# Algorithms for Chromatic Sums, Multicoloring, and Scheduling Dependent Jobs

Magnús M. Halldórsson ICE-TCS, School of Computer Science, Reykjavik University, Iceland.*
Guy Kortsarz Rutgers University, Camden, New Jersey.†

May 18, 2017

## 1 Introduction

This survey deals with problems at the intersection of two scientific fields: graph theory and scheduling. They can either be viewed as scheduling *dependent jobs* – jobs with resource conflicts – or as graph coloring optimization involving different objective functions.

Our main aim is to illustrate the various interesting algorithmic techniques that have been brought to bear. We will also survey the state of the art, both in terms of approximation algorithms, lower bounds, and polynomial time solvability.

We first formulate the problems, both from a scheduling and from a graph theory perspective, before setting the stage.

### 1.1 Scheduling Perspective

The input consists of *jobs* that need to be *processed* on *machines*. There are enough machines available (although the case of limited number of machines can also be handled). We view the processing to take place in *time steps* or *rounds*. Each job $J$ has an integral *length* or processing time $x(J)$, so it must be processed by a machine during $x(J)$ of the rounds. The task is to process the jobs on the machines as quickly as possible. What makes the task non-trivial is that each job requires an *exclusive* access to a set of resources. (That means that two jobs that require the same resource cannot be processed simultaneously.) Therefore, the jobs processed during a given round must use disjoint sets of resources.

There are two essential modes of processing the jobs. In the *non-preemptive* setting, once a job is processed, it must be executed to completion. That means that a job is always processed in consecutive rounds. In the *preemptive* case, there are no constraints of this sort. Finally, in the important special case of *unit-length jobs*, each job is processed only for a single round.

We will consider two objective functions: minimizing the total length of the schedule, and minimizing the average completion time. Given a legal schedule, let $f(J)$ be the time $J$ is finished. From the view of the *system*, it is preferable to finish the last job early. This is known as minimizing the *makespan* of the

*mmh@ru.is. Supported by Icelandic Research Fund grants 120032011 and 152679-051.
†guyk@camden.rutgers.edu. Supported by NSF grants 1218620 and 1540547

schedule, or $\max_J f(J)$. The *users* that "own" the jobs are most concerned that *their* job is finished early. The natural objective to consider for that is to minimize the *sum of completion times*, $\sum_J f(J)$, which is equivalent to minimizing the *average* finish time of a job.

## 1.2 A Graph Theoretic Description

An alternative view of the above problems is as *multicoloring* problems, with colors being positive integers. The number $x(J)$ is then called the *color requirement* of $J$. The jobs naturally define a conflict graph with vertices representing jobs, and with an edge between a pair of jobs that require the same resource. Using graph terminology, a round is *an independent set* (namely a set of vertices no two of which share an edge). Thus, a schedule is an ordered collection of independent sets, or a *multicoloring* (not just a *coloring* since vertices can require multiple colors). Each vertex (job) $v$ is to receive $x(v)$ colors (i.e., belong to $x(v)$ independent sets), so that adjacent vertices receive disjoint sets of colors. *In preemptive multicoloring problems*, there is no restriction on the $x(v)$ colors that each vertex can receive, except that sets assigned to adjacent vertices must be disjoint, as before. A different problem variant occurs when we require the colors assigned to a vertex to form a contiguous interval – we refer to such an assignment as a *non-preemptive multicoloring*.

We consider two objective functions. We assume that the colors are enumerated as the natural numbers, $1, 2, 3, \ldots$ The *makespan* corresponds to the number of colors used, which equals the largest color used. The *chromatic sum* criteria corresponds to $\sum_v f(v)$, with $f(v)$ being the largest color given to vertex $v$.

A coloring with few colors does not necessarily imply a low sum coloring. Even for `SC`, there are examples for which using the minimum $\chi(G)$ number of colors gives a sum that is $\Theta(\sqrt{n})$-factor larger than the minimum sum.

There are also examples in which preemptive coloring leads to a strictly smaller color sum. Consider a path $A - B - C$ with color requirements $x(A) = 1, x(B) = 2, x(C) = 5$. The reader may verify that every non-preemptive coloring has sum at least 12, while there is a preemptive coloring with sum 11.

## 1.3 Notation

The input is an undirected, unweighted graph $G = (V, E)$. For $S \subseteq V$, let $\mathcal{S}(S) = \sum_{v \in S} x(v)$ be the sum of the color requirements of nodes in $S$, and let $\mathcal{S}(G) = \mathcal{S}(V)$. Let $p$ denote the largest color requirement. Let $\chi(G)$ be the minimum number of colors required to (multi)color $G$. Let $N(v)$ be the set of neighbors of vertex $v$. We may intermix the scheduling and graph theory vocabulary in the remainder.

We denote `SC` for minimum *sum coloring* problem (where $x(v) = 1$ for every $v$); `pMC` (`npMC`) for preemptive (non-preemptive) minimum *makespan multicoloring*; and `pSMC` (`npSMC`) for preemptive (non-preemptive) *sum multicoloring* problem, respectively.

## 1.4 Applications for Sum (Multi-)Coloring

**Wire minimization in VLSI design,** [41], terminals lie on a single vertical line, where each terminal is represented by a short interval on this line. To its right are vertical bus lines, separated with unit spacing. Pairs of terminals are to be connected via horizontal wires on each side to a vertical lane, with non-overlapping pair utilizing the same lane. With the vertical segments fixed, the wire cost corresponds

to the total length of horizontal segments. Numbering the lanes in increasing order of distance from the terminal line, lane assignment to a terminal corresponds to coloring the terminal's interval by an integer. The wire-minimization problem then corresponds to sum coloring of an interval graph.

**Storage allocation** is a problem of minimizing the total distance distance traveled by a robot moving in a warehouse [45]. Goods are located in a single corridor from the delivery spot. They are checked in and out at known times. Two robots can move only if their intervals do not intersect. Since the starting point and destinations are on a line, this gives the problem of multicoloring *interval graphs*.

