J Comb Optim (2012) 24:1–14
DOI 10.1007/s10878-010-9290-1

# Max-coloring paths: tight bounds and extensions

**Telikepalli Kavitha · Julián Mestre**

Published online: 28 January 2010


**Abstract** The max-coloring problem is to compute a legal coloring of the vertices of a graph $G = (V, E)$ with vertex weights $w$ such that $\sum_{i=1}^{k} \max_{v \in C_i} w(v_i)$ is minimized, where $C_1, \ldots, C_k$ are the various color classes. For general graphs, max-coloring is as hard as the classical vertex coloring problem, a special case of the former where vertices have unit weight. In fact, in some cases it can even be harder: for example, no polynomial time algorithm is known for max-coloring trees. In this paper we consider the problem of max-coloring paths and its generalization, max-coloring *skinny* trees, a broad class of trees that includes paths and spiders. For these graphs, we show that max-coloring can be solved in time $O(|V| +$ time for sorting the vertex weights). When vertex weights are real numbers, we show a matching lower bound of $\Omega(|V| \log |V|)$ in the algebraic computation tree model.

**Keywords** Coloring · Weighted graph · Approximation algorithm · Lower bound

## 1 Introduction

The *max-coloring problem* takes as input a graph $G = (V, E)$ with a weight function $w : V \to \mathbb{N}$. The problem is to compute a legal coloring of $V$ such that

T. Kavitha's work done as part of the DST-MPG partner group "Efficient Graph Algorithms" at IISc Bangalore.

T. Kavitha (✉)
Indian Institute of Science, Bangalore, India
e-mail: kavitha@csa.iisc.ernet.in

J. Mestre
Max-Plank-Institut für Informatik, Saarbrücken, Germany
e-mail: jmestre@mpi-inf.mpg.de

$\sum_{i=1}^{k} \max_{v \in C_i} w(v_i)$ is minimized, where $C_1, \ldots, C_k$ are the various color classes. When all the weights are 1, this problem reduces to the classical problem of determining a legal coloring of $V$ using the least number of colors.

The max-coloring problem arises in the problem of partitioning a set of $n$ jobs into batches, where all jobs in a batch start at the same time, and the batch is completed when its last job finishes. The length of a batch is the length of the longest job in this batch. The graph $G$ has each job as a vertex and each edge in $G$ captures a conflict between a pair of jobs, that is, these jobs cannot be processed in the same batch. Thus a valid schedule is a legal coloring of $G$, since each color class consists of jobs that are conflict-free and can be processed together as one batch. The goal is to minimize the makespan of the batch schedule, which is the same as computing a max-coloring of $G$.

Since classical coloring is a special case of max-coloring (when all weights are 1), the problem is hard for general graphs. Unlike the classical coloring problem, max-coloring bipartite graphs is APX-hard (Pemmaraju and Raman 2005). Even max-coloring of trees is non-trivial: there is an exact algorithm with running time $O(n^{\log n})$ where $n$ is the number of vertices, but only a PTAS is known if we restrict ourselves to polynomial time algorithms (Pemmaraju and Raman 2005). Constant factor approximation algorithms and NP-hardness results for various classes of perfect graphs were given in Epstein and Levin (2007).

The focus of this paper is on max-coloring paths and the more general problem of max-coloring *skinny trees*, these are trees where the set of vertices of degree at least 3 form an independent set. It is easy to show that at most 3 color classes are needed to optimally max-color a path (see, e.g., Brucker 2004). This simple observation immediately yields an $O(n^4)$ time algorithm (Guan and Zhu 1997). This was subsequently improved to $O(n^2)$ by Escoffier et al. (2006). More recently, Halldórsson and Shachnai (2008) gave a recursive $O(n \log n)$ time algorithm.

In this paper we show algorithms for max-coloring paths and skinny trees that take time $O(n + S(n))$, where $S(n)$ is the time it takes to sort the vertex weights. Since the vertex weights are integers, when they are polynomially bounded, our algorithms take $O(n)$ time using radix sort to sort the vertex weights. More generally, using the randomized algorithm of Han and Thorup for integer sorting (Han and Thorup 2002) makes the expected running time of our algorithms $O(n\sqrt{\log \log n})$. Our algorithms can be easily adapted to give $1 + \epsilon$ approximation of the optimal max-coloring in $O(n + 1/\epsilon)$ time.

We next consider the problem of max-coloring a path with *real* vertex weights and show a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model. Thus the complexity of max-coloring a path with real vertex weights is $\Theta(n \log n)$.

### 1.1 Applications of max-coloring

As stated earlier, the max-coloring problem occurs in batch scheduling where the edges in the graph capture conflicts between pairs of jobs that cannot be processed in the same batch. Such problems occur naturally in metropolitan networks and in distributed computing as described in Halldórsson and Shachnai (2008). In metropolitan networks during the transmission of real-time messages, each sender node gets a set

of fixed length slots in which he fills up messages and transmits to the receiver. The same slots can be given to multiple senders to improve transmission times (Han et al. 1995). A batch consists of messages that use the same set of slots. The problem of minimizing the number of slots used to transmit all the messages yields an instance of the max-coloring problem.

In distributed operating systems the scheduler identifies processes that do not use the same resources into a batch and executes them simultaneously on several processors until all the jobs in the batch complete (Tanenbaum 1995; Liu and Beck 2006). The problem of minimizing the total completion time yields an instance of max-coloring.

*Organization of the paper* Although our result for skinny trees implies our result for paths, we first present our algorithm for max-coloring paths as this algorithm and analysis are simpler. This is done in Sect. 2. In Sect. 3 we present our algorithm to max-color skinny trees. Section 4 contains our approximation algorithm for this problem. Section 5 contains our lower bound result.

