \leq i \leq j_2+1$. In what follows we will prove that between two consecutive resets of $\mathcal{I}_i$, procedure `total-recolor`(i, i) can be called at most $k!$ times. This will finish the proof, as any `total-recolor`(j_1, j_2) call resetting $\mathcal{I}_i$ adds an interval to either $\mathcal{S}_{i-1}$ or $\mathcal{S}_i$ or both, so there can be at most $2k$ such calls. Note that non-resetting calls of `total-recolor`() do not alter $\mathcal{S}_{i-1}$ and do not alter $\mathcal{S}_i$, so between two resets of $\mathcal{I}_i$ separators $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ remain unchanged (although their colors may change). It may happen that we split $\mathcal{I}_j$, but then $\mathcal{I}_j$ ceases to exists and hence is recolored no more (instead, the instances that $\mathcal{I}_j$ splits into are recolored). In what follows we consider a time period between two consecutive resets of $\mathcal{I}_i$. We refer to this time period as a phase. The phase starts when an interval has been added to either $\mathcal{S}_{i-1}$ or $\mathcal{S}_i$ or both and lasts as long as no other interval is added to $\mathcal{S}_{i-1}$ or $\mathcal{S}_i$ and as long as $\mathcal{I}_i$ is not split.

Let $\mathcal{I}_i^f$ be the instance $\mathcal{I}_i$ after the last insertion within the phase. In what follows we always view $\mathcal{I}_i$ as a current instance, before inserting a new interval I_{new}. We let $\mathcal{J}_i = \mathcal{S}_{i-1} \cup \mathcal{I}_i \cup \mathcal{S}_i$ and $\mathcal{J}_i^f = \mathcal{S}_{i-1} \cup \mathcal{I}_i^f \cup \mathcal{S}_i$.

For solution Sol maintained by the algorithm we define Sol_{i-1} and Sol_i to be Sol restricted to $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ respectively. Similarly, for any optimum solution Opt for $\mathcal{J}_i^f$ we define its restriction to $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ as Opt_{i-1} and Opt_i. Let `Greedy`$(\mathcal{J}_i^f)$ be the optimal greedy solution to $\mathcal{J}_i^f$. Observe that if we permute colors of an optimal solution for $\mathcal{J}_i^f$, the solution remains optimal. This leads us to define the optimal solution space $\Sigma = \mathfrak{S}_k \circ$ `Greedy`$(\mathcal{J}_i^f)$, where $\mathfrak{S}_k$ denotes the permutation group on $[k]$. In other words, Σ contains all color permutations of `Greedy`$(\mathcal{J}_i^f)$. Observe that Σ is closed under taking permutations.

Let now Sol be the solution produced by the algorithm at the beginning of the phase, i.e., after the reset insertion. Let $\text{Opt} \in \Sigma$ be the optimal solution such that $\text{Opt}_{i-1} = \text{Sol}_{i-1}$. One must exist, since we can permute the colors of `Greedy`$(\mathcal{J}_i^f)$ in order to match Sol on $\mathcal{S}_{i-1}$. Let $\tau_S \in \mathfrak{S}_k$ be any permutation such that $\text{Sol}_i = \tau_S \circ \text{Opt}_i$. Observe, that if τ_S can be chosen as identity permutation, `total-recolor`(i, i) is never called in this phase. Hence, we may assume that τ_S is not the identity. So far we have $\text{Sol}_{i-1} = \text{Opt}_{i-1}$ and $\text{Sol}_i = \tau_S \circ \text{Opt}_i$.

Within the phase there are two types of events that affect the coloring maintained by the algorithm on $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$. Event of type A is a call to `total-recolor`(j, k) for $k < i$, which permutes the colors on $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ with the same permutation. Event of type B is a call to `total-recolor`(i, i), which leaves the colors on $\mathcal{S}_{i-1}$ intact while permuting colors on $\mathcal{S}_i$.

Let us define $\textsc{Sol}^{(j)}$ to be the solution maintained by the algorithm right after the j'th event. For some $\sigma, \tau \in \mathfrak{S}_k$ we get $\textsc{Sol}^{(j)}_{i-1} = \sigma \circ \textsc{Sol}^{(j-1)}_{i-1}, \textsc{Sol}^{(j)}_{i} = \sigma \circ \textsc{Sol}^{(j-1)}_{i}$ if the j'th event is of type A and $\textsc{Sol}^{(j)}_{i-1} = \textsc{Sol}^{(j-1)}_{i-1}, \textsc{Sol}^{(j)}_{i} = \tau \circ \textsc{Sol}^{(j-1)}_{i}$ if the j'th event is of type B.

Also, after the j'th event, we define $\sigma_j, \tau_j \in \mathfrak{S}_k$ to be such that $\textsc{Sol}^{(j)}_{i-1} = \sigma_j \circ \textsc{Sol}_{i-1}$ and $\textsc{Sol}^{(j)}_{i} = \tau_j \circ \textsc{Sol}_{i}$. Our goal is to obtain ${\tau_j}^{-1} \circ \sigma_j = \tau_S$ for some j. If that holds then `total-recolor`(i,i) is never called again in this phase, because then we have $\textsc{Sol}^{(j)}_{i-1} = \sigma_j \circ \textsc{Sol}_{i-1} = \sigma_j \circ \textsc{Opt}_{i-1}$ and $\textsc{Sol}^{(j)}_{i} = \tau_j \circ \textsc{Sol}_{i} = \tau_j \circ \tau_S \circ \textsc{Opt}_i = \sigma_j \circ \textsc{Opt}_i$. But then there is optimal solution $\sigma_j \circ \textsc{Opt}$ that certifies that we can recolor $\mathcal{J}_i$ in compliance with $\textsc{Sol}^{(j)}_{i-1}$ and $\textsc{Sol}^{(j)}_{i}$.

Now observe, that if we apply the same permutation $\alpha \in \mathfrak{S}_k$ to both $\textsc{Sol}^{j}_{i-1}$ and $\textsc{Sol}^{j}_{i}$, i.e., if $\textsc{Sol}^{(j+1)}_{i-1} = \alpha \circ \textsc{Sol}^{(j)}_{i-1} = \alpha \circ \sigma_j \circ \textsc{Sol}_{i-1}$ and $\textsc{Sol}^{(j+1)}_{i} = \alpha \circ \textsc{Sol}^{(j)}_{i} = \alpha \circ \tau_j \circ \textsc{Sol}_{i}$, then $\tau^{-1}_{j+1} \circ \sigma_{j+1} = (\alpha \circ \tau_j)^{-1} \circ \alpha \circ \sigma_j = \tau_j^{-1} \circ \sigma_j$, so permutation $\tau_j^{-1} \circ \sigma_j$ stays the same when permuting colors on $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ in the same way. Hence, the only way it can change is due to `total-recolor`(i,i).

However, if `total-recolor`(i,i) is called, that means that the new interval causes that the current coloring $\textsc{Sol}^{(j)}_{i-1}$ and $\textsc{Sol}^{(j)}_{i}$ cannot be used on $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ now, and hence it cannot be used ever again in the future. This holds because we only add intervals, so any future instance contains the current instance, and any coloring for the future instance is a coloring for the current instance as well. This means that for $k > j$ we have $\tau_k^{-1} \circ \sigma_k \neq \tau_j^{-1} \circ \sigma_j$. For the proof of this fact assume otherwise: $\tau_j^{-1} \circ \sigma_j = \tau_k^{-1} \circ \sigma_k = (\alpha \circ \tau_j)^{-1} \circ \beta \circ \sigma_j$. This implies $\alpha = \beta$ and $\textsc{Sol}^{(k)}_{i-1} = \alpha \circ \textsc{Sol}^{(j)}_{i-1}$ and $\textsc{Sol}^{(k)}_{i} = \alpha \circ \textsc{Sol}^{(j)}_{i}$. But this cannot happen since we already know that the combined coloring $\textsc{Sol}^{(j)}_{i-1}$ and $\textsc{Sol}^{(j)}_{i}$ cannot be used for $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$, and neither can any permutation of this coloring. But permutation $\sigma_j^{-1} \circ \tau_j$ can only take $k!$ different values until it reaches τ_S. This concludes the proof. ◀

4 Lower bounds for general interval graphs

In this section we provide lower bounds on the recourse budget needed in order to maintain an optimum coloring of an interval graph. The following definition allows us to compare different colorings locally and to formulate necessary conditions for optimum colorings.