**Session scheduling on a path:** In a path network,pairs of vertices need to communicate, for which they need use of the intervening path. If two paths intersect, the corresponding sessions cannot be held simultaneously. In this case, it would be natural to expect the sessions (i.e.,jobs") to be of different lengths, leading to the sum multicoloring problem on interval graphs

**Resource-constrained scheduling**: Say that we are given a collection of *n jobs* of integral lengths and a collection of *resources*. We assume that each job requires an exclusive access to particular subset of the resources to execute. The resource-constrained scheduling problem can then be modeled as a multicoloring problem on the conflict graph. We address here the case where each task uses up to $k$ resources. The conflict graph is then $k+1$-*claw free*, i.e., does not contain the star $K_{1.k}$ as an induced subgraph, since disjoint vertices must use disjoint sets of resources. A natural example of a limited resource is processors in a multi-processor system. In the biprocessor task scheduling problem, we are given a set of jobs and a set of processors, and each job requires the exclusive use of two dedicated processors. We are interested in finding a schedule which minimizes the sum of completion times of the jobs. In scheduling terms, this problem is denoted by $P|\text{fix}_j|\sum C_j$, with $|\text{fix}_j| = 2$. In the special case of two resources per task, such as the biprocessor task scheduling problem, the conflict graph is a line graph. Another application of sum multicoloring of $(k+1)$-claw free graphs is scheduling data migration over a network, also known as the *file transfer problem* (see, e.g., [10, 29]). Suppose that the network is fully connected, and we have a set of files $f_1, \ldots, f_M$; each file $f_i$ needs to migrate from a source $s_i$ to a destination $t_i$. During migration, $f_i$ requires the exclusive access to $s_i$ and $t_i$, for a pre-specified time interval; thus, $f_i$ and $f_j$ are in conflict if $\{s_i, t_i\} \cap \{s_j, t_j\} \neq \emptyset$. Our goal is to find a migration schedule that minimizes the sum of completion times, where the sum is taken over all files. This translates to the sum multicoloring problem on the conflict graph of the files, which is a line graph.

## 1.5 Approximation Algorithms and Inapproximability

We deal primarily with minimization problems. The goal is to find an $s_i$ with $v(s_i)$ minimum. All the problems we consider are $NP$-hard. Since $NP$-hard problems are unlikely to admit a polynomial time solution, we need to settle for an approximate solutions. Let $opt(I)$ be the value of the best solution for problem instance $I$. For a minimization (respectively maximization) problem $P$, an algorithm is called *a $\rho$-approximation algorithm*, if it runs in time polynomial in the input size, and for every input $I$, the algorithm outputs a solution of cost at most $\rho \cdot opt(I)$ (respectively, at least $opt(I)/\rho$).

A *lower bound* $\rho$ on the approximability of a problem is a proof that approximating the problem within factor $\rho$ in polynomial time would give a polynomial time algorithm for solving the problem exactly. Since the latter problem is $NP$-hard, so is the former.

*A polynomial-time approximation scheme* (PTAS) for a minimization problem is an algorithm that, given

a constant parameter $\epsilon$, produces a $(1+\epsilon)$-approximate solution for every $\epsilon$. The running time should be polynomial for $\epsilon$ constant, but could, e.g., be $n^{f(1/\epsilon)}$, for some arbitrary function $f$, where $n$ is the size of the instance. A *fully polynomial time approximation scheme* (FPTAS) for $P$ is an a $1+\epsilon$-approximation algorithm that runs in time polynomial in $n$ *and* in $1/\epsilon$.

### 1.6 Graph Classes

We define here the main classes of graphs that we will encounter.

*Perfect graphs* (see [21]) are those for which the clique number equals the chromatic number, in every induced subgraph. *Chordal graphs* are graphs that contain no cycle of size 4 or larger as an induced subgraph. An *Interval graph* is a special case of a Chordal graph: vertices correspond to intervals on a line with two vertices adjacent whenever their intervals intersect. *Bipartite* graphs are the 2-colorable graphs.

A *k-tree* is a graph that can be formed by the following process. Starting with a $k+1$-clique (complete graph on $k+1$ vertices), and keep adding vertices adjacent to some $k$-clique. A *Partial k-tree* is a subgraph of a $k$-tree. These graphs are exactly the graphs of *Treewidth k* (the definition via treewidth is more complex and outside our scope).

A *Line graph* has a vertex for each edge of an underlying graph, with adjacencies between edges that intersect. A graph is $k+1$-claw free if it does not contains the star $K_{1,k+1}$ as an induced subgraph.

*Planar graphs* are those that can be drawn on a plane with no two edges intersecting. *Hexagon graphs* are a subclass of planar grpah defined in [38], with important applications for cellular networks. A *Unit disc graph* [8] has vertices represented by unit circles in the plane, with two vertices being adjacent if the circles intersect.

**Outline** In the next section, we overview quickly the state-of-the-art on sum-coloring and multicoloring problems. We then devote a section to each of the major approaches used for tackling sum (multi-)coloring problems: Greedy approaches in Sec. 3, randomized "doubling" in Sec. 4, partitioning in Sec. 5, and delays in Sec. 6. We then close with a few open problems.

## 2 The State of the Art

We briefly summarize the known results on makespan coloring and sum-coloring problems.

**Makespan problems** (Multi)coloring a *Line graph* is equivalent to (multi)coloring the edges of a graph, so that adjacent edges are assigned disjoint sets of colors.

For certain graphs, the `pMC` problem can be reduced to the ordinary coloring problem. A vertex $v$ of color requirement $x(v)$ is replaced by a clique of $x(v)$ vertices (connecting a copy of $v$ to a copy of $u$ if $u$ and $v$ are connected). This reduction is polynomial if $p$ is polynomial in $n$, but can often be done implicitly for large values of $p$. This gives an optimum algorithm for `pMC` on certain perfect graphs such as *Chordal graphs*. But large cliques cannot be introduced to *Bipartite* (or 2-colorable) graphs, hence the reduction does not work for general perfect graphs. Similarly, one cannot introduce large cliques into a Planar graphs.

The above reduction, also does not work for the non-preemptive case. For a summary of known results for Minimum coloring, `pMC`, and `npMC` see Table 1. The $O^*()$ notation hides $poly(\log\log n)$ factors.