## 2 Max-coloring a path in time for sorting the vertex weights

Our input is a path $(v_1, v_2, \ldots, v_n)$ with a weight function $w : \{v_1, \ldots, v_n\} \to \mathbb{N}$. Since the degree of each vertex is at most 2, it is immediate that optimally max-coloring a path requires at most three colors: call these colors RED, BLUE, and GREEN. If a fourth color was used on a vertex $v$, then one of RED, BLUE, and GREEN is missing from $v$'s neighbors and so the color of $v$ can be changed to this missing color and thus the fourth color can be eliminated.

Let the weight of a color class be the weight of a heaviest vertex colored by that color. We assume throughout that the color class RED has the heaviest weight, followed by the color class BLUE and then the color class GREEN. Notice that the weight of RED equals the weight of the heaviest vertex $w_{\max} = \max_{v \in V} w(v)$. Then the problem of max-coloring a path is the problem of determining the weights of the BLUE and GREEN color classes so that their sum is minimized. Any coloring always has the form $[w_{\max}, \beta, \gamma]$, where the first coordinate denotes the weight of the RED color class, the second coordinate $\beta$ denotes the weight of the BLUE color class and the third coordinate $\gamma$ the weight of the GREEN color class.

Our algorithm can be viewed as consisting of two parts: *initialization* and the *main* part.

*Initialization* This part consists of computing a coloring $[w_{\max}, b_0, g_0]$ where $b_0$ is the least possible weight of the BLUE color class. This lower bound $b_0$ is simply $\max_{i=1}^{n-1} \min\{w(v_i), w(v_{i+1})\}$, since for no $i$ can both $v_i$ and $v_{i+1}$ be colored RED. We state this observation as Lemma 1.

**Lemma 1** *In every valid coloring* $[w_{\max}, \beta, \gamma]$ *we must have* $\beta \geq b_0$, *where* $b_0 = \max_{i=1}^{n-1} \min\{w(v_i), w(v_{i+1})\}$.

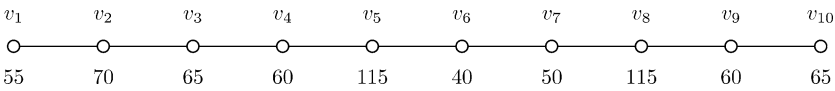


**Fig. 1** Weight of BLUE is at least 65, since both $v_2$ and $v_3$ cannot be colored RED

For example, consider the path in Fig. 1; in this example $b_0 = 65$.

If $b_0 = w_{\max}$, then the optimal solution is to 2-color the path, so let us assume that $b_0 < w_{\max}$. After computing $b_0$, we identify the nodes $u_0, \ldots, u_r$ whose weight is strictly larger than $b_0$. For each $i = 1, \ldots, r$ we find a minimum weight vertex $z_i$ among vertices in the path that lie in between $u_{i-1}$ and $u_i$. Finally we decompose the path into the sequence $\rho_0, z_1, \rho_1, \ldots, \rho_{r-1}, z_r, \rho_r$, where each $\rho_i$ is a subpath of the original path; notice that $u_i \in \rho_i$. Going back to the example of Fig. 1, the $u$-vertices are $v_2$, $v_5$, and $v_8$; the $z$-vertices are $v_4$ and $v_6$; and the path decomposition is $\langle (v_1, v_2, v_3), v_4, (v_5), v_6, (v_7, v_8, v_9, v_{10}) \rangle$.

Let $O$ be the odd-indexed vertices in the path, that is, $O = \{v_1, v_3, \ldots\}$, and let $E$ be the even-indexed vertices, that is, $E = \{v_2, v_4, \ldots\}$. For each subpath $\rho$ we denote with $o_\rho$ the maximum weight among its odd-indexed vertices and with $e_\rho$ the maximum weight among its even-indexed vertices:

$$o_\rho = \max_{v \in \rho \cap O} w(v) \quad \text{and} \quad e_\rho = \max_{v \in \rho \cap E} w(v).$$

For example, in Fig. 1, the subpath $\rho = (v_7, v_8, v_9, v_{10})$ has $o_\rho = 60$ and $e_\rho = 115$. This information about the subpaths will be useful in the main part.

*The main part* Here we determine various candidate colorings $[w_{\max}, b_j, g_j]$ in an iterative manner. These colorings are induced by the different path decompositions the algorithm keeps throughout its execution. For a given path decomposition, consider the following coloring: First color all $z$-nodes GREEN and then color each subpath with RED and BLUE as to minimize the weight of the BLUE class.

To begin with, we have the coloring $[w_{\max}, b_0, g_0]$. The main part of the algorithm deals with the following question: *by decreasing $g_0$ to some smaller value $g_i$ (which implies that $b_0$ has to be raised to some value $b_i$), can we get a lower value of $b_i + g_i$?*

Our algorithm goes through all values $g_i$ that become candidates for being the maximum weight vertex in the GREEN color class and for each such $g_i$, it determines the *minimum* value of $b_i$ that allows $[w_{\max}, b_i, g_i]$ to be a legal coloring. The coloring with the least $b_i + g_i$ value among all these candidates will be output as the optimal max-coloring.

The algorithm is formally presented in Fig. 2.

The path decompositions and triplets computed by the algorithm when run on the instance from Fig. 1 are as follows.

| $i$ | Path decomposition | Triplet |
|---|---|---|
| 0 | $\langle (v_1, v_2, v_3), v_4, (v_5), v_6, (v_7, v_8, v_9, v_{10}) \rangle$ | [115, 65, 60] |
| 1 | $\langle (v_1, v_2, v_3, v_4, v_5), v_6, (v_7, v_8, v_9, v_{10}) \rangle$ | [115, 70, 40] |
| 2 | $\langle (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}) \rangle$ | [115, 115, 0] |

And thus, the optimal solution is the coloring induced by [115, 70, 40].