▶ **Definition 14.** *Let $\mathcal{I}$ be a set of intervals, let $k \in \mathbb{N}$ be the chromatic number of $\mathcal{I}$, and let $R = [a,b) \subset \mathbb{R}$. The* gap *of a set $C \subseteq \mathcal{I}$ of disjoint intervals is given by* $\text{gap}_R(C) := |R| - \sum_{I \in C} |R \cap I|$. *The* total gap *of a partition $\mathcal{C}$ of $\mathcal{I}$ into disjoint sets wrt. R is* $\text{gap}_R(\mathcal{C}) := \sum_{C \in \mathcal{C}} \text{gap}_R(C)$. *The* total gap *of $\mathcal{I}$ wrt. R is given by* $\text{gap}_R(\mathcal{I}) := k \cdot |R| - \sum_{I \in \mathcal{I}} |R \cap I|$.

The following fact provides a formal criterion for optimality of a coloring. Note that in every proper coloring all intervals receiving the same color are disjoint.

▶ **Fact 15.** *We have $\text{gap}_R(\mathcal{I}) = \text{gap}_R(\mathcal{C}^\star)$, where $\mathcal{C}^\star$ is a partition of $\mathcal{I}$ corresponding to any optimum coloring of $\mathcal{I}$.*

We are now ready to construct an instance that requires many recolorings. The main building block for the bad instance is a *staircase* gadget S_k that guarantees a linear number of recolorings overall (cf. Fig. 3). We will later use multiple copies of this gadget to force $\Omega(\sqrt{n})$ amortized recolorings.

The gadget consists of three sets L, X, R of intervals. We start with an initial configuration of intervals in these sets, which we assume can be colored optimally with k colors without ever recoloring (if an algorithm needs recolorings, this only strengthens our bound). We

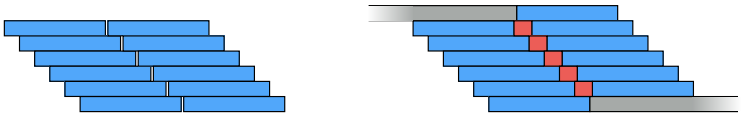

Figure 3 Illustration of the open (left) and closed (right) staircase gadget.

call the initial configuration *open*. Later, we introduce additional intervals in each of the three sets in such a way that the chromatic number increases by exactly one, to $k+1$, and such that a significant portion of the previously colored intervals need to be recolored in order not to exceed $k+1$ colors. We refer to the final configuration of the staircase as *closed*. Importantly, we ensure that both in the open and the closed configuration there is a unique way to optimally color the intervals (apart from renaming colors). This ensures that "from the outside" the gadget behaves like a clique of k intervals in the open configuration and a clique of $k+1$ intervals in the closed configuration.

We start by describing the open (initial) configuration (cf. Fig. 3 (left)). We set $L = \{L_i\}_{i=1}^{k} := \{[i-\Delta, i)\}_{i=1}^{k}$, $X = \emptyset$, and $R = \{R_i\}_{i=1}^{k} := \{[i+\varepsilon, i+\Delta)\}_{i=1}^{k}$, where $0 < \varepsilon < 1/k$ is sufficiently small and $\Delta \geq k+1$ is sufficiently large. Observe that the open staircase can be colored with k colors simply by coloring L_i, R_i with color i, and k colors are needed because L and R each are a clique of size k. The total gap in the interval $[1, k+\varepsilon)$ is $\text{gap}_{[1,k+\varepsilon)}(L \cup R) = k\varepsilon < 1$. By Fact 15, no optimal solution with k colors can therefore afford to leave a gap of size 1 or larger in any color. Since L and R each form a clique, assigning the same color to $L_1 = [1-\Delta, 1)$ and $R_i = [i+\varepsilon, i+\Delta)$ with $i \geq 2$ leads to a gap of $i+\varepsilon-1 > 1$, and it follows that L_1, R_1 must get the same color. Repeating this argument, so must L_i, R_i for every $i \in \{1, \ldots, k\}$. This means that (up to permuting the colors) there is a unique coloring of the open staircase with k colors, as intended.

To obtain the closed configuration (cf. Fig. 3 (right)), we add the interval $L_0 := (-\infty, 1+\varepsilon)$ to L, the interval $R_{k+1} := [k, \infty)$ to R, and the intervals $X = \{X_i\}_{i=1}^{k-1} := \{[i, i+1+\varepsilon)\}_{i=1}^{k-1}$. Note that the sets of intervals of the closed staircase can be colored with $k+1$ colors and zero total gap in the interval $[1, k+\varepsilon)$: we can simply color L_{i-1}, R_i with color i for $i \in \{1, \ldots, k+1\}$ and X_i with color $i+1$ for $i \in \{1, \ldots, k-1\}$. This means that every coloring with $k+1$ colors must have zero total gap in the interval $[1, k+\varepsilon)$, by Fact 15. Since every point is the endpoint of at most two intervals in L, X, R, there is a unique way of coloring the closed staircase with $k+1$ colors, as intended.

Finally, consider the bipartite graph that has the elements of L on one side and the elements of R on the other, with an edge connecting an interval from L to an interval from R if they do not intersect. The staircase matching induces a unique matching in this graph, where each edge selected in the matching corresponds to a color. We call this matching the *stair matching* of S_k and conclude the following lemma.

▶ **Fact 16.** *The staircase gadget S_k has chromatic number k when open and $k+1$ when closed. In either configuration there is a unique optimum coloring (up to renaming colors), and the stair matchings of these two colorings are perfect and disjoint.*

Since the stair matchings are disjoint, when adding intervals to obtain the closed staircase from the open one, many intervals need to be recolored.

▶ **Fact 17.** *When transitioning from the open to the closed staircase gadget, at least k intervals of the open staircase must be recolored to maintain an optimum coloring.*

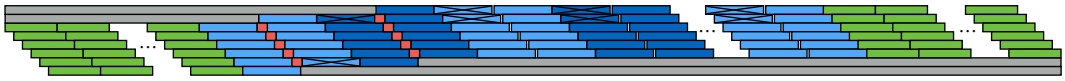

Figure 4 Illustration of the construction in the proof of Theorem 19 after round 2. Green intervals are (shifted) copies of Z and crossed-out intervals are passive.

We now describe a *connector* gadget C_k that generalizes the interface between consecutive staircase gadgets as well as further gadgets. The connector gadget consists of an *L-connector* and an *R-connector*, and is defined as follows. The L-connector of size k is a set of intervals of the form $\{[a_i, x+i)\}_{i=1}^k$ with $a_i \leq x$, and the R-connector of size k is of the form $\{[x+i, b_i)\}_{i=1}^k$ with $b_i > x+k$. Here $x \in \mathbb{R}$ is an arbitrary offset. Together, the L-connector and R-connector form the connector. Observe that for $i \in \{1, \ldots, k\}$ the intervals L_i are an R-connector, and the intervals R_i are an L-connector. The following property of connector gadgets is obvious.

▶ **Fact 18.** *There is a unique coloring of the connector gadget C_k with k colors (up to renaming colors).*

▶ **Theorem 19.** *For every $k \in \mathbb{N}$, there is an instance of online interval graph coloring with chromatic number $\Theta(k)$ and $\Theta(k^2)$ vertices that requires an amortized recourse budget of $\Omega(k)$ to maintain an optimum solution.*

Proof. We fix any number $k \in \mathbb{N}$ and any online coloring algorithm. We start by introducing a large set of intervals offline that we allow the algorithm to color in a batch (i.e., not online and without need to recolor), before introducing additional intervals online that each require significant recoloring.