Table 1: Known results for multicoloring problems

| | **Coloring** | | **Multicoloring** | |
|---|---|---|---|---|
| | *u.b.* | *l.b.* | pMC | npMC |
| General graphs | $O^*(n/\log^3 n)$[23] | $n^{1-\epsilon}$[14] | $O^*(n/\log^3 n)$[23] | |
| Perfect graphs | 1 [21] | | | |
| Bipartite Graphs | 1 Folklore | | 1 Folklore | 1 Folklore |
| Chordal graphs | 1 [20] | | 1 [21] | |
| Interval graphs | 1 [20] | | 1 [21] | $2+\epsilon$ [9] |
| Partial $k$-trees | 1 [2] | | 1 [28] | 1 [28] |
| Planar graphs | 4/3 [1] | NPC | 11/6 [34] | |
| Hexagon graphs | 4/3 [38, 40] | NPC [38] | 4/3 [38] | |
| Line graphs | Opt+1 [44] | NPC [22] | 1.1 [39] | |

**Sum coloring problems** The SC problem was first studied explicitly by Kubicka [30]. Efficient algorithms have been given for trees [30], partial $k$-trees [28], and regular bipartite graphs [35]. See Table 2 for a summary of best results known.

Table 2: Known results for sum (multi-)coloring problems

| | **SC** | | **SMC** | |
|---|---|---|---|---|
| | *u.b.* | *l.b.* | pSMC | npSMC |
| General graphs* | $n/\log^3 n$ [4, 13] | $n^{1-\epsilon}$ [4, 14] | $n/\log^3 n$ [6, 13] | $n/\log^2 n$ [6] |
| Perfect graphs | 3.591 [16] | APX [7] | 5.436 [16] | $O(\log n)$ [6] |
| Interval graphs | 1.796 [16] | APX [43] | 5.436 [16] | $11.273+\epsilon$ [16] |
| Bipartite graphs | 27/26 [36] | APX [7] | 1.5 [6] | 2.8 [6] |
| Partial $k$-trees | 1 [28] | | PTAS [24] | FPTAS [24] |
| Planar graphs | PTAS [24] | NPC [24] | PTAS [24] | PTAS [24] |
| Trees | 1 [30] | | PTAS [25] | 1 [25] |
| Line graphs | 1.8298 [27] | NPC | 2 [6] | 7.682 [16] |
| Line graphs of trees | 1 | | PTAS [37] | |
| $k$-claw free graphs | $k$ | | $k$ [8] | $1.796k^2+0.5$ |
| Intersection of $k$-sets | $k$ [4] | | $k$ [6] | $3.591k$+.5 [18] |
| Unit disc graphs | 2 [8] | NPC | 2 [8] | |

# 3 Sum Coloring Fundamentals

We examine first two of the simplest and most natural solution strategies for sum coloring.

**Canonical colorings** The most basic property of a good sum coloring is that it be *minimal*: no vertex can be moved to a smaller color without destroying the coloring property. Namely, each vertex colored $c$ should have neighbors colored $1, 2, \ldots, c-1$. Such a coloring is sometimes called *canonical*.

We can observe that a canonical coloring has sum at most $n+m$, where $n$ ($m$) is the number of vertices (edges), respectively. Form an arbitrary order of the vertices that is consistent with the canonical coloring, and let $b_i$ denote the number of neighbors of vertex $i$ that precede it in the ordering. Observe that each edge $(i,j)$ contributes exactly one to $b_i + b_j$; therefore, the sum equals the number of edges, $\sum_i b_i = m$. We can also note that the color of $i$ is at most $b_i + 1$, and thus the color sum is at most $\sum_i (b_i + 1) = m + n$. This is also equivalent to $\frac{d+2}{2}n$, where $d$ is the average degree of the graph, for an approximation ratio of at most $\frac{d+2}{2}$ [31].

This can be extended to an "everywhere sparse" parameter. A graph has *inductiveness* (or *degeneracy*) $D$ if there is an ordering of the vertices such that each vertex has at most $D$ neighbors that precede it in the ordering. Such an ordering can be found by repeatedly removing the minimum degree vertex. If we then color the vertices in this order, using the smallest available color for each vertex, the resulting chromatic sum is at most $\frac{D+2}{2}n$.

**Greedy colorings** A natural heuristic is to color the vertices one color set at a time, each time finding largest possible independent set in the remaining graph. After all, the more vertices that are colored with the first color, the smaller the sum is likely to be. We call this the *Greedy* algorithm.

More formally, let $R_i$ be the set of vertices that have not been colored by *Greedy* with the first $i-1$ colors. Initially, clearly $R_0 = V$. In each round $i = 1, 2, \ldots$, *Greedy* finds a maximum independent set $X_i$ in the graph induced by $R_i$, assigns those vertices the color $i$, and updates $R_{i+1} = R_i \setminus X_i$.

**Theorem 3.1 ([4])** Greedy *achieves 4-approximation for* SC.

We give a simpler proof from [15], that was stated for the related *Sum Set Cover* problem.

The proof idea is to compare the areas of geometric figures that represent the two colorings, the greedy and the optimal coloring. We form a *histogram* for both colorings, with a column for each vertex of unit width and height proportional to some measure of the contribution of that vertex to the objective function. The key argument is showing that if we reduce the scale of the greedy histogram by a factor of two, along both the $x$- and $y$-dimensions, then it will fit inside the optimal histogram. Since the areas of the original histograms correspond to the color sums, this immediately shows that the sum of the greedy coloring is at most four times that the optimal chromatic sum.

The optimal histogram has a column for each vertex, of unit width and height equivalent to the color assigned. It follows immediately that the area equals the color sum. We order the columns of the histogram in non-decreasing height, which corresponds to the order of the vertices in the optimal coloring.

Define the *price* of $X_i$ as $p_i = |R_i|/|X_i|$. The greedy histogram has a unit-width column for each vertex; the height of the column is $p_i$ if the vertex was colored $i$ by *Greedy* (i.e., if contained in $X_i$). The columns are ordered by the greedy sets $X_i$, and hence the heights of rectangles in the greedy histogram are not necessarily monotone.