**Algorithm 1** MAX-COLORING-PATH ($\langle v_1, \ldots, v_n \rangle, w$)

1. $b_0 \leftarrow \max_{1 \le j \le n-1} \min\{w(v_j), w(v_{j+1})\}$
2. Scan the path from left to right identifying nodes $u_0, \ldots, u_r$ s.t. $w(u_j) > b_0$.
3. For each $i = 1, \ldots, r$, find a vertex $z_i$ with minimum weight in $(u_{i-1}, \ldots, u_i)$.
4. Build the path decomposition $\rho_0, z_1, \rho_1, \ldots, \rho_{r-1}, z_r, \rho_r$ and compute $o_{\rho_i}$ and $e_{\rho_i}$ for each $i = 0, \ldots, r$.
5. $g_0 \leftarrow \max_{1 \le j \le r} w(z_i)$
6. $Q \leftarrow$ queue containing the vertices $z_i$ in non-increasing order of $w(z_i)$.
7. **for** $i = 1, \ldots, r$ **do**
8. $z \leftarrow$ pop first vertex from $Q$
9. let $\rho'$ and $\rho''$ be the subpaths flanking $z$ in the current path decomposition
10. replace $\rho', z, \rho''$ with a new subpath $\rho$ in the path decomposition
11. compute $e_\rho$ and $o_\rho$ from $w(z)$, $e_{\rho'}$, $o_{\rho'}$, $e_{\rho''}$, and $o_{\rho''}$
12. $b_i \leftarrow \max\{\min\{e_\rho, o_\rho\}, b_{i-1}\}$
13. $g_i \leftarrow$ weight of the next $z$-vertex in $Q$
14. **return** best coloring among the candidates $[w_{\max}, b_i, g_i]$ for $i = 0, \ldots, r$.

**Fig. 2** Pseudocode of our algorithm for max-coloring a path

## 2.1 Correctness

In this subsection we will establish the correctness of algorithm MAX-COLORING-PATH. To that end, we first prove a few technical lemmas.

**Lemma 2** *Every candidate triplet* $[w_{\max}, b_i, g_i]$ *has a corresponding valid coloring.*

*Proof* We will show how to construct a valid $[w_{\max}, b_i, g_i]$ coloring using the path decomposition the algorithm had at the end of the $i$th iteration. Consider the following coloring: First color all $z$-nodes GREEN and then color each subpath with RED and BLUE so as to minimize the weight of the BLUE class. We prove by induction on $i$ this is a $[w_{\max}, b_i, g_i]$ coloring.

Consider the coloring induced by the path decomposition right before the first iteration. We claim this is $[w_{\max}, b_0, g_0]$ or better. To see this, we color the $z$-vertices GREEN and the $u$-vertices RED. The remaining vertices have weight at most $b_0$, so coloring them with RED and BLUE gives us the desired coloring.

Assume, by induction, that the coloring induced by the path decomposition after the $i-1$st iteration is $[w_{\max}, b_{i-1}, g_{i-1}]$. Suppose the $i$th iteration pops vertex $z$ for processing from $Q$. Notice that $z$ has weight $g_{i-1}$. The coloring at the end of the $i$th iteration can only differ from the one from the previous iteration in the nodes in $\rho', z, \rho''$, which are merged into a new subpath $\rho$. Thus, the weight of the BLUE color class is the maximum of $b_{i-1}$ and the weight of the BLUE color class in the optimal 2-coloring of $\rho$, which in turn must be $\min\{e_\rho, o_\rho\}$. Similarly, the weight of the GREEN color class in the new coloring is $g_i$. Therefore, we have a $[w_{\max}, b_i, g_i]$ coloring. □

**Lemma 3** *Let* $i < r$ *be such that* $b_i < b_{i+1}$. *For every valid coloring* $[w_{\max}, \beta, \gamma]$ *if* $\beta < b_{i+1}$ *then* $\gamma \ge g_i$.

*Proof* Consider the $i+1$st iteration of the algorithm and let $z$ be the vertex removed from the queue (recall that $w(z) = g_i$). Since $b_{i+1} > b_i$ it must be the case that the

weight of the BLUE class in the optimal 2-coloring of the subpath $\rho$ created after merging $z$ with its neighboring subpaths is $b_{i+1}$. It follows that any valid coloring $[w_{\max}, \beta, \gamma]$ such that $\beta < b_{i+1}$ must color at least one vertex in $\rho$ GREEN. However, $z$ and all other nodes in $\rho$ have weight greater than or equal to $g_i$ thus $\gamma \geq g_i$. □

Everything is in place to prove the correctness of our algorithm.