We first describe the offline intervals. We introduce multiple gadgets that each start with an R-connector and end with an L-connector. In the following, each gadget (after the first) is shifted to the right, such that it forms a connector gadget with the previous gadget. Let $Z := \{[i, k+i)\}_{i=1}^k$, i.e., Z is both an R- and an L-connector. We introduce k copies of Z, each shifted as described (green intervals in Fig. 4).

Since the copies of Z form a chain of connector gadgets, by Fact 18, there is a unique way to color these gadgets with k colors. We further introduce k shifted open staircase gadgets and then another k shifted copies of Z. Overall, our construction so far uses $n_{\text{open}} = 4k^2$ intervals, has chromatic number k, and, by Fact 16 and Fact 18, there is a unique way to color all intervals with k colors.

We now present additional sets of intervals online in k rounds. In each round, we close the leftmost open staircase gadget by introducing $k+1$ new intervals. In each round the new intervals of the form L_0 and R_{k+1} overlap all intervals outside the staircase being closed. Thus, while the chromatic number increases by one, the effective number of available colors in all gadgets to the right remains unchanged. We call an interval of a staircase *passive* if it is part of an R-connector (resp. L-connector) and shares a color with any interval of the form L_0 (resp. R_{k+1}), and *active* otherwise. This means in particular that in each round a single interval of every open staircase becomes passive. By Corollary 17, at least k intervals of a staircase need to be recolored when it is being closed. Since each active interval is part of a connector gadget outside the staircase, and since each such connector gadget and every other staircase must be colored in a unique way (Fact 16 and Fact 18), recoloring an active interval requires to recolor all other intervals of the same color to the left or to the right of the staircase. Thus, in round i, at least $k-i+1$ active intervals need to be recolored, each affecting k copies of Z, such that the total number of intervals that need

to be recolored is at least $(k-i+1)k$. After closing the staircase, by Fact 16, there is again a unique way to color it. This means that we can repeat the process with the next staircase, restricting everything to the colors that are occupied by the current gadget, and so on. Overall, the number of intervals that need to be recolored in k rounds is at least $\sum_{i=1}^{k}(k-i+1)k = k^3 - k^2(k+1)/2 + k^2 = \Omega(k^3)$.

Overall, we introduce $k+1$ new intervals in each round, so the total number of intervals is $n = n_{\text{open}} + k(k+1) = 5k^2 + k$. The chromatic number increases by one in every round, hence the chromatic number of the final graph is $k' = 2k$. The amortized recourse budget the algorithm needs thus is $\Omega(k^3/n) = \Omega(k)$, as claimed. ◀

The next statements follow from Theorem 19 by setting $k = \Theta(\sqrt{n})$, and by observing that we can always add isolated vertices without affecting k.

▶ **Corollary 20.** *Maintaining an optimum coloring of an interval graph online, requires an amortized recourse budget of $\Omega(\sqrt{n})$ in general (when k is not fixed).*

▶ **Corollary 21.** *Maintaining an optimum coloring of an interval graph with chromatic number $k \in \mathcal{O}(\sqrt{n})$ online, requires an amortized recourse budget of $\Omega(k)$ in general.*

5 Trading off recourse budget with query times

Up to this point we worked in a model where we need to maintain the coloring explicitly, i.e., after each insertion of an interval we need to recolor every interval whose color changes. We showed an algorithm, which achieves this by recoloring amortized $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ intervals, and for this algorithm an efficient implementation is not obvious. In this section we give an efficient algorithm maintaining the optimum coloring, but we relax the model. So far we insisted on recoloring all intervals immediately. This requirement allows us to retrieve the color of any interval in constant time, and is moreover crucial for some applications. In this section we do not focus on maintaining an explicit coloring, but rather we design a coloring oracle: a data structure that can be queried for the color of an interval. Our data structure supports interval additions in $\mathcal{O}(k^2 \log^3 n)$ amortized time, and it answers queries for a color of a particular interval in $\mathcal{O}(\log n)$ time. Between two consecutive updates it answers queries consistently with some optimal proper coloring. We only sketch the data structure here, and leave some details and the formal proof of the following theorem to Appendix A.

▶ **Theorem 22.** *There is a dynamic datastructure that stores a k-colorable set of intervals $\mathcal{I}$ and returns the color of any $I \in \mathcal{I}$ according to an optimum proper coloring of $\mathcal{I}$ in $\mathcal{O}(\log n)$ time. Furthermore, it needs $\mathcal{O}(k^2 \log^3 n)$ amortized time to insert a new interval.*

We store the intervals of the instance $\mathcal{I}$ in a modified *interval tree* [13]. That is, we maintain a binary search tree T, for which each node v stores the x-coordinate $l_v \in \mathbb{R}$ of a vertical line and a subset $\mathcal{S}_v \subseteq \mathcal{I}$ of intervals. For a node v of T, let T_v be the subtree of T rooted at v, and let $\mathcal{I}_v$ contain all intervals stored in the sets $\mathcal{S}_u$ for nodes u of T_v. We say that an interval I is *stored in* T_v if T_v has a node u such that $I \in \mathcal{S}_u$, i.e., $I \in \mathcal{I}_v$. The tree T has the following properties.

1. If $\mathcal{I}_v = \emptyset$ then v is a leaf of T with undefined value l_v and empty set $\mathcal{S}_v$.
2. Otherwise, T_v has a defined value $l_v \in \mathbb{R}$ and two child nodes x and y in T, for which the (defined) values l_u of all nodes u of T_x are smaller than l_v, while the (defined) values l_u of all nodes u of T_y are larger than l_v. The trees T_x and T_y are called the *left* and *right subtree* of T_v, respectively.

3. The set $\mathcal{S}_v = \{I \in \mathcal{I}_v \mid \mathrm{beg}(I) \leq l_v \leq \mathrm{end}(I)\}$ contains all intervals of $\mathcal{I}_v$ intersecting l_v. The left and right subtrees T_x and T_y of T_v recursively store all intervals in $\mathcal{I}_x = \{I \in \mathcal{I}_v \mid \mathrm{end}(I) < l_v\}$ and $\mathcal{I}_y = \{I \in \mathcal{I}_v \mid \mathrm{beg}(I) > l_v\}$, respectively.

The sets $\mathcal{S}_v$ stored in all nodes v of T partition $\mathcal{I}$, and an interval $I \in \mathcal{I}$ is stored in the highest node v of T for which I contains l_v. Therefore each set $\mathcal{S}_v$ is a separator of the intervals $\mathcal{I}_v$ stored in T_v, i.e., no interval from the left subtree of T_v overlaps with an interval from the right subtree of T_v. Furthermore, the intervals of any set $\mathcal{S}_v$ form a clique, as they all intersect l_v, and thus at most k intervals are stored in a node v if $\mathcal{I}$ is k-colorable.

For reasons that will become apparent later, at all times we will make sure that the root r of T has x-coordinate $l_r = 0$, and we assume w.l.o.g. that $\mathrm{beg}(I) > 0$ for all $I \in \mathcal{I}$. This means that all intervals of $\mathcal{I}$ are stored in the right subtree of T. We will also make sure that for any node $v \neq r$ different from the root, if $\mathcal{S}_v = \emptyset$ then v is a leaf of T. This ensures that the number of nodes of T is linear in $n = |\mathcal{I}|$.