Let $Gr = \sum_{i=1}^{k} i|X_i|$ be the sum and $k$ be the number of colors of the coloring returned by *Greedy*. Note that $R_i = \cup_{j \geq i} X_j$.

**Claim 3.2** *The area of the greedy histogram equals* $Gr = \sum_{i=1}^{k} p_i|X_i|$.

**Proof:** Since for each vertex in $X_i$ there is a column of height $p_i$, the area of the greedy histogram is clearly $\sum_{i=1}^{k} p_i|X_i| = \sum_{i=1}^{k} |R_i|$, applying the definition of $p_i$. On the other hand, the coloring of each $X_i$ by Greedy

delays the remaining vertices by $R_i$; thus, $Gr = \sum_{i=1}^{k} |R_i|$. □

Consider Figure 1.

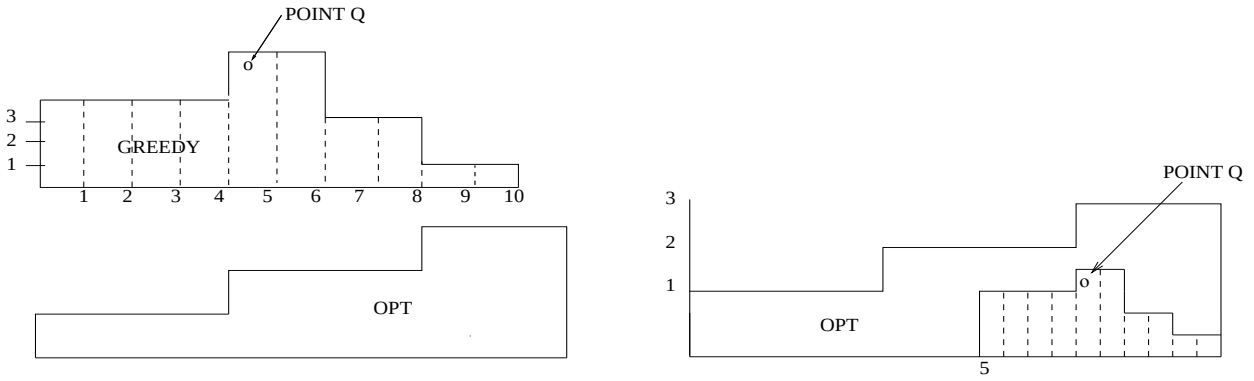


Figure 1: Optimum and greedy histograms (left); the shrunk greedy histogram aligned with the optimal one (right).

We now scale the greedy histogram by halving both the height and width of each column. Thus, in this *shrunk greedy histogram*, the columns corresponding to vertices in $X_i$ have width $1/2$ and height $p_i/2$. We position the shrunk greedy histogram over the optimal histograms so that their lowest right points is collocated. For example, in Figure 1, the greedy histogram has 10 rectangles, ranging from $x = 0$ to $x = 10$, so we place the shrunk greedy histogram so that it starts at point $x = 5$.

To establish our claim, we show that the shrunk histogram is completely contained inside the optimal one. To this end, consider an arbitrary point $Q$ inside the shrunk histogram; see Fig. 1 for an example. Let $i$ be the round in which *Greedy* colored the corresponding vertex (whose rectangle contains $Q$). Let $h$ be the height of $Q$ from the baseline, and note that $h$ is at most the height of its rectangle in the shrunk greedy solution, or $p_i/2$. Recall the set $R_i$ of vertices that *Greedy* has not colored by the start of round $i$. Let $r$ be the distance from $Q$ to the right edge of the shrunk greedy histogram (rounded down to the nearest integer). Note that there are $|R_i|$ vertices left to be colored when *Greedy* starts round $i$, and thus there are at most $|R_i|$ columns to the right of $Q$ in the greedy histogram. In the shrunk histogram, where each is of width $1/2$, that makes for a distance at most $|R_i|/2$; thus, $r \leq |R_i|/2$.

The issue boils down to the height of the $r$-th last column in the optimal histogram, which equals to the number of colors used in the optimal solution for the first $n - r$ vertices. The greedy rule implies that the largest independent set among the vertices in $R_i$ is of size $|X_i|$. To color all but $|R_i|/2 \geq r$ vertices of $R_i$, the optimal sum coloring uses at least $|R_i|/(2|X_i|) = p_i/2$ colors. Thus, to color all but $r$ vertices of the whole graph, the optimal solution must use at least $p_i/2$ colors. Hence, the height of the column containing $Q$ in the optimal histogram is at least $p_i/2$, and $Q$ is therefore contained inside the optimal histogram. This completes the proof of 4-approximation.

A construction is given in [5] showing that the performance ratio of Greedy is in fact no better than 4.

# 4 Slicing and Dicing using Randomized Geometric Choices

**A two-person game** We will introduce the technique as the following two-person game. Player 1 (the adversary) writes down a secret, a positive real $z \geq 1$, on a piece of paper and folds it. Player 2 (the algorithm) then produces a series of guesses $x_0, x_1, \ldots$ until a guess exceeds $z$, i.e., $x_k \geq z$, for some guess $k$. The goal of the algorithm is to minimize the total sum of the guesses, *relative* to the value of the secret: minimize $\frac{\sum_{i=0}^{k} x_i}{z}$. As the algorithm must perform well on every $z$, it effectively seeks to minimize $\max_z \frac{\sum_{i=0}^{k} x_i}{z}$.

The natural deterministic strategy is the classic doubling trick: select $x_i = 2^i$, for $i = 0, 1, \ldots$. This strategy has a worst case *competitive* ratio of 4. The worst case occurs when $x_{k-1}$ just barely undershoots $z$, in which case the final guess $x_k$ overshoots by a factor of 2; the other factor of 2 comes from the cost of the previous guesses $\sum_{i=0}^{k-1} x_k$ in comparison with $x_k$. This turns out to be the best possible deterministic strategy [33].

