# Batch Coloring Flat Graphs and Thin

Magnús M. Halldórsson[*]         Hadas Shachnai[†]

### Abstract

Batch scheduling of conflicting jobs is modeled by *batch coloring* of a graph. Given an undirected graph and the number of colors required by each vertex, we need to find a proper batch coloring of the graph, namely, to partition the vertices to batches which are independent sets and assign to each batch a contiguous set of colors, whose size is equal to the maximum color requirement of any vertex in this batch. When the objective is to minimize the sum of job completion times, we get the *batch sum coloring* problem; when we want to minimize the maximum completion time of any job (or, the *makespan*) we get the *max coloring* problem.

Given the hardness of batch coloring on general graphs, already for the special case of unit color requirements (known as *sum coloring* and the classic *graph coloring* problem, respectively), it is natural to seek out classes of graphs where effective solutions can be obtained efficiently. In this paper we give the first *polynomial time approximation schemes* for batch sum coloring on several classes of "non-thick" graphs that arise in applications. This includes paths, trees, partial $k$-trees, and planar graphs. Also, we give an improved $O(n \log n)$ exact algorithm for the max-coloring problem on paths.

## 1   Introduction

In the classic *p-batch* scheduling problem [8], given is a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, where $J_j$ has processing time, or *length*, $p_j$. We need to partition the jobs to subsets, such that in each subset we *batch* all the jobs to be processed jointly, that is, all the jobs in the batch start at the same time; a batch is completed when the last job in the batch finishes its processing.[1]

Many real-life scenarios impose restrictions on the subsets of jobs that can be processed simultaneously. Such conflicts among the jobs are often modeled by an undirected conflict graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$. The length of vertex $j$ is the processing time of $J_j$; there is an edge $(i, j) \in E$ if the pair of jobs $J_i$ and $J_j$ cannot be processed in the same batch. In each time period we can process a *batch* of jobs that forms an independent set in $G$. Indeed, scheduling the jobs in $\mathcal{J}$ is equivalent to finding a proper *batch coloring* of the graph $G$, where we need to assign to each batch a distinct contiguous set of colors, whose size is equal to the maximum length of any vertex in the batch.

We consider the above problems of batch scheduling with conflicts under three different measures. In the *max-coloring (*MAX-COL*)* problem we minimize the *makespan* of the batch schedule, or the sum of the batch lengths. In the *minimum sum of job completion times problem* (SJC), we minimize the sum of the completion times of the jobs, and in the *minimum sum of batch completion*

[1]We refer to the *unbounded* version of the problem.

*times problem* (SBC), we minimize the sum, over all the jobs $J_j$, of the completion time of the batch that contains $J_j$.

When all jobs have unit processing times, MAX-COL reduces to the classical problem of finding a proper vertex coloring of a graph using the minimum number of colors, while SJC and SBC reduce to the *sum coloring* problem [3].

In this paper we give the first *polynomial time approximation schemes (PTAS)*, as well as exact algorithms, for the above problems on several subclasses of graphs that arise in applications. This includes paths, trees, partial $k$-trees, planar and, more generally, "flat" graphs (see in Section 3).

We say that an approximation algorithm $\mathcal{A}$ yields an approximation ratio of $r$, for some $r \geq 1$, if for any instance $I$ of our problems $\mathcal{A}(I) \leq r \cdot OPT(I)$, where $OPT(I)$ is the value of an optimal solution. A problem admits a PTAS if it can be approximated in polynomial time within factor $1 + \varepsilon$, for any constant $\varepsilon > 0$. When the complexity of the scheme is of the form $f(\epsilon)n^c$, where $c > 0$ is some constant, we get an *efficient PTAS (EPTAS)*.

## 1.1 Motivation

We list below several natural applications of our problems.

**Metropolitan Networks:** Transmission of real-time messages in metropolitan networks is done by assigning to each sender node a set of fixed length slots, in which the message content is filled and transmitted in sequence to the receiver. To improve transmission times of the messages, the same slots can be assigned to messages with non-intersecting paths [21]. A group of messages using the same set of slots can be viewed as a batch. The number of slots in the set is the maximum length of any message transmitted in these slots. The problem of minimizing the number of slots used to transmit all messages then yields an instance of MAX-COL.