**Theorem 1** *Algorithm* MAX-COLORING-PATH *outputs a feasible optimal max-coloring.*

*Proof* First, we note that from Lemma 2 it follows that the solution output is a valid coloring. To argue that it is optimal, we show that for any valid coloring $[w_{\max}, \beta, \gamma]$ there is a candidate triplet at least as good. If $\beta \geq b_r$ then our algorithm considers the candidate solution $[w_{\max}, b_r, g_r]$ where $g_r = 0$ (at the last iteration the queue is empty), so the solution output is at least as good as $[w_{\max}, \beta, \gamma]$. Otherwise, there exists $0 \leq i < r$ such that $b_i \leq \beta < b_{i+1}$ (recall that, by Lemma 1, for any valid coloring $\beta \geq b_0$). Therefore, by Lemma 3 we have $\gamma \geq g_i$ and since the algorithm considers the candidate solution $[w_{\max}, b_i, g_i]$, the solution output is at least as good as $[w_{\max}, \beta, \gamma]$. It follows that our algorithm outputs an optimal solution. □

### 2.2 Running time

The initial path decomposition and the first coloring can be computed in $O(n)$ time. The path decomposition is maintained as a doubly linked list. Building the queue $Q$ requires that we sort the weights of the $z_j$ vertices. After that, in each iteration we update the path decomposition by merging two adjacent subpaths. We can carry out the update in $O(1)$ time by keeping for each vertex in $Q$, a pointer to its position in the path decomposition. Each iteration removes one $z$-vertex from our path decomposition, so the total running time of the for loop is $O(r)$. Hence, other than the sorting done in Line 6, our algorithm runs in $O(n)$ time. The sorting time is our most expensive step. In the RAM model, this step can be carried out in $O(n\sqrt{\log \log n})$ expected time using the algorithm of Han and Thorup (Han and Thorup 2002) for integer sorting. In fact, this sorting step is more efficient in many cases, for example when the weights are polynomially bounded in $n$.

**Theorem 2** *The running time of our algorithm is $O(n + S(r))$, where $S(r)$ is the time it takes to sort the weights of the vertices $z_1, \ldots, z_r$ used in the initial path decomposition.*

## 3 Extension to skinny trees

In this section we present our algorithm for max-coloring a tree $T = (V, E)$ having the property that the set of vertices of degree at least 3 forms an independent set. We call such a tree *skinny*. As we will see next, any such tree has the interesting property that it always has an optimal coloring with at most 3 colors. Two notable examples of trees falling in this class are paths (every vertex has degree at most 2) and spiders (there is a single vertex with degree 3 or more).

**Lemma 4** *Every skinny tree $T$ has an optimal max-coloring using at most* 3 *colors.*

*Proof* Suppose a fourth color is used, where colors RED $\geq$ BLUE $\geq$ GREEN denote the three heaviest color classes. Consider any vertex $x$ that is colored by a color different from these 3 colors. If this is a degree $\leq 2$ vertex, then one of RED, BLUE, GREEN is missing from its neighborhood. Hence we can change the color of $x$ to this missing color and this results in a valid coloring and the value to be optimized does not increase by this change of color. Hence after we perform this step for every such $x$ of degree at most 2, only vertices of degree 3 or higher may be colored by a color other than RED, BLUE, or GREEN.

Let $y$ be such a vertex. If $y$ has no GREEN-colored neighbors, then we can change the color of $y$ to GREEN. So let us assume that $y$ has a neighbor $v$ that is colored GREEN. By the structure of $T$, we know that the degree of $v$ is at most 2. Since one neighbor of $v$ is $y$, there is at most another neighbor of $v$; hence one of RED, BLUE is missing from $v$'s neighborhood—thus we can change the color of $v$ from GREEN to this missing color in RED, BLUE. The value to be optimized does not increase by this change of color. Repeating this step for every neighbor of $y$ that is colored GREEN allows us to eliminate GREEN from $y$'s neighborhood. Now $y$ can be colored GREEN. Hence it follows that the optimal max-coloring on such a graph uses at most 3 colors. □

From Lemma 4, an $O(n^3)$ time algorithm follows easily: Pemmaraju and Raman (2005) gave an $O(kn)$ time algorithm that given a tree $T$ and a tuple $[W_1, \ldots, W_k]$ determines if there is a feasible coloring $[W_1, \ldots, W_k]$ of $T$; since there are $O(n^2)$ possible triples $[w_{\max}, \beta, \gamma]$ for 3-colorings, we get an $O(n^3)$ time algorithm. We improve on this by showing an almost linear time algorithm for max-coloring skinny trees. Our algorithm has the same two basic parts as the algorithm in Sect. 2: an *initialization* part and a *main* part. We first describe the algorithm and then proceed to show its correctness. The implementation details will be described later. The complete pseudocode for our algorithm is given in Fig. 3.

*Initialization* As we did before, we look at each edge and determine a lower bound $b_0$ for BLUE as $b_0 = \max_{(u,v)\in T} \min\{w(u), w(v)\}$. Since for any edge $(u, v)$, both

**Algorithm 2** MAX-COLORING-SKINNY-TREE($T, w$)

1. $b_0 \leftarrow g_0 \leftarrow \max_{(u,v)\in T} \min\{w(u), w(v)\}$
2. $H \leftarrow \{v \in V \mid w(v) > b_0\}$
3. $L \leftarrow \{v \in V \mid w(v) \leq b_0\}$
4. $Q \leftarrow$ list $z_1, \ldots, z_{|L|}$ of vertices in $L$ in non-increasing order of weight
5. **for** $z_i$ in $Q$ **do**
6. transfer $z_i$ from $L$ to $H$
7. let $C$ be the connected component of $H$ containing $z_i$
8. $[\alpha, \beta] \leftarrow$ optimally 2-color $C$
9. $b_i \leftarrow \max\{\beta, b_{i-1}\}$
10. $g_i \leftarrow \max_{v\in L} w(v)$
11. record $[w_{\max}, b_i, g_i]$ as a candidate solution
12. **return** best coloring among the candidate solutions

**Fig. 3** Pseudocode of our algorithm for max-coloring a skinny tree

$u$ and $v$ cannot be colored RED, it follows that $b_0$ is a lower bound for BLUE. We partition the vertices into a set of heavy nodes $H = \{u \in T \mid w(u) > b_0\}$ and a set of light nodes $L = \{u \in T \mid w(u) \leq b_0\}$.