Instead of storing the color of each interval explicitly, we associate a permutation $\tau_e \in \mathfrak{S}_k$ with each edge e of the search tree T. Also, for each node v of T we store the intervals of the set $\mathcal{S}_v$ in a fixed order, so that $\mathcal{S}_v = \{I_1, \ldots, I_j\}$ for $j \leq k$. The color of an interval $I_i \in \mathcal{S}_v$ is obtained by applying the permutations along the path P_v from the root r of T to v to the index i. That is, let $e_1, e_2, \ldots, e_h$ be the sequence of edges of P_v such that e_1 is incident to r (note that e_1 connects r to the right subtree of T by our assumption that $l_r = 0$). We denote the composite permutation along the path P_v by $\sigma_{e_h} = (\tau_{e_h} \circ \cdots \circ \tau_{e_2} \circ \tau_{e_1})$, and the color of $I_i \in \mathcal{S}_v$ is $\sigma_{e_h}(i)$. Thus the color of any interval can be retrieved in time linear in the height of the tree, by first finding its index i in the node storing it and then following the search path back to the root to compute the image of i in the composite permutation defining its color. It is also clear that there exist permutations for the edges that imply a proper k-coloring of the intervals if $\mathcal{I}$ is k-colorable. In fact, only the permutation τ_{e_h} of the last edge e_h on P_v for some particular node v needs to be picked in relation to all previous permutations along P_v, so that the indices of $\mathcal{S}_v$ are permuted according to a fixed proper k-coloring.

To obtain logarithmic query times, we make sure that the tree T is α-*balanced* [13] at all times. That is, let $n_v = |\mathcal{I}_v|$ be the number of intervals stored in subtree T_v rooted at v, and let α be a fixed constant such that $1/2 < \alpha < 1$. For any subtree $T_v \neq T$ (i.e., $v \neq r$) we maintain the property that $\max\{n_v^-, n_v^+\} \leq \lceil \alpha n_v \rceil$, where n_v^- and n_v^+ are the number of intervals stored in the left and right subtrees of T_v, respectively. As an easy consequence we get that the height of T is $\log_{1/\alpha}(n) + \mathcal{O}(1) = \mathcal{O}(\log n)$. To maintain this invariant, we store n_v in node v.

5.1 Updates

We now describe how to update the search tree T and the permutations on its edges, so that the colors induced by the permutations form a proper k-coloring and the tree is α-balanced at all times. When a new interval I_{new} arrives, it is stored in the interval tree T in the usual way [13]. That is, we follow the search path for I_{new} starting from the root. As soon as we encounter a node v in T such that I_{new} belongs to the set $\mathcal{S}_v$ (because $l_v \in I_{\mathrm{new}}$), we add I_{new} to $\mathcal{S}_v$. The index i of the new interval I_{new} in $\mathcal{S}_v$ is the highest available, i.e., $i = |\mathcal{S}_v|$ when $I_{\mathrm{new}} \in \mathcal{S}_v$. Additionally, we increase the variables n_u along the nodes u of the path P_v from v to the root of T by one each, to count the new interval I_{new} in the subtrees T_u. If no node v for which $l_v \in I_{\mathrm{new}}$ is found, let w be the leaf of T at the end of the search path P_w for I_{new}. We set $l_w = \mathrm{beg}(I_{\mathrm{new}})$, and add I_{new} as the only interval in the set $\mathcal{S}_w$. We also create two new leaves and set them as the new left and right subtrees of w. Again, we increase the variables n_u along the nodes u of P_w.

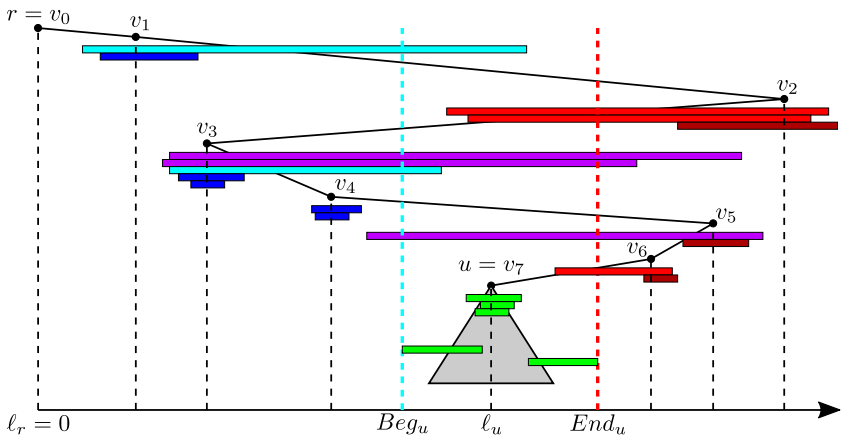

Figure 5 The path P_u with nodes v_0 to v_7 from the root r of the search tree T to the node u. The bars represent intervals, which are stored in the highest node w for which they contain l_w (black dashed lines). The subtree T_u is shaded in grey and stores $\mathcal{I}_u$ (green intervals). The leftmost and rightmost points of these intervals are beg_u and end_u (blue and red dotted lines), which define the sets $\mathcal{L}_u$ and $\mathcal{R}_u$ (including the light blue and light red intervals, respectively). Some intervals can be in the intersection of $\mathcal{L}_u$ and $\mathcal{R}_u$ (purple intervals). The remaining intervals are either to the left of beg_u or to the right of end_u (dark blue and dark red intervals, respectively). In this example, $L = \{v_2v_3, v_6v_7\}$ and $R = \{v_1v_2, v_4v_5\}$.

When adding I_{new} the tree T may become unbalanced, i.e., there may be a node $u \neq r$ of T for which $\max\{n_u^-, n_u^+\} > \lceil \alpha n_u \rceil$. Note that u must be on the path P_v from the root r to the node v into which I_{new} was added. To make T α-balanced again, we identify the closest such node u to the root. We then rebalance T_u by first retrieving all intervals $\mathcal{I}_u$ stored in T_u, and then sorting all the endpoints $\text{beg}(I)$ and $\text{end}(I)$ of the intervals $I \in \mathcal{I}_u$. Next a new balanced tree is built to take the place of T_u, using the standard recursive procedure to create interval trees. That is, it takes as input a set of intervals $\mathcal{I}'$ (initially set to $\mathcal{I}_u$) and their sorted endpoints. The procedure creates a new root vertex w of the current tree, and sets l_w to the median of all endpoints of $\mathcal{I}'$. It then identifies the set $\mathcal{S}_w$ containing all intervals of $\mathcal{I}'$ that intersect l_w. The left and right subtrees are then recursively built for the subsets of $\mathcal{I}'$ of all intervals to the left of l_w and to the right of l_w, respectively. In case $\mathcal{I}' = \emptyset$, a leaf is created and the recursion terminates. Note that the number of endpoints of value less than the median is at most $|\mathcal{I}'|$, as there are $2|\mathcal{I}'|$ endpoints. Since each interval has two endpoints, the left subtree will contain at most $|\mathcal{I}'|/2$ intervals, and analogously this is also true for the right subtree. Therefore this results in a new tree T_u for which $\max\{n_w^-, n_w^+\} \leq n_w/2$ for every node w of T_u, i.e., this tree is perfectly balanced.

To update the permutations we need some definitions (cf. Figure 5). For any node u of T, let $\text{beg}_u = \min\{\text{beg}(I) \mid I \in \mathcal{I}_u\}$ be the left-most point of any interval stored in T_u, and accordingly let $\text{end}_u = \max\{\text{end}(I) \mid I \in \mathcal{I}_u\}$ be the right-most point. We then define the two sets $\mathcal{L}_u = \{I \in \mathcal{I} \mid \text{beg}(I) \leq \text{beg}_u < \text{end}(I)\} \setminus \mathcal{I}_u$ and $\mathcal{R}_u = \{I \in \mathcal{I} \mid \text{beg}(I) \leq \text{end}_u < \text{end}(I)\} \setminus \mathcal{I}_u$ of intervals not stored in T_u but containing beg_u or end_u, respectively. Note that each interval in $\mathcal{L}_u$ or $\mathcal{R}_u$ must be stored in some internal node $w \notin \{r, u\}$ of P_u (meaning that it is contained in $\mathcal{S}_w$), as $\mathcal{S}_w$ is non-empty and separates $\mathcal{I}_u$ from the intervals stored in the (left or right) subtree of T_w not containing u. Let also L and R be the set of edges of P_u that cross the boundary defined by end_u in the sense that $xy \in L$ if x is the parent of y and $l_x \geq \text{end}_u > l_y$, and $xy \in R$ if x is the parent of y and $l_x < \text{end}_u \leq l_y$.