With randomization against an oblivious adversary (one that fixes its choice independent of the random bits of the algorithm), we can do better, obtaining an expected competitive ratio of $e \sim 2.781$. The idea is to average out the cost of overshooting by randomly selecting a starting guess; to do so optimally, we additionally change the base from 2 to $e$. Choose $\alpha$ uniformly at random from $[0, 1)$, and select guesses $x_i = e^{i+\alpha}$. The last guess is the smallest $k$ for which $x_k = e^{k+\alpha} \geq z$, or $k + \alpha \geq \ln z$. The random variable $k + \alpha - \ln z$ is distributed like $\alpha$, uniformly in $[0, 1)$, and thus $x_k/z = e^{k+\alpha-\ln z}$ has expected value $E(\alpha) = e - 1$. This means that in expectation, the overshoot is a factor of $e - 1$. The previous guesses contribute a further multiplicative factor of $e/(e-1)$, since $\sum_{i=0}^{k} x_k = \frac{e^{k+\alpha+1}-1}{e-1} < \frac{e \cdot e^{k+\alpha}}{e-1} = \frac{e}{e-1} \cdot x_k$.

This technique is often applied when the guesses and the answer must be integers; it is then simple to just round the guesses down.

**Using the game for sum (multi)coloring** An easy application of this game is in solving `npSMC` on easily colorable graphs. Let us illustrate it on bipartite graphs, using the *color requirement* of each vertex as the value used in the game. We obtain subgraphs $V_0, V_1, \ldots$, where $V_i$ consists of the vertices with color requirement in $(x_{i-1}, x_i]$. We then two-color the underlying subgraph induced by $V_i$ (i.e., compute a bipartition), giving a multicoloring of the subgraph using at most $2x_i$ colors. Finally, the colorings of the subgraphs are concatenated to form the final schedule.

The doubling strategy then ensures that *each vertex* $v$ is fully colored within the first $8x(v)$ colors. This is obviously an 8-approximation, since the vertex cannot be completed before color $x(v)$. By using the randomized strategy, the *expected cost for each vertex* is $2e \cdot x(v)$, resulting in a coloring of expected cost $2e \cdot \mathcal{S}(G) = 2\sum_{v \in V(G)} x(v)$. (It can also be derandomized.) By a finer analysis and by adjusting the multiplier appropriately, the bound can be improved to $2.796\mathcal{S}(G)$ [6].

**Ways of applying the technique** The first application of this technique on sum coloring problems was the above application for `npSMC` of bipartite graphs, using the color requirements. The slicing technique can, perhaps surprisingly, be used in two additional ways.

Another way is to use the fractional values of a linear programming relaxation. This has resulted in $2e$-approximation for `pSMC` on perfect graphs, a 7.7-approximation for `npSMC` for line graphs and 11.3-approximation on interval graphs, and 3.6-approximation for `SC` on perfect graphs [16].

The third way is to use a subroutine for the *Maximum induced k-colorable subgraph* problem, letting $x_i$ denote the value of $k$ in the $i$-th call. This problem is polynomially solvable in interval graphs and comparability graphs (see [26]). That leads to constant-approximation for `SC` on interval, comparability, co-comparability, trapezoid and permutation graphs [26].

Interestingly, all three variations of the technique were applied in [12] to the same problem, which is a variant of `npSMC` where jobs are run in batches and no new job can be started before the whole previous batch completes.

# 5 Approximation Schema via Instance-Sensitive Partitioning

The randomized geometric partitioning technique is oblivious in that the actual distribution of values in the instance does not affect the partitioning. For finer approximation, we may need to construct the partition more carefully. Let us examine when we can hope to obtain a polynomial-time approximation scheme (PTAS).

As before, we aim to find breakpoints $b_1, b_2, \ldots$, that define subproblems $V_1, V_2, \ldots$, where $V_i$ consists of vertices with color requirements in the range $(b_{i-1}, b_i]$ (with $b_0 = 0$). We compute a schedule for each $V_i$, and concatenate them in sequence. That is, if we use $c_i$ colors for subproblem $V_i$ and denote $C_i = c_1 + c_2 + \cdots + c_i$, then we schedule the coloring of $V_i$ during rounds $C_{i-1} + 1, C_{i-1} + 2, \ldots, C_{i-1} + c_i$.

The impact of selecting a particular breakpoint $b_i$ defining a subproblem $V_i$ is that it delays the rest of the vertices (with color requirement larger than $b_i$). We can compute this *delay cost* as $Delay(b_i) = g(b_i) \cdot C(V_i)$, where $g(b) = |\{v : x(v) > b\}|$ equals the number of vertices with larger color requirement and $C(V_i)$ is the length of the schedule computed for subproblem $V_i$. The total delay cost of a sequence of breakpoints is then $Delay = \sum_i Delay(b_i) = \sum_{i=1} g(b_i) \cdot C(V_i)$. The key question is how large $Delay$ is compared with the cost of the optimal solution $Opt$; as a simple bound, we can use that it is at least the sum of the optimal costs of the subproblems, $Opt \geq \sum_i Opt(V_i)$. To this end, we need a schedule that is *short*, i.e., of small makespan, while at the same time, the coloring on each subproblem must also be efficient, with a good approximation of the chromatic sum. We therefore need a *simultaneous approximation* of the sum and the makespan objectives.

Recall that $\mathcal{S}(G) = \sum_{v \in V} x(v)$ denotes the sum of color requirements of a graph $G$ and $p$ denotes the largest color requirement.

The following result achieves the desired partitioning.

**Proposition 5.1 (The breakpoint lemma [24])** *Let $X = \{x_1, \ldots, x_m\}$ be a set of non-negative reals, and let $q$ be a number. Let $g(x)$ be the number of elements in $X$ greater or equal to $x$. Then, there is a sequence of integral breakpoints $b_i$, $i = 1, 2, \ldots$, with $\sqrt{q} \leq b_{i+1}/b_i \leq q$, such that $\sum_{i=1}^{m} g(b_i) \cdot b_i \leq \frac{1}{\ln \sqrt{q}} \mathcal{S}(X)$. Additionally, this can be achieved with a polynomial time algorithm.*

With the right algorithm for scheduling the subproblems, we can obtain a good solution for the whole.