Notice that the vertices in $H$ form an independent set. Hence there exists a $[w_{\max}, b_0, g_0]$ coloring with $g_0 = b_0$ where the vertices in $H$ are colored RED and the vertices in $L$ are colored BLUE and GREEN. In order to reduce this cost we will transfer some vertices from $L$ to $H$. This will reduce the cost of the GREEN class, but it may drive up the cost of the BLUE class. In this way, we produce a number of candidate solutions. At the end, we output the coloring with minimum cost. How these candidate solutions are produced is discussed next.

*The main part* In the $i$th iteration we transfer a node $u \in L$ of maximum weight to $H$. Then we optimally color with RED and BLUE the nodes in $H$ to obtain a coloring with cost $[\alpha, \beta]$. We extend this into a proper 3-coloring by coloring the remaining vertices of $L$ with RED, BLUE, and GREEN. The cost of this coloring is at most $[w_{\max}, b_i, g_i]$ where $b_i = \max\{\beta, b_{i-1}\}$ and $g_i = \max_{u \in L} w(u)$. We record this coloring as a candidate solution and at the end of the algorithm (when $L = \emptyset$) we return the best candidate solution thus produced.

### 3.1 Correctness

The correctness of the algorithm relies on the following lemmas.

**Lemma 5** *Let $H$ be a subset of the vertices of a skinny tree $T$. Then every coloring of $H$ with* RED *and* BLUE *can be extended to the whole tree using an extra color* GREEN.

*Proof* First color with GREEN all the vertices in $T \setminus H$ whose degree is 3 or more. The vertices that remain uncolored form a collection of paths whose endpoints have degree 2 or 1 in the original tree. These left-over paths can be easily colored without creating any conflicts. □

**Lemma 6** *In every valid coloring $[w_{\max}, \beta, \gamma]$ of $T$ we have $\beta \geq b_0$.*

The above lemma follows from the fact that $b_0 = \max_{(u,v) \in T} \min\{w(u), w(v)\}$.

**Lemma 7** *Let $i$ be such that $b_i < b_{i+1}$ and $[w_{\max}, \beta, \gamma]$ be a valid coloring of $T$ with $\beta < b_{i+1}$. Then we must have $\gamma \geq g_i$.*

*Proof* Let $H$ and $L$ be the set of heavy and light vertices after the $i$th iteration is executed and let $z_{i+1}$ be the vertex in $L$ to be transferred to $H$ in the $i+1$st iteration. Recall that $g_i = \max_{v \in L} w(v) = w(z_{i+1})$.

The vertices in $H$ form a number of connected components in $T$. If the vertex $z_{i+1}$ is adjacent to none of these components then the cost of 2-coloring $H + z_{i+1}$ cannot be more than 2-coloring $H$. Therefore, adding $z_{i+1}$ must either enlarge a single component or merge several components. At any rate, the cost of optimally 2-coloring this new component must be $[\alpha, b_{i+1}]$—the cost of coloring the remaining

components does not change with the update. Since the coloring $[w_{\max}, \beta, \gamma]$ from the lemma statement has $\beta < b_{i+1}$, it must be the case that at least one vertex in the component is colored GREEN. Since all nodes in the component have weight greater or equal than $g_i$, it follows that $\gamma \geq g_i$. □

**Theorem 3** *The algorithm* MAX-COLORING-SKINNY-TREES *outputs a feasible optimal max-coloring.*

*Proof* First we note that by Lemma 5, every candidate solution is feasible and thus the algorithm outputs a feasible solution.

Let $[w_{\max}, b_r, g_r]$ be the last coloring produced by the algorithm. For each valid 3-coloring $[w_{\max}, b, g]$ where $b < b_r$ it follows from Lemma 7 and the fact that $b \geq b_0$ that our algorithm finds an algorithm just as good. If $b \geq b_r$ then the last coloring produced is at least as good since $g_r = 0$ (at this point $L = \emptyset$). □

### 3.2 Implementation details

We will show that except for Line 4, the algorithm runs in linear time. In fact, we only need to argue that the time spent on Lines 7 and 8 is linear as it is straightforward to implement the remaining lines in linear time. The following theorem shows this can be done.

We conclude this section with Theorem 4.

**Theorem 4** *The algorithm* MAX-COLORING-SKINNY-TREES *can be implemented to run in $O(n + S(n))$ time, where $S(n)$ is the time it takes to sort the vertices by weight.*

*Proof* First we need to keep track of the connected components of $H$. This can be done using a union-find data structure. We note that each time we process a node $z_i$, we need to find the components of its neighbors so we execute at most $\deg(z_i)$ union and $\deg(z_i)$ find operations. Normally, this would add up to superlinear time since the best we can get is $O(1)$ time per union and $O(\alpha(n))$ amortized time per find where $\alpha$ is the inverse Ackermann's function (Tarjan 1975, 1979). However, in our case, the union operations follow the underlying tree structure and for precisely this setup there exists a data structure that can execute both operations in $O(1)$ amortized time (Gabow and Tarjan 1985). This means that we can execute Line 7 in $O(\deg(z_i))$ amortized time.

In order to carry out Line 8 within the same time bound, we need to keep some information about our components. We assume that $T$ is rooted at some arbitrary vertex and we define $E$ as the set of vertices with even depth and $O$ as the set of vertices in $T$ with odd depth. For each connected component $C$ in $H$, we store the maximum weight of vertices in $C \cap E$ and the maximum weight of vertices in $C \cap O$; we denote these two values with $e_C = \max_{v \in C \cap E} w(v)$ and $o_C = \max_{v \in C \cap O} w(v)$. (Note that it is very easy to maintain this information as we transfer vertices from $L$ to $H$.) Then an optimal 2-coloring of $C$ is simply $[\max\{e_C, o_c\}, \min\{e_C, o_C\}]$.