The algorithm performs the following steps after I_{new} was added to the set $\mathcal{S}_v$.

1. If there is a node $w \neq r$ on P_v for which $\max\{n_w^-, n_w^+\} > \lceil \alpha n_w \rceil$, then let w be the closest such node to the root r. Rebuild the subtree T_w to obtain a new perfectly balanced subtree T_w. In this case set $u = w$ in the following, while otherwise $u = v$.
2. First retrieve beg_u and end_u, and then $\mathcal{L}_u$ and $\mathcal{R}_u$ together with all colors of intervals in $\mathcal{L}_u$ and $\mathcal{R}_u$ using the permutations stored on the edges of P_u.
3. Starting with the current coloring of $\mathcal{L}_u$, use the greedy algorithm to color $\mathcal{I}_u \cup \mathcal{R}_u$ with at most k colors. As the intervals in $\mathcal{R}_u$ form a clique (they all contain end_u), there is a permutation $\mu \in \mathfrak{S}_k$ mapping the old colors of $\mathcal{R}_u$ to its new colors.
4. The permutations stored on edges e of P_u and T_u are updated to encode the new colors for the intervals in $\mathcal{I}_u \cup \mathcal{R}_u$ as follows. Let σ_e and σ'_e be the composite permutations along the path from the root to edge e before and after the update, respectively.
   a. For any edge e of P_u that is neither in L nor in R, the permutation τ_e remains unchanged.
   b. For any $e \in L$ the permutation τ_e is chosen such that $\sigma'_e = \sigma_e$.
   c. For any $e \in R$ the permutation τ_e is chosen such that $\sigma'_e = \sigma_e \circ \mu$ for the permutation μ of step 3.
   d. Permutations τ_e for edges e of T_u are simply chosen so that the σ_e induce the new colors of $\mathcal{I}_u$.

The proof of correctness and runtime analysis of this data structure can be found in Appendix A.

References

1 Patrick Baier, Bartłomiej Bosek, and Piotr Micek. On-line chain partitioning of up-growing interval orders. *Order*, 24(1):1–13, 2007. doi:10.1007/s11083-006-9050-0.

2 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Algorithms and data structures*, volume 10389 of *Lecture Notes in Comput. Sci.*, pages 97–108. Springer, Cham, 2017. doi:10.1007/978-3-319-62127-2_9.

3 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–959. SIAM, Philadelphia, PA, 2018. doi:10.1137/1.9781611975031.61.

4 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. SIAM, Philadelphia, PA, 2018. doi:10.1137/1.9781611975031.1.

5 Bartłomiej Bosek, Stefan Felsner, Kamil Kloch, Tomasz Krawczyk, Grzegorz Matecki, and Piotr Micek. On-line chain partitions of orders: a survey. *Order*, 29(1):49–73, 2012. doi:10.1007/s11083-011-9197-1.

6 Bartłomiej Bosek, H. A. Kierstead, Tomasz Krawczyk, Grzegorz Matecki, and Matthew E. Smith. An easy subexponential bound for online chain partitioning. *Electron. J. Combin.*, 25(2):Paper No. 2.28, 23, 2018. doi:10.37236/7231.

7 Bartłomiej Bosek and Tomasz Krawczyk. A subexponential upper bound for the on-line chain partitioning problem. *Combinatorica*, 35(1):1–38, 2015. doi:10.1007/s00493-014-2908-7.

8 Bartłomiej Bosek and Tomasz Krawczyk. On-line partitioning of width w posets into $w^{O(\log \log w)}$ chains. *CoRR*, arXiv:1810.00270, 2018. arXiv:1810.00270.

9 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014*, pages 384–393. IEEE Computer Soc., Los Alamitos, CA, 2014. doi:10.1109/FOCS.2014.48.

10 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. Shortest augmenting paths for online matchings on trees. *Theory Comput. Syst.*, 62(2):337–348, 2018. doi:10.1007/s00224-017-9838-x.

11 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In *LATIN 2018: Theoretical informatics*, volume 10807 of *Lecture Notes in Comput. Sci.*, pages 201–216. Springer, Cham, 2018. doi:10.1007/978-3-319-77404-6_1.

12 Marek Chrobak and Maciej Ślusarek. On some packing problem related to dynamic storage allocation. *RAIRO Inform. Théor. Appl.*, 22(4):487–499, 1988.

13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

14 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to Adwords. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 982–991. ACM, New York, 2008.

15 Magnús M. Halldórsson and Christian Konrad. Improved distributed algorithms for coloring interval graphs with application to multicoloring trees. In *Structural information and communication complexity*, volume 10641 of *Lecture Notes in Comput. Sci.*, pages 247–262. Springer, Cham, 2017. doi:10.1007/978-3-319-72050-0_15.

16 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.

17 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for MaxClique, chromatic number and Min-3Lin-Deletion. In *Automata, languages and programming. Part I*, volume 4051 of *Lecture Notes in Comput. Sci.*, pages 226–237. Springer, Berlin, 2006. doi:10.1007/11786986_21.

18 Henry A. Kierstead. An effective version of Dilworth's theorem. *Trans. Amer. Math. Soc.*, 268(1):63–77, 1981. doi:10.2307/1998337.

19 Henry A. Kierstead. Recursive ordered sets. In *Combinatorics and ordered sets (Arcata, Calif., 1985)*, volume 57 of *Contemp. Math.*, pages 75–102. Amer. Math. Soc., Providence, RI, 1986. doi:10.1090/conm/057/856233.

20 Henry A. Kierstead and William T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congr. Numer.*, 33:143–153, 1981.

21 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: efficient dynamic algorithms for the Steiner tree. In *STOC'15—Proceedings of the 2015 ACM Symposium on Theory of Computing*, pages 11–20. ACM, New York, 2015.

22 Dániel Marx. Parameterized coloring problems on chordal graphs. *Theoret. Comput. Sci.*, 351(3):407–424, 2006. doi:10.1016/j.tcs.2005.10.008.

23 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *Automata, languages, and programming. Part I*, volume 7391 of *Lecture Notes in Comput. Sci.*, pages 689–700. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_58.

24 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. AdWords and generalized online matching. *J. ACM*, 54(5):Art. 22, 19, 2007. doi:10.1145/1284320.1284321.

25 Cara Monical and Forrest Stonedahl. Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 469–476, New York, NY, USA, 2014. ACM. doi:10.1145/2576768.2598233.

26 Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Inform. Process. Lett.*, 37(1):21–25, 1991. doi:10.1016/0020-0190(91)90245-D.

27 Carlos A. S. Oliveira and Panos M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.*, 32(8):1953–1981, 2005. doi:10.1016/j.cor.2003.12.007.

28 Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *2011 International Conference on Communications, Computing and Control Applications, CCCA 2011*, pages 1–5, 2011. doi:10.1109/CCCA.2011.6031437.

29 Jean-Jacques Pansiot and Dominique Grad. On routes and multicast trees in the internet. *SIGCOMM Comput. Commun. Rev.*, 28(1):41–50, 1998. doi:10.1145/280549.280555.

30 Davy Preuveneers and Yolande Berbers. ACODYGRA: An agent algorithm for coloring dynamic graphs. In *6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 381–390, 2004. URL: https://lirias.kuleuven.be/1654548.

31 Fred S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968)*, pages 139–146. Academic Press, New York, 1969.

32 Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 30:1–30:12, Piscataway, NJ, USA, 2016. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=3014904.3014945.

33 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th European Symposium on Algorithms*, volume 112 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 72, 16. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.

34 Mario Valencia-Pabon. Revisiting Tucker's algorithm to color circular arc graphs. *SIAM J. Comput.*, 32(4):1067–1072, 2003. doi:10.1137/S0097539700382157.