We say that an instance is *p-bounded* if all color requirements are at most $p$. The following result follows easily from the breakpoint lemma, since the delays amount to at most $O(1/\log p)Opt$.

**Corollary 5.2** *Suppose there is an efficient algorithm for a sum multi-coloring problem on p-bounded graphs in graph class $\mathcal{G}$ that: a) approximates the sum within $1 + O(1/\log p)$-factor, and b) uses at most $O(p)$ colors. Then, we can approximate the sum multi-coloring problem on all graphs in $\mathcal{G}$ within factor $1 + O(1/\log p)$.*

We give examples of this approach on three different graph classes.

## 5.1 Bounded Treewidth Graphs

Graphs of small treewidth are a good example of graphs that can be handled efficiently, by efficient solution of $p$-bounded graphs and classical *rounding-and-scaling*.

To begin with, $p$-bounded instances can be handled exactly.

**Theorem 5.3** [24] *For any integers $k$ and $p$, the* `npSMC` *problem on $p$-bounded graphs with treewidth $k$ admits an exact algorithm that runs in time $O(n \cdot (k \cdot p \cdot \log n)^{k+1})$.*

This result follows from standard techniques once it is established that the number of colors in optimal sum multicolorings is $O(kp \log n)$.

Suppose we are aiming for a $1+\epsilon$-approximation. We round the color requirements up to the nearest multiple of $q = \lfloor \epsilon p / n^2 \rfloor$, and produce an $n^2/\epsilon$-bounded instance $I'$ where all lengths are scaled down by a factor of $q$. We can solve it exactly in $(n/\epsilon)^{O(k)}$ time and scale back the solution to apply to the original instance. The cost formed by the rounding can be shown to increase the sum cost by at most $1+\epsilon$-factor.

Since the complexity is polynomial in $n$ and $1/\epsilon$, we obtain an FPTAS.

**Corollary 5.4** `npSMC` *admits an* FPTAS *on graphs of constant treewidth.*

## 5.2 Planar Graphs

The starting point of most algorithms for planar graphs is the following classic result.

**Theorem 5.5 (Baker's decomposition [3])** *Given a graph $G = (V, E)$ and integer $k$, we can find in polynomial time a subset $U \subseteq V$ with at most $n/k$ vertices inducing a treewidth-2 graph, such that the graph $G'$ induced by $V \setminus U$ is of treewidth at most $k$.*

Our approach is to use the breakpoint lemma, and apply the following $1+\epsilon$-approximate algorithm on each subgraph $V_i$. Let $h$ be such that $2(5 + 8 \lg h)/h = \epsilon/2$.

**Step 1:** Partition the vertices of $V_i$ into $U_i$ and $V_i \setminus U$, as per Thm. 5.5, with $k = \lceil hp \rceil$.

**Step 2:** Compute an FPTAS solution of $G'_i$, the graph induced by $V_i \setminus U$, as per Cor. 5.4.

**Step 3:** Let $W$ be the set of vertices with color $(1 + 8 \lg h)p$ and larger in the solution of $G'_i$. Color the subgraph induced by $U \cup W$ using $4p$ colors (using that the underlying graph is planar, and therefore 4-colorable), and append it to the coloring of $V_i \setminus (U \cup W)$.

It can be shown that after using $p(1 + 8 \lg h)$ colors, all but at most $Opt_i/(hp)$ vertices have been fully colored in the coloring of $G'_i$, where $Opt_i$ is the optimal multi-chromatic sum of $V_i$. Thus, $W$ contains at most $Opt_i/(hp)$. Also, by Thm. 5.5 and the value of $k$, $|U| \leq Opt_i/(hp)$. The cost of coloring $U \cup W$ is at most $(4 + 1 + 8 \lg h)p \cdot |U \cup W| \leq 2(5 + 8 \lg n)/h \cdot Opt_i = \epsilon Opt_i/2$. Hence, the resulting coloring of $V_i$ is a $1+\epsilon$-approximation. Combined with the breakpoint lemma, we obtain a PTAS for (unrestricted) planar graphs.

### 5.3 Generalization to Minor-Free Graphs

let $H$ be a fixed graph (whose size is viewed as a constant). To *contract* an edge $(u, v)$ in an undirected graph is to merge $u, v$ into a single vertex whose neighborhood is $N(u) \cup N(v) \setminus \{u, v\}$. $H$ is a *minor* of $G$ if it can be derived from $G$ by a sequence of vertex and edge deletions and contractions of edges.

**Definition 5.1** *A graph is $H$-minor-free iff $H$ is not a minor of $G$*

**Theorem 5.6 ([11])** *For every $H$-minor-free graph and integer $k$, there is a constant $c = c_H$ such that the vertices can be partitioned to $k+1$ disjoint sets so that every $k$ of the sets induce a graph with treewidth bounded by $c_h \cdot k$. Furthermore, such a partition can be found in polynomial time.*

As graphs with constant treewidth are $O(1)$-colorable, the above is exactly what we need for the proof. Hence:

**Theorem 5.7** `npSMC` *on $H$-minor-free graph admits a* PTAS*, for any fixed graph $H$.*

## 6 Delaying Large Jobs: Paying Your Dues

### Sum Coloring $k+1$-claw-free Graphs

In non-preemptive problems, a large job once selected must be processed without preemption, and can thereby cause delay for many small jobs. One useful algorithmic strategy is therefore to delay the large jobs. We consider here a delay technique that results in a $2k(2k-1)$-approximation for `npSMC` on $k+1$-claw free graphs. For simplicity we assume all color requirements are distinct, resolving ties arbitrarily.

The idea is as follows. We form a new color requirement $y(v) = (\beta+1) \cdot x(v)$ for each vertex $v$, where $\beta$ is a parameter to be determined. During the first $\beta x(v)$ rounds that $v$ is selected it *waits*, and only becoming *active* once it is selected for the $\beta x(v)+1$-th time. Note that neighbors cannot both be selected in the same round, and, e.g., cannot both be waiting. Note that the rounds of waiting can be non-consecutive, while the active rounds are consecutive. When selecting vertices, our algorithm gives preference to jobs with small length (namely, small $x(v)$).