It follows that the total time we spend in Lines 7 and 8 is $O(\sum_{v \in T} \deg(z_i)) = O(n)$. The remaining lines can clearly be implemented to run in linear time as well. □

## 4 A fully polynomial-time approximation scheme

Except for the sorting step, the rest of our algorithm for skinny trees runs in $O(n)$ time. In this section we exploit this fact to get an algorithm that computes a $(1+\epsilon)$-approximation of the optimal max-coloring in $O(n+1/\epsilon)$ time.

**Theorem 5** *There is an $O(n+1/\epsilon)$ time $(1+\epsilon)$ approximation for max-coloring a skinny tree.*

*Proof* The idea is very simple. Given a skinny tree and a weight function $w$ and a real number $\epsilon > 0$, we create another weight function $w'$ so that for every vertex $v_i$ in the tree

$$w'(v_i) = \left\lfloor \frac{3w(v_i)}{\epsilon w_{\max}} \right\rfloor.$$

The weights under $w'$ are integers in the range $[0, \lceil \frac{3}{\epsilon} \rceil]$. Therefore, by Theorem 2 we can find an optimal max-coloring in $O(n+\frac{1}{\epsilon})$ time using bucket sort for sorting the weights of the $z_j$ vertices. We claim that this coloring is a $1+\epsilon$ approximation for the original weights $w$. Let OPT be the optimal coloring under $w$ and $w(\text{OPT})$ be its cost; similarly, let OPT$'$ be the optimal coloring under $w'$ (the one output by our algorithm) and $w'(\text{OPT})$ be its cost.

We bound the weight of our solution as follows

$$\begin{aligned}
w(\text{OPT}') &\le \big(w'(\text{OPT}') + 3\big)\frac{\epsilon w_{\max}}{3} && \text{[there are only three colors]}\\
&\le \big(w'(\text{OPT}) + 3\big)\frac{\epsilon w_{\max}}{3} && \text{[OPT}' \text{ is optimal for } w']\\
&\le w(\text{OPT}) + \epsilon w_{\max} && \text{[by definition of } w']\\
&\le (1+\epsilon)\, w(\text{OPT}). && \text{[since } w(\text{OPT}) \ge w_{\max}]
\end{aligned}$$

□

## 5 Lower bounds on max-coloring a path with real weights

Our algorithm for max-coloring a path takes $O(n + S(r))$ time, where $S(r)$ is the time to sort the weights of $r$ vertices. The max-coloring problem was defined for integral vertex weights since this problem was motivated by scheduling problems, where the time to process each job is integral. However the max-coloring problem can be naturally extended to real vertex weights. When the vertex weights are real, it takes $O(n \log n)$ time to sort the weights, so our algorithm also takes $O(n \log n)$ time. We will now show that this is the optimal running time for max-coloring a path in the real algebraic computation tree model. In this model the operations $\{+, -, \times, \div, \sqrt{\ }, =0, \ge 0\}$ on reals can be performed at unit cost.

Our lower bounding method works via a problem that we call the *max-difference problem*. In the max-difference problem, we are given a set $\{y_1, \ldots, y_n\}$ of real numbers and let $\langle y_{\pi(1)}, \ldots, y_{\pi(n)} \rangle$ be these elements sorted in nondecreasing order. The problem is to determine the maximum value of $y_{\pi(i+1)} - y_{\pi(i)}$ for $1 \le i \le n-1$.

### 5.1 The reduction from max-difference to max-coloring a path

We will show an $O(n)$ time reduction from the max-difference problem to the max-coloring a path problem. Given a set $\{y_1, \ldots, y_n\}$ of $n$ real numbers, let us determine the maximum and minimum of $y_1, \ldots, y_n$ in $O(n)$ time. Assume without loss of generality that $y_1$ is the minimum and $y_n$ is the maximum. We form an instance of the max-coloring problem on a path as follows.

The path $(v_1, v_2, \ldots, v_N)$ consists of $N = 5(n-1)+2$ vertices. The leftmost vertex $v_1$ has weight $y_n + y_1 - \epsilon$, where $\epsilon > 0$ is a small constant. The next vertex $v_2$ has weight $2y_n$. Recall that the largest value among the $y_i$'s is $y_n$; the weight of $2y_n$ will be the largest vertex weight in our max-coloring instance. We can therefore assume that $v_2$ is colored RED. Let the next heaviest color class be BLUE. Note that since $v_1$ is adjacent to $v_2$, the BLUE color class has weight at least $y_n + y_1 - \epsilon$.

The remaining vertices $(v_3, \ldots, v_N)$ can be split into $n-1$ blocks, each with 5 vertices. Let us call the 5 vertices of the $i$th block $z_i, u_i, x_i, x_i', u_i'$, where $z_i$ has weight 0, vertices $u_i$ and $u_i'$ have weight $y_n + y_i$ and vertices $x_i$ and $x_i'$ have weight $y_n - y_i$. Note that if both $u_i$ and $u_i'$ are colored RED, then one of $x_i, x_i'$ has to be colored GREEN. The vertices $z_i$ of weight 0 in each block allow the blocks to be independent of each other.