35 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Effective and efficient dynamic graph coloring. *Proc. VLDB Endow.*, 11(3):338–351, 2017. doi:10.14778/3157794.3157802.

36 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, New York, 2006. doi:10.1145/1132516.1132612.

A Correctness and runtime of the balanced interval tree

Recall that the interval tree T has the following properties.

1. If $\mathcal{I}_v = \emptyset$ then v is a leaf of T with undefined value l_v and empty set $\mathcal{S}_v$.
2. Otherwise, T_v has a defined value $l_v \in \mathbb{R}$ and two child nodes x and y in T, for which the (defined) values l_u of all nodes u of T_x are smaller than l_v, while the (defined) values l_u of all nodes u of T_y are larger than l_v. The trees T_x and T_y are called the *left* and *right subtree* of T_v, respectively.
3. The set $\mathcal{S}_v = \{I \in \mathcal{I}_v \mid \mathrm{beg}(I) \leq l_v \leq \mathrm{end}(I)\}$ contains all intervals of $\mathcal{I}_v$ intersecting l_v. The left and right subtrees T_x and T_y of T_v recursively store all intervals in $\mathcal{I}_x = \{I \in \mathcal{I}_v \mid \mathrm{end}(I) < l_v\}$ and $\mathcal{I}_y = \{I \in \mathcal{I}_v \mid \mathrm{beg}(I) > l_v\}$, respectively.

To insert a new interval I_{new}, we follow the search path for I_{new} starting from the root. As soon as we encounter a node v in T such that I_{new} belongs to the set $\mathcal{S}_v$ (because $l_v \in I_{\mathrm{new}}$), we add I_{new} to $\mathcal{S}_v$. The index i of the new interval I_{new} in $\mathcal{S}_v$ is the highest available, i.e., $i = |\mathcal{S}_v|$ when $I_{\mathrm{new}} \in \mathcal{S}_v$. Additionally, we increase the variables n_u along the nodes u of the path P_v from v to the root of T by one each, to count the new interval I_{new} in the subtrees T_u. If no node v for which $l_v \in I_{\mathrm{new}}$ is found, let w be the leaf of T at the end of the search

path P_w for I_{new}. We set $l_w = \text{beg}(I_{\text{new}})$, and add I_{new} as the only interval in the set $\mathcal{S}_w$. We also create two new leaves and set them as the new left and right subtrees of w. Again, we increase the variables n_u along the nodes u of P_w.

In order to rebalance a subtree T_u rooted at a node u we use the following standard procedure. It takes as input a set of intervals $\mathcal{I}'$ (initially set to $\mathcal{I}_u$) and their sorted endpoints. The procedure creates a new root vertex w of the current tree, and sets l_w to the median of all endpoints of $\mathcal{I}'$. It then identifies the set $\mathcal{S}_w$ containing all intervals of $\mathcal{I}'$ that intersect l_w. The left and right subtrees are then recursively built for the subsets of $\mathcal{I}'$ of all intervals to the left of l_w and to the right of l_w, respectively. In case $\mathcal{I}' = \emptyset$, a leaf is created and the recursion terminates. Note that the number of endpoints of value less than the median is at most $|\mathcal{I}'|$, as there are $2|\mathcal{I}'|$ endpoints. Since each interval has two endpoints, the left subtree will contain at most $|\mathcal{I}'|/2$ intervals, and analogously this is also true for the right subtree. Therefore this results in a new tree T_u for which $\max\{n_w^-, n_w^+\} \leq n_w/2$ for every node w of T_u, i.e., this tree is perfectly balanced.

We will now prove the correctness of the algorithm, and later turn to analyzing its amortized runtime.

▶ **Lemma 23.** *The algorithm maintains an α-balanced interval tree T for $\mathcal{I}$ for which the permutations stored on the edges induce a proper k-coloring of $\mathcal{I}$, if k is the chromatic number of $\mathcal{I}$.*

Proof. That the algorithm maintains an α-balanced tree is clear from step 1 and the procedure to rebalance subtrees. That it is an interval tree follows from the fact that adding I_{new} to the first node along the search path for I_{new} of T will store it in the highest node v of T for which I_{new} contains l_v, as required. Furthermore, this property is also maintained when rebalancing a subtree T_w. As no interval will ever be added to the set $\mathcal{S}_r$ of the root r (assuming $\text{beg}(I) > 0$ for all $I \in \mathcal{I}$) and since $T_r = T$ will never be considered for rebalancing, we maintain the invariant that $l_r = 0$ and all of $\mathcal{I}$ is stored in the right subtree of T. Finally, when adding a new interval to a node or rebalancing a subtree, any node v with $\mathcal{S}_v = \emptyset$ will be a leaf of T, except for the root.

To prove that the coloring induced by the permutations of T's edges is a proper k-coloring, we proceed by induction. The base case is when the tree does not store any intervals, which is trivial. Now consider one step of the algorithm in which some interval I_{new} is added to T, and let u be the node operated on during the execution, i.e., $u = w$ if w is rebalanced and then recolored, or $u = v$ if no subtree needs to be rebalanced and I_{new} is added to $\mathcal{S}_v$. The main observation is that $\mathcal{L}_u$ and $\mathcal{R}_u$ form separators. More concretely, let $\mathcal{L}_u^- = \{I \in \mathcal{I} \mid \text{end}(I) < \text{beg}_u\}$ and $\mathcal{R}_u^+ = \{I \in \mathcal{I} \mid \text{beg}(I) > \text{end}_u\}$, and note that $\mathcal{I}$ is partitioned into $\mathcal{L}_u^-$, $\mathcal{I}_u$, $\mathcal{R}_u^+$, and $\mathcal{L}_u \cup \mathcal{R}_u$ ($\mathcal{L}_u$ and $\mathcal{R}_u$ may share some intervals). For any $I \in \mathcal{I}_u \cup (\mathcal{R}_u \setminus \mathcal{L}_u) \cup \mathcal{R}_u^+$ we have $\text{beg}_u \leq \text{beg}(I)$, while for any $I \in \mathcal{I}_u \cup (\mathcal{L}_u \setminus \mathcal{R}_u) \cup \mathcal{L}_u^-$ we have $\text{end}_u \geq \text{end}(I)$. This means that $\mathcal{L}_u$ separates $\mathcal{L}_u^-$ from $\mathcal{I}_u \cup (\mathcal{R}_u \setminus \mathcal{L}_u) \cup \mathcal{R}_u^+$, and similarly $\mathcal{R}_u$ separates $\mathcal{R}_u^+$ from $\mathcal{I}_u \cup (\mathcal{L}_u \setminus \mathcal{R}_u) \cup \mathcal{L}_u^-$. Thus a k-coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u$ and a k-colouring of $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ together form a k-coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$, if the two given colorings agree on the colors of the separator $\mathcal{L}_u$. Furthermore, a k-coloring of $\mathcal{R}_u \cup \mathcal{R}_u^+$ together with a k-coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ forms a k-coloring of $\mathcal{I}$ if the two given colorings agree on the colors of $\mathcal{R}_u$. Hence if we separately prove that the permutations induce a proper k-coloring for each of the three sets $\mathcal{L}_u^- \cup \mathcal{L}_u$, $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$, and $\mathcal{R}_u \cup \mathcal{R}_u^+$, then $\mathcal{I}$ is properly k-colored.