1. Let $I \leftarrow \emptyset$, $j \leftarrow 1$, $I_1 \leftarrow \emptyset$
2. **While $G \neq \emptyset$ do:**
   (a) Let $I_j$ be the set of active vertices that are not fully processed.
   (b) Iteratively add to $I_j$ the vertex of smallest $x(v)$ among vertices with no neighbors in $I_j$, until $I_j$ is a maximal independent set
   (c) Give color $j$ to the vertices in $I_j$ and update the color requirements $y_v$.
   (d) Delete fully colored vertices
   (e) $j \leftarrow j+1$
3. Return the resulting coloring

**Analysis** We shall show the following.

**Theorem 6.1** *The above algorithm yields a $2k \cdot (2k-1)$-approximation for* `npSMC` *on $k+1$-claw free graphs.*

This implies a 12-approximation for Line graphs, which are 3-claw free.

A vertex that is neither active nor waiting in a round (and has not completed its processing) is said to be *delayed*. This must be caused by neighbors that are either active or waiting in that round. It can only be delayed by waiting neighbors that are shorter, due to the rule of preference.

**Definition 6.1** *Let $N_s(v)$ be the neighbors of $v$ of smaller color requirement, and let $N_b(v)$ be those neighbors of $v$ that are scheduled to completion before $v$. Define $d_g(v) = \mathcal{S}(N_s(v))$ and $d_b(v) = \mathcal{S}(N_b(v))$ to be the total color requirements of these sets.*

Let $f(v)$ be the time (round) when our algorithm finishes scheduling vertex $v$. A part of $f(v)$ comes from the $(\beta+1)x(v)$ rounds in which $v$ is waiting or active. During other rounds, $v$ is delayed by its neighbors, either the shorter ones in $N_s$ or the longer ones in $N_b$. We say $v$ experiences a *good delay* if it is delayed by a shorter neighbor; otherwise, the delay is *bad* for $v$. More formally, if $I$ is the set of vertices active in a round, then $v$ experiences a good delay if $I \cap N_s(v) \neq \emptyset$, and bad delay otherwise (in which case $I \cap N_b(v) \neq \emptyset$).

Our main lower bound on the optimal cost *Opt* involves the following measure. Let $Q(G) = \sum_{uv \in E} \min\{x(v), x(w)\}$.

**Claim 6.2** $Opt \geq \mathcal{S}(G) + Q(G)/k$

**Proof:** For a vertex $v$, let $N^-(v)$ be the set of vertices that finish before $v$ in the optimal solution. Since the graph is $k+1$-claw free, at most $k$ vertices in $N^-(v)$ can be simultaneously active in any round. The finish time of $v$ in the optimal solution is then at least $x(v) + \lceil \frac{1}{k} \sum_{u \in N^-(v)} x(u) \rceil$. Summing over the vertices in the graph, the bound follows from the observation that $Q(G) = \sum_v \sum_{u \in N^-(v)} \min(x(v), x(u))$. □

We bound the good and bad delays separately. Bounding the former is straightforward.

**Claim 6.3** $\sum_v d_g(v) \leq (\beta+1) \cdot Q(G)$.

**Proof:** A vertex $v$ can experience at most $(\beta+1)\sum_{u \in N_s} x(u)$ good delay. Thus, the total good delay is at most $(\beta+1)\sum_v \sum_{u \in N_s} x(u) = (\beta+1)Q(G)$. □

The key idea is to bound the bad delays in terms of the good ones.

**Claim 6.4** $d_b(v) \leq \dfrac{(k-1) \cdot d_g(v)}{\beta - k + 1}$.

**Proof:** We say that a round is an *event for* $v$ if a neighbor of $v$ in $N_b$ is waiting. The total waiting of nodes in $N_b$ is $\sum_{u \in N_b} \beta \cdot x(u)$, and each such wait occurred during an event for $v$. Since the graph is $k+1$-claw free, at most $k$ neighbors of $v$ can be active or waiting in the same round. If some neighbor of $v$ is waiting, then $v$ is neither active nor waiting, which means that it was delayed either by a shorter neighbor or an active longer neighbor. Therefore, at most $k-1$ longer neighbors of $v$ can be waiting in a round, in particular during an event for $v$. Thus, there are at least

$$\sum_{u \in N_b} \frac{\beta \cdot x(u)}{k-1} \geq \beta \cdot \frac{d_b(v)}{k-1}$$

events for $v$. Since $v$ is delayed during an event, there are at most $d_g(v) + d_b(v)$ events for $v$ Combining the bounds on the number of events, we have that $d_g(v) + d_b(v) \geq \beta \cdot d_b(v)/(k-1)$. The claim follows by rearranging the expression. □

**Proof of Thm. 6.1:** Set $\beta = 2(k-1)$. Then, $d_b(v) \leq d_g(v)$, by Claim 6.4. The finish time of vertex $v$ by our algorithm is then $x(v) + d_g(v) + d_b(v) \leq x(v) + 2d_g(v)$. Summing over the vertices, the cost $Alg$ of the algorithm's schedule is bounded by $Alg - \mathcal{S}(G) \leq 2 \cdot (2k-1)Q(G) \leq 2(2k-1) \cdot k(Opt - \mathcal{S}(G))$, by Claims 6.3 and Claim 6.2. □

# 7 Open problems

In addition to improving the results in Table 2 consider the following open problems:

1. Does `npSMC` admits a constant ratio on *chordal* graphs? Maybe even on *perfect graphs*?
2. What is the ratio of `npMC` on planar graphs?
3. Give a a significant lower bound on approximating Open Shop scheduling.

# References

[1] K. Appel and W. Haken. Every planar map is four colorable. Bulletin of the American Mathematical Society **82**(5):711–712, 1976.

[2] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.* **23**(1):11–24, 1989.

[3] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM* **41**:153–180, Jan. 1994.

[4] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, T. Tamir. On chromatic sums and distributed resource allocation. *Inf. Comp.* **140**:183–202, 1998.

[5] A. Bar-Noy, M. M. Halldórsson and G. Kortsarz. A Matched Approximation Bound for the Sum of a Greedy Coloring. *Information Processing Letters* **71**(3–4):135–140, 1998.