There is a coloring of the path where the BLUE color class has weight $y_n + y_1 - \epsilon$ (the least possible weight for BLUE). Since $\epsilon > 0$, this forces all the $2(n-1)$ vertices of weights in $\{y_n + y_1, \ldots, y_n + y_{n-1}\}$ to be colored RED in the initial coloring. This thereby forces certain vertices with weights in $\{y_n - y_1, \ldots, y_n - y_{n-1}\}$ to be colored GREEN. So the GREEN color class has weight $y_n - y_1$ here (since $y_1$ is the minimum among the $y_i$'s, $y_n - y_1$ is the largest among the $y_n - y_i$'s). Thus the sum of the BLUE and GREEN color classes in the above coloring is $y_n + y_1 - \epsilon + y_n - y_1 = 2y_n - \epsilon$. Here we only compute the sum of BLUE and GREEN color classes to compare colorings since the RED always has weight $2y_n$.

If we want a lower weight for the GREEN color class, then we need to increase the weight of the BLUE color class. Suppose we want the weight of the GREEN color class to go down to $y_n - y_{\pi(k)}$; this means that all the vertices with weights in $\{y_n - y_1, y_n - y_{\pi(2)}, \ldots, y_n - y_{\pi(k-1)}\}$ are colored RED or BLUE.

**Lemma 8** *For all the vertices of weight $y_n - y_{\pi(k-1)}$ or more to be colored* non-GREEN, *a vertex of weight $y_n + y_{\pi(k-1)}$ should be colored* BLUE.

*Proof* Note that for each $i$, we have 2 vertex weights of $y_n - y_i$ (vertices $x_i$ and $x_i'$ in Fig. 4) sandwiched between $u_i$ and $u_i'$ of weight $y_n + y_i$ each. So while the BLUE color class has weight $< y_n + y_i$, the 2 vertices $u_i$ and $u_i'$ have to be colored RED, which forces us to color one of $x_i, x_i'$ GREEN. Thus for both the vertices of weight $y_n - y_{\pi(k-1)}$ to be colored *non*-GREEN, a vertex of weight $y_n + y_{\pi(k-1)}$ has to be colored BLUE. □

We will take $\epsilon$ to be a very small value, say $\epsilon < \frac{y_n - y_1}{n-1}$.

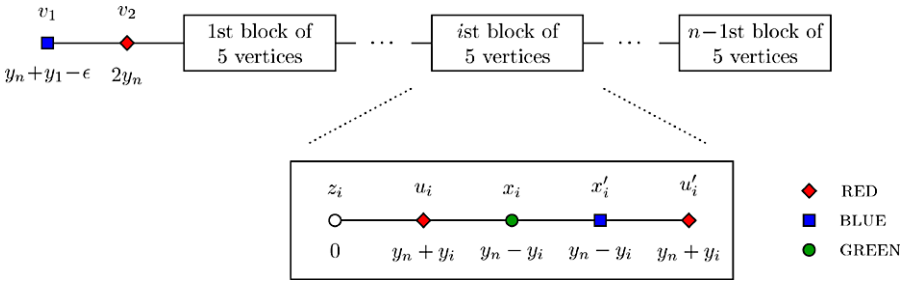


**Fig. 4** The overall structure of the path in our reduction. There are $5(n-1)+2$ vertices in total with $n-1$ blocks of 5 vertices each. The $i$th block is shown in detailed. The name of each vertex appears above the vertex and its weight, below

**Lemma 9** *Let* $[w_{\max}, \beta, \gamma]$ *be an optimal solution for the max-coloring instance and* $\Delta = y_{\pi(i)} - y_{\pi(i-1)}$ *be the optimum value of the maximum difference instance. Then* $\beta + \gamma = 2y_n - \Delta$.

*Proof* It is easy to see that for any $k$, where $2 \le k \le n$, with $y_n + y_{\pi(k-1)}$ as the heaviest BLUE vertex weight, we are free to color vertices with weights in $\{y_n + y_{\pi(1)}, y_n + y_{\pi(2)}, \ldots, y_n + y_{\pi(k-2)}\}$ BLUE. Thus we can make all vertices with weights in $\{y_n - y_{\pi(1)}, y_n - y_{\pi(2)}, \ldots, y_n - y_{\pi(k-1)}\}$ *non*-GREEN and the GREEN color class has weight $y_n - y_{\pi(k)}$. This makes the sum of weights of BLUE and GREEN equal to $y_n + y_{\pi(k-1)} + y_n - y_{\pi(k)} = 2y_n + y_{\pi(k-1)} - y_{\pi(k)}$; in particular, when $k = i$, the sum of BLUE and GREEN color classes equals $2y_n - \Delta$.

Our candidate colorings yield the following weights for the sum of BLUE and GREEN color classes:

$$2y_n - \epsilon, 2y_n + y_{\pi(1)} - y_{\pi(2)}, \ldots, 2y_n + y_{\pi(k-1)} - y_{\pi(k)}, \ldots, 2y_n + y_{\pi(n-1)} - y_{\pi(n)},$$

which is minimized for $2y_n + y_{\pi(i-1)} - y_{\pi(i)} = 2y_n - \Delta$. In other words, minimizing $\beta + \gamma$ is the same as maximizing $\Delta$. □

Thus solving the max-coloring a path problem solves the max-difference problem. An $O(t(n))$ running time for max-coloring a path implies an $O(t(n)+n)$ algorithm, that is, an $O(t(n))$ algorithm for the maximum difference problem, since $t(n) \ge n$ for the max-coloring problem.