Let I be any interval from $\mathcal{L}_u^- \cup \mathcal{L}_u$, w be the node of T storing I, and e be the last edge of P_w, i.e., which is farthest from the root r of T. If $I \in \mathcal{L}_u^-$, then no edge of P_w can be from T_u, by the above observation that $\mathcal{L}_u$ separates $\mathcal{L}_u^-$ from $\mathcal{I}_u$. The same is true for P_w if

$I \in \mathcal{L}_u$, since $\mathcal{L}_u$ contains no interval from $\mathcal{I}_u$ by definition. In case no edge of P_w belongs to L or R, according to step 4 every edge f of P_w stores the same permutation τ_f before and after I_{new} was added. This implies $\sigma'_e = \sigma_e$ for the respective composite permutations σ'_e and σ_e along P_w before and after the update. Otherwise, let xy be the farthest edge of P_w from the root r that belongs to $L \cup R$, where x is the parent of y. If $xy \in L$ then $\sigma'_{xy} = \sigma_{xy}$ by step 4, while τ_f is unchanged on any edge f of P_w that is farther than y from the root. Thus if π_{yw} is the composite permutation along P_w from y to w (with π_{yw} being the identity permutation in the trivial case when $y = w$) we obtain $\sigma'_e = \pi_{yw} \circ \sigma'_{xy} = \pi_{yw} \circ \sigma_{xy} = \sigma_e$. For the last case $xy \in R$, note that since T is a search tree, it must be that $l_z \geq \text{end}_u$ for any node z after y on the search path P_w: otherwise some edge after y on P_w would cross the boundary end_u, i.e., it would be in L, contradicting the fact that xy is the last edge of P_w that is in $L \cup R$. Hence for $z = w$ we obtain $\text{end}(I) \geq l_w \geq \text{end}_u$. But as $I \in \mathcal{L}_u \cup \mathcal{L}_u^-$ we also get $\text{beg}(I) \leq \text{beg}_u \leq \text{end}_u$ and so $I \in \mathcal{L}_u \cap \mathcal{R}_u$. Therefore the permutation μ of step 3 maps the color of I to itself, and if I is the ith interval of $\mathcal{S}_w$, by our choice of τ_{xy} in step 4 we get $\sigma'_e(i) = (\pi_{yw} \circ \sigma'_{xy})(i) = (\pi_{yw} \circ \sigma_{xy} \circ \mu)(i) = (\sigma_e \circ \mu)(i) = \sigma_e(i)$. In conclusion, every interval of $\mathcal{L}_u \cup \mathcal{L}_u^-$ has the same color before and after inserting I_{new}, and thus $\mathcal{L}_u \cup \mathcal{L}_u^-$ is properly k-colored by induction.

Next consider an interval I from $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$. We already know that if $I \in \mathcal{L}_u$ then it keeps its color from before the update, i.e., the permutations on T's edges induce the same color of I that the greedy algorithm assigns to it. By step 4, any interval of $\mathcal{I}_u$ (including I_{new}) also obtains the colors assigned to it by the greedy algorithm. If $I \in \mathcal{R}_u \setminus \mathcal{L}_u$ then $\text{beg}(I) > \text{beg}_u$ and $I \notin \mathcal{I}_u$. Thus the node w of T storing I is not in T_u. Furthermore, following the search path P_w from the root r must end in a node w for which $l_w > \text{end}_u$, if w is not in T_u and $l_w \geq \text{beg}(I) > \text{beg}_u$. As a consequence, P_w has some edge of R, since $l_r = 0$ and thus following the search path P_w there must be some edge of P_w that crosses end_u in order to reach w. Furthermore, if xy is the edge of P_w that lies in R and is farthest from the root, where x is the parent of y, then no edge of P_w between y and w can belong to L, as such an edge would cross over to the left of end_u but $l_w > \text{end}_u$. Hence by step 4 all edges f of P_w between y and w maintain their permutations τ_f during the update. Let e be the last edge of P_w and let π_{yw} denote the composite permutation along P_w from y to w, which is the identity permutation if $y = w$. By the choice of τ_{xy} in step 4, we have $\sigma'_e = \pi_{yw} \circ \sigma'_{xy} = \pi_{yw} \circ \sigma_{xy} \circ \mu = \sigma_e \circ \mu$. Thus the colors of all intervals of $\mathcal{R}_u \setminus \mathcal{L}_u$ are permuted according to μ, which by definition of μ in step 3 then means that all of $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ is colored according to the greedy algorithm. This implies a proper k-coloring of this set, due to the correctness of the greedy algorithm.

For the last set $\mathcal{R}_u \cup \mathcal{R}_u^+$ we already know that any interval I from $\mathcal{R}_u$ is colored according to the permutation $\sigma_e \circ \mu$, if e is the last edge of the path P_w to the node w storing I (we argued this separately for $I \in \mathcal{R}_u \setminus \mathcal{L}_u$ and $I \in \mathcal{R}_u \cap \mathcal{L}_u$ above). This is also true for any $I \in \mathcal{R}_u^+$, since the premise is the same as for intervals from $\mathcal{R}_u \setminus \mathcal{L}_u$: we have $\text{beg}(I) > \text{end}_u \geq \text{beg}_u$ and $I \notin \mathcal{I}_u$ as $\mathcal{R}_u$ separates $\mathcal{I}_u$ from $\mathcal{R}_u^+$. Therefore the colors of intervals in $\mathcal{R}_u \cup \mathcal{R}_u^+$ are permuted by μ relative to the colors induced by the permutations of the edges of T before the update. Hence $\mathcal{R}_u \cup \mathcal{R}_u^+$ is properly k-colored by induction, which concludes the proof. ◀

In order to bound the amortized runtime of one step when adding an interval I_{new} to the search tree T, we first determine the actual runtime.

▶ **Lemma 24.** *Let u be the node of T for which the update algorithm is run, let p_u be the number of nodes on the path P_u from the root of T to u, and let t_u be the number of nodes of the subtree T_u of T rooted at u. Then the update algorithm takes $\mathcal{O}(k(t_u + p_u)\log n)$ time.*

Proof. Finding the node v in which to store I_{new} and a node w on P_v for which T_w needs to be rebalanced is linear in the height of T, and can thus be done in $\mathcal{O}(\log n)$ time as T is α-balanced. If $n_w = |\mathcal{I}_w|$ denotes the number of intervals stored in T_w, it is known that rebalancing T_w can be done in $\mathcal{O}(n_w \log n_w)$ time [13] for step 1. Next we set $u = w$ or $u = v$ depending on whether some tree was rebalanced. As $|\mathcal{S}_x| \leq k$ for every node x of T, we have $n_u \leq kt_u$, and the time to rebalance can be bounded by $\mathcal{O}(kt_u \log n)$.

Retrieving beg_u and end_u in step 2 needs linear time in the height of the tree T_u, i.e., it can be done in $\mathcal{O}(\log n)$ time. If the number of nodes of P_u is denoted by p_u then the number of intervals stored in nodes of P_u is at most kp_u, by the observation that each set stored in a node forms a clique in a k-colorable graph. Thus retrieving $\mathcal{L}_u$ and $\mathcal{R}_u$ together with their colors takes $\mathcal{O}(kp_u)$ time if traversing P_u bottom up towards the root and in each step computing the composite permutation σ_e for each edge e of P_u from the permutation $\sigma_{e'}$ of the previous edge e'.

For step 3, also the set $\mathcal{I}_u$ needs to be retrieved, which can be done in $\mathcal{O}(n_u)$ time given u. The runtime of the greedy algorithm [26] to color $\mathcal{I}_u \cup \mathcal{R}_u$ given the colors of $\mathcal{L}_u$ is $\mathcal{O}((n_u + k)\log(n_u + k))$ as both $\mathcal{L}_u$ and $\mathcal{R}_u$ form a clique in a k-colorable graph. Finding the permutation μ takes $\mathcal{O}(k)$ time. As $n_u \leq kt_u$, the time spent for step 3 can be bounded by $\mathcal{O}(k(t_u + p_u)\log n)$.

To update the permutations on edges e of P_u and T_u in step 4, the algorithm can traverse P_u and T_u bottom up towards the root of T in order to first compute the composite permutations σ_e. Then it can traverse P_u and T_u top down from the root in order to compute σ'_e and τ_e given σ'_f of the parent edge f of e, as τ_e is uniquely defined by σ'_f in all four cases (a) to (d). Thus this takes $\mathcal{O}(k(t_u + p_u))$ time, which concludes the proof. ◀

To obtain the amortized runtime we give a proof using the potential function method [13].