[6] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, H. Shachnai, and R. Salman. Sum multicoloring of graphs. *J. Algorithms* **37**(2):422–450, 2000.

[7] A. Bar-Noy and G. Kortsarz. The minimum color-sum of bipartite graphs. *J. Algorithms* **28**:339–365, 1998.

[8] A. Borodin, I. Ivan, Y. Ye and B. Zimny. On sum coloring and sum multi-coloring for restricted families of graphs. *Theor. Comput. Sci.* **418**:1–13, 2012.

[9] A. L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold and M. Thorup. OPT versus LOAD in Dynamic Storage Allocation. *SIAM Journal on Computing* **33**(3):632–646, 2004.

[10] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling File Transfers. *SIAM Journal on Computing* **14**(3):744–780, 1985.

[11] E. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. Proc. *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 637–646, 2005.

[12] L. Epstein, M. M. Halldórsson, A. Levin and H. Shachnai. Weighted Sum Coloring in Batch Scheduling of Conflicting Jobs *Algorithmica* **55**(4):643–655, 2009.

[13] U. Feige. Approximating Maximum Clique by Removing Subgraphs *SIAM J. Discrete Math.* **18**(2):219–225.

[14] U. Feige and J. Kilian. Zero Knowledge and the Chromatic number. *Journal of Computer and System Sciences* **57**(2):187–199, October 1998.

[15] U. Feige, L. Lovász and P. Tetali. Approximating Min Sum Set Cover. *Algorithmica* **49**(4):219–234, 2004.

[16] R. Gandhi, M. Halldórsson, G. Kortsarz and H. Shachnai. Improved Bounds for Sum Multicoloring and Weighted Completion Time of Dependent Jobs. *ACM Transaction on Algorithms* **4**(1), 2008.

[17] R. Gandhi, M. Halldórsson, G. Kortsarz and H. Shachnai. Corrigendum: Improved results for data migration and open shop scheduling. *ACM Transactions on Algorithms* **9**(4):34, 2013.

[18] R. Gandhi, M. Halldórsson, G. Kortsarz and H. Shachnai. Improved results for data migration and open shop scheduling. *ACM Transactions on Algorithms* **2**(1):116–129, 2006.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[20] F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM J. Comput.* **1**(2):180–187, 1972.

[21] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1993.

[22] I. Holyer. The NP-completeness of Edge-Coloring. *SIAM J. Comput.* **10**(4), 1981.

[23] M. M. Halldórsson. A Still Better Performance Guarantee for Approximate Graph Coloring. *Inf. Process. Lett.* **45**(1):19–23, 1993.

[24] M. M. Halldórsson and G. Kortsarz. Tools for multicoloring with applications to planar graphs and partial $k$-trees. *J. Algorithms* **42**(2), 334–366, 2002.

[25] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multi-coloring trees. *Inf. Computation*, **180**(2):113–129, 2003.

[26] M. M. Halldórsson, G. Kortsarz, H. Shachnai. Sum coloring interval and $k$-claw free graphs with application to scheduling dependent jobs. *Algorithmica* **37**:187–209, 2003.

[27] M. M. Halldórsson, G. Kortsarz and M. Sviridenko. Sum edge coloring of multigraphs via configuration LP. *ACM Transactions on Algorithms* **7**(2):22, 2011.

[28] K. Jansen. The optimum cost chromatic partition problem. In *Proc. Third Italian Conference on Algorithms and Complexity (CIAC '97)*, LNCS 1203, pages 25–36, 1997.

[29] Y. Kim. Data migration to minimize average completion time. *Algorithmica* **63**(1–2):347–362, 2012.

[30] E. Kubicka. The chromatic sum of a graph. PhD thesis, Western Michigan University, 1989.

[31] E. Kubicka, G. Kubicki, and D. Kountanis. Approximation Algorithms for the Chromatic Sum. In *Proc. First Great Lakes Computer Science Conference*, LNCS 1203, pages 15–21, 1989.

[32] A. Kovács. Sum-multicoloring on paths. In *Proc. 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 68–80, 2004.

[33] Y. Kortsarts and J. Rufinus. Teaching the power of randomization using a simple game. In *Proc. 39th SIGCSE Technical Symposium on Computer Science Education, (SIGCSE)*, pages 460–463, 2006.

[34] M. Kchikech and O. Togni. Approximation Algorithms for Multicoloring Planar Graphs and Powers of Square and Triangular Meshes. *Discrete Mathematics and Theoretical Computer Science* **8**(1):159–172, 2006.

[35] M. Malafiejski. The complexity of the chromatic sum problem on cubic planar graphs and regular graphs. *Electronic Notes in Discrete Mathematics*, **8**, May 2001.

[36] M. Malafiejski, K. Giaro, R. Janczewski, and M. Kubale. Sum coloring of bipartite graphs with bounded degree. *Algorithmica* **40**(4):235–244, 2004.

[37] D. Marx. Minimum sum multicoloring on the edges of trees. *Computational Complexity* **14**(4):308–340.

[38] C. McDiarmid and B. Reed. Channel assignment and weighted coloring. *Networks* **36**(2):114–117, 2000.

[39] T. Nishizeki and K. Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM Journal on Discrete Mathematics* **3**(3):391–410, 1990.

[40] L. Narayanan and S. Shende. Static frequency assignment in cellular networks. *Algorithmica* **29**(3), 396–401, 2001.

[41] S. Nicoloso, M. Sarrafzadeh and X. Song. On the sum coloring problem on interval graphs. *Algorithmica* **23**:109–126,1999.

[42] M. Queyranne and M. Sviridenko. Approximation Algorithms for Shop Scheduling Problems with Minsum Objective. *Journal of Scheduling* **5**:287–305, 2002.

[43] D. Marx. A short proof of the NP-completeness of minimum sum interval coloring. *Oper. Res. Lett.* **33**(4):382–384, 2005

[44] V. G. Vizing. On the estimate of the chromatic class of $p$-graphs. *Diskret. Analiz.* **3**:23–30, 1964.

[45] G. Woeginger. Private communication, 1997.