### 5.2 A lower bound for the max-difference problem

In this section we will show that in the algebraic computation tree model, if there exists an $O(t(n))$ algorithm for the max-difference problem then there exists an $O(t(n)+n)$ algorithm for the following problem: Given $\{x_1, \ldots, x_n\} \in \mathbb{R}^n$, do these numbers in sorted increasing order form a *non-trivial arithmetic progression*? That is, this problem "*Is-it-an-AP*" asks if there a $d > 0$ such that

$$\{x_1, \ldots, x_n\} = \{\min, \min + d, \min + 2d, \ldots, \min + (n-1)d\},$$

where min is the minimum among $x_1, \ldots, x_n$. Then we will show an $\Omega(n \log n)$ lower bound for *Is-it-an-AP*.

The $O(n)$ time reduction from "Is-it-an-AP" to "max-difference" is simple. We compute the max-difference of $x_1, \ldots, x_n$, call this $\Delta$, and the maximum and minimum $x$ values, call these $x_{\max}$ and $x_{\min}$. It is easy to see that $\{x_1, \ldots, x_n\}$ forms a *yes* instance of "Is-it-an-AP" if and only if $x_{\max} - x_{\min} = \Delta(n-1)$ and $\Delta > 0$.

*Lower bound on "Is-it-an-AP"* Our lower bounding method is the same as in the Element Distinctness problem. The *yes* instances of this problem form at least $n!$ connected components (the same reasoning as in Element Distinctness), so it follows from Ben-Or's result that any algebraic computation tree solving the *Is-it-an-AP* problem has depth $\Omega(\log n! - n)$, which is $\Omega(n \log n)$. Note that this problem can certainly be solved in $O(n \log n)$ time by just sorting the elements and checking if this sequence is a non-trivial arithmetic progression.

*Number of connected components formed by* yes *instances* Let $\{x_1, \ldots, x_n\}$ be a *yes* instance for Is-it-an-AP. Thus every permutation of $\{x_1, \ldots, x_n\}$ is also a *yes* instance. The *yes* instances partition $\mathbb{R}^n$ into connected components. Thus $(x_1, \ldots, x_n)$ belongs to such a component.

The claim is that no two permutations of $(x_1, \ldots, x_n)$ belong to the same connected component. Consider any two permutations $(x_{\pi(1)}, \ldots, x_{\pi(n)})$ and $(x_{\tau(1)}, \ldots, x_{\tau(n)})$. If these two points belong to the same connected component then there is a continuous path joining $(x_{\pi(1)}, \ldots, x_{\pi(n)})$ and $(x_{\tau(1)}, \ldots, x_{\tau(n)})$ in this component. There exist indices $i$ and $j$ such that $x_{\tau(i)} < x_{\tau(j)}$ but $x_{\pi(i)} > x_{\pi(j)}$, thus there exists a point $(r_1, \ldots, r_n)$ on this path with $r_i = r_j$. Such a point cannot be a *yes* instance of Is-it-an-AP, since if $\{r_1, \ldots, r_n\}$ in sorted order forms an arithmetic progression then its common difference $d$ has to be 0, however such a point is a *no* instance since $d$ must be strictly larger than zero for a *yes* instance of Is-it-an-AP. This proves that the *yes* instances form at least $n!$ connected components.

We can conclude that the Is-it-an-AP problem has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model; by our earlier reductions so does the max-difference problem and so does the problem of max-coloring a path with real vertex weights. Thus we have shown the following theorem.

**Theorem 6** *The complexity of max-coloring a path with real vertex weights is* $\Theta(n \log n)$ *in the algebraic computation tree model.*

## 6 Conclusions

We showed an upper bound of $O(n +$ time to sort the vertex weights) for the problem of max-coloring a path on $n$ vertices and then generalized this to skinny trees. These algorithms can be adapted to compute $1 + \epsilon$ approximations in $O(n + 1/\epsilon)$ time. We also showed a matching lower bound of $\Omega(n \log n)$ for max-coloring paths with *real* vertex weights in the algebraic computation tree model. It remains an interesting open problem to improve the time complexity of max-coloring more general classes of trees and, ultimately, to optimally max-color an arbitrary tree in polynomial time.

**Acknowledgements** Thanks to Magnús Halldórsson and Rajiv Raman for helpful discussions.



## References

Brucker P (2004) Scheduling algorithms, 4th edn. Springer, Berlin

Epstein L, Levin A (2007) On the max-coloring problem. In: Proceedings of the 5th workshop on approximation and online algorithms

Escoffier B, Monnot J, Paschos VT (2006) Weighted coloring: further complexity and approximability results. Inf Process Lett 97(3):98–103

Gabow HN, Tarjan RE (1985) A linear-time algorithm for a special case of disjoint set union. J Comput Syst Sci 30(2):209–221

Guan DJ, Zhu X (1997) A coloring problem for weighted graphs. Inf Process Lett 61(2):77–81

Halldórsson MM, Shachnai H (2008) Batch coloring flat graphs and thin. In: Proceedings of the 11th Scandinavian workshop on algorithm theory, pp 198–209

Han Y, Thorup M (2002) Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43th annual IEEE symposium on foundations of computer science, pp 135–144

Han C-C, Hou C-J, Shin KJ (1995) On slot reuse for isochronous services in DQDB servers. In Proc of 16th IEEE real-time systems symposium, pp 222–231

Liu H, Beck M, Huang J (2006) Dynamic co-scheduling of distributed computation and replication. In: Proc of CCGRID, pp 592–600

Pemmaraju SV, Raman R (2005) Approximation algorithms for the max-coloring problem. In: Proceedings of the 32th international colloquium on automata, languages and programming, pp 1064–1075

Tanenbaum AS (1995) Distributed operating systems. Prentice Hall, New York

Tarjan RE (1975) Efficiency of a good but not linear set union algorithm. J ACM 22(2):215–225

Tarjan RE (1979) A class of algorithms which require nonlinear time to maintain disjoint sets. J Comput Syst Sci 18(2):110–127