Proof of Theorem 22. As for Lemma 24, let t_u be the number of nodes in T_u and p_u be the number of nodes of P_u. Given a potential function Φ, the amortized runtime is given by the sum of the actual runtime per update, which is $\mathcal{O}(k(t_u + p_u)\log n)$ by Lemma 24, and Δ_Φ, which is the difference between the potential after and before adding an interval I_{new} to T.

To define the potential, let $h = \mathcal{O}(\log n)$ be the maximum height of the α-balanced tree T, and for any node u let $m_u = \max\{n_u^-, n_u^+\}$, $s_u = |\mathcal{S}_u|$, and $a_u = \sum_{w \in V(P_u)} s_w$ be the number of intervals stored in nodes of P_u. Then define

$$\Gamma(u) = \max\left\{\frac{m_u - n_u/2}{\alpha - 1/2}, 0\right\}, \qquad \beta = 4k^2h + 2k,$$
$$\Lambda(u) = 2ks_u \cdot (kp_u - a_u), \qquad \Phi(u) = \beta \cdot \Gamma(u) + \Lambda(u).$$

Note that each node of P_u stores at most k intervals so that $a_u \leq kp_u$ and thus $\Lambda(u) \geq 0$. Hence $\Phi(u) \geq 0$ and we can define a potential function $\Phi = C \log n \cdot \sum_{u \in V(T)} \Phi(u)$, where C is the constant hidden in the $\mathcal{O}$-notation of the actual runtime according to Lemma 24. Note that the change Δ_Φ is only influenced by the addition of the new interval I_{new} into node v, and the rebuilding of a subtree T_w in step 1 of the algorithm. That is, none of the steps 2 to 4 change any of the terms of Φ.

To bound the amortized runtime, we distinguish the cases when some subtree T_w is rebalanced and when not. For the former case, let us begin by determining Δ_Γ, i.e., the change in $\sum_{u \in V(T)} \Gamma(u)$ during an update. After I_{new} is inserted into v we have $m_w > \lceil \alpha n_w \rceil$ at the node w before T_w is being rebuilt in step 1. This means that before inserting I_{new} we had $m_w \geq \lceil \alpha n_w \rceil \geq \alpha n_w$, and thus $\Gamma(w) \geq n_w$. After rebuilding T_w it is perfectly balanced and we have $m_x \leq n_x/2$ for every node x of T_w, so that now $\Gamma(x) = 0$. In particular,

during the update, $\Gamma(w)$ decreased from at least n_w to 0. Note that compared to before the update, T_w may contain a different set of nodes after it is rebuilt. Nevertheless, the sum $\sum_{x\in V(T_w)}\Gamma(x)$ will decrease during the update, as afterwards $\Gamma(x)=0$ for every node x of T_w. In the remaining tree T the value $\Gamma(u)$ can only increase by at most 1 for nodes u along the path P_w. Hence we get that $\Delta_\Gamma \le p_w - n_w$.

We now determine Δ_Λ, i.e., the change in $\sum_{u\in V(T)}\Lambda(u)$ during an update, when a tree T_w is rebuilt in step 1. Note that $\Lambda(u)$ does not change for any node u of T that is not contained in T_w. As observed above, compared to before, T_w may contain a different set of nodes after it is rebuilt. However, the set of intervals $\mathcal{I}_w$ stored in T_w remains the same. We therefore consider the contribution of each interval in $\mathcal{S}_x$ towards $\Lambda(x)$ for any node x of T_w, before and after the update. Let us define $\Lambda'(u) = 2k(kp_u - a_u)$ for every node u, so that the contribution of every interval $I\in\mathcal{S}_u$ to $\Lambda(u)$ is $\Lambda'(u)$. For any node x of T_w, by definition of a_x and p_x we obtain

$$\Lambda'(x) = \sum_{y\in V(P_x)} 2k(k-s_y) = \Lambda'(w') + \sum_{y\in V(Q_x)} 2k(k-s_y),$$

where $Q_x\subseteq P_w$ is the path from x to w and w' is the parent of w (which exists since $w\neq r$). We may bound $\Lambda'(x)$ from below by $\Lambda'(w')$, and from above by $\Lambda'(w')+2k^2p_x$. As $\Lambda'(w')$ is unchanged during the update, the contribution of each interval $I\in\mathcal{I}_w$ different from I_{new} changes by at most $2k^2p_x$, where x is the node of T_w storing I after the update. As I_{new} was not present in T_w before, its contribution adds $\Lambda'(w')+2k^2p_x$ for the node x storing I_{new} after T_w is rebuilt. We may bound p_x by the height h of T after the update for any node x, and $\Lambda'(w')$ is at most $2k^2h$. Thus we get $\Delta_\Lambda \le n_w\cdot 2k^2h + \Lambda'(w') \le 2k^2h(n_w+1)$, where n_w also counts I_{new} in $\mathcal{I}_w$.

Since $\beta = 4k^2h+2k$ and $n_w\ge 1$, as a consequence of the above we obtain

$$\begin{aligned}\Delta_\Phi = C(\beta\Delta_\Gamma+\Delta_\Lambda)\log n &\le C\left(\beta(p_w-n_w)+2k^2h(n_w+1)\right)\log n\\ &\le C\left(\beta p_w - (4k^2h+2k)n_w + 4k^2hn_w\right)\log n \le C\left(\beta p_w - 2kn_w\right)\log n.\end{aligned}$$

We have that $t_w\le 2n_w$, since we maintain the invariant that for every node u except the root of T, if $\mathcal{S}_u=\emptyset$ then u is a leaf of the complete binary tree T. Hence the actual runtime according to Lemma 24 can be upper bounded by $Ck(2n_w+p_w)\log n$, which means that the amortized runtime is $C(\beta+k)p_w\log n = \mathcal{O}(k^2\log^3 n)$ in case a subtree T_w is rebalanced in step 1, since $\beta=\mathcal{O}(k^2\log n)$ and $p_w\le h=\mathcal{O}(\log n)$.

We now turn to the case when no subtree is rebalanced in step 1 and the only change of Φ is due to I_{new} being added to a node v of T. Note that $\Gamma(u)$ only changes along the nodes u of path P_v, where m_u may increase by 1. Thus $\Delta_\Gamma\le\frac{p_v}{\alpha-1/2}$. To bound Δ_Λ we consider two cases: either v was an existing internal node of T, or v was a leaf and is then converted into an internal node. In the first case, a_u of every node u of T_v increases by 1 due to the new interval I_{new} stored in the ancestor v of u, and so $\Lambda(u)$ decreases by $2ks_u$. At the same time, $\Lambda(u)$ is unchanged for any node u not in T_v, and we get $\Delta_\Lambda\le -2kn_v$. Hence in this case $\Delta_\Phi \le C\left(\frac{\beta p_v}{\alpha-1/2}-2kn_v\right)\log n$. As we have seen the actual runtime can be upper bounded by $Ck(2n_v+p_v)\log n$, and thus the amortized runtime becomes $Cp_v\left(\frac{\beta}{\alpha-1/2}+k\right)\log n = \mathcal{O}(k^2\log^3 n)$.

Finally, if I_{new} is added to a leaf v of T, then v is converted into an internal node by adding two leaves to v. For any leaf x, $\Lambda(x)=0$ as $s_x=0$, and thus these new nodes do not contribute to Δ_Λ. However v was formerly a leaf and now contains I_{new}, so that its contribution to Δ_Λ is $2k(kp_v-a_v)\le 2k^2p_v$. Hence we get $\Delta_\Phi\le C\left(\frac{\beta p_v}{\alpha-1/2}+2k^2p_v\right)\log n = \mathcal{O}(k^2\log^3 n)$. The subtree T_v only stores I_{new} so that $n_v=1$ and the actual runtime is $Ck(2n_v+p_v)\log n = \mathcal{O}(k\log^2 n)$. Thus in this case we also obtain an amortized runtime of $\mathcal{O}(k^2\log^3 n)$, which concludes the proof. ◀