**Distributed Computation:** In distributed operating systems (see, e.g., [28, 22]) the scheduler identifies subsets of non-conflicting, or cooperating processes, that can benefit from running at the same time interval (e.g., since these processes do not use the same resources, or communicate frequently with each other); then, each subset is executed simultaneously on several processors, until *all* the processes in the subset complete. When the objective is to minimize the sum of job completion times, we get an instance of SJC; when we want to minimize the total completion time of the schedule, we get an instance of MAX-COL.

**Batch production:** A manufacturing system consists of a set of production lines, to be loaded with raw material for processing/assembling the various products. A subset of the products can be batched and manufactured in parallel in a certain shift if each uses a distinct set of production lines. Suppose we are given a list of orders for products, and we want to partition the orders to batches, to be run in a sequence of shifts. We can model the problem as an undirected graph, where each vertex represents an order (=product), and two vertices are adjacent if the corresponding products share the use of certain production line. When the goal is to find a production schedule which minimizes the average completion time of an order, we get an instance of SJC (or SBC, depending on whether an order can be completed before its batch completes); when we need to minimize the total completion time of the schedule, this yields an instance of MAX-COL.

Other applications of our problems include, e.g., dynamic storage allocation and memory management in wireless protocol stack (see, e.g., [18, 26].)

The graph classes that we study here arise in the above applications from tree-like (or planar) network topologies [23] ([24]), conflict graphs of processes generated for computer programs [28],

and scheduling jobs with bounded resource requirements (which yield *pebbly* graphs; see in Section 3.4).

## 1.2   Related Work

There is a wide literature on batch scheduling, namely, our problems with empty conflict graph. (A comprehensive survey of known results for these problems is given, e.g., in [8].) In the following we mention known results for our problems on various classes of conflict graphs.

**Batch sum coloring:**   The problem of batch coloring so as to minimize the sum of job completion times was introduced in [4]. For general graphs, both SBC and SJC are hard to approximate within factor $n^{1-\varepsilon}$, for any $\varepsilon > 0$, unless NP=ZPP. This follows from the hardness of the sum coloring problem [3, 16], which is a special case of our problems. The paper [4] gives constant approximation algorithms for SJC on several classes of graphs including perfect graphs, bipartite graphs, $k$-colorable graphs and line graphs. The paper [14] improves the approximation ratios for these classes, as well as for $(k+1)$-claw free graphs, and interval graphs. It is also shown in [14] that SBC can be approximated within factor $4\rho$ for any graph class on which the maximum independent set problem can be approximated within factor $\rho$, for some $\rho \geq 1$.

**Max coloring:**   The max-coloring problem was first considered by Guan and Zhu [18], who gave a polynomial time algorithm for max-coloring trees with fixed number of colors (see also [26]). They also gave an $O(n^4)$-time algorithm for paths, while a quadratic algorithm was presented more recently in [15]. In this paper we give an improved $O(n \log n)$-time algorithm for the problem. A PTAS is known for trees [26, 15] and for partial $k$-trees [15]. MAX-COL is known to be NP-hard for bipartite graphs, line graphs of bipartite graphs and split graphs, [11] and for interval graphs [15]. Constant factor approximation algorithms have been given for bipartite graphs [26], planar graphs [15], interval graphs [25], and perfect graphs [26, 13]

**Multicoloring problems:**   Our batch coloring problems with minsum objective relate closely to the *non-preemptive sum multicoloring (*NPSMC*)* problem, defined as follows. We are given a graph $G = (V, E)$, where each vertex $v$ is associated with a color requirement $x(v) \geq 1$. We need to find a proper multicoloring of the vertices in $G$ such that each vertex $v$ is assigned $x(v)$ distinct contiguous colors, and the sum of the highest colors assigned to the vertices is minimized. Formally, let $f(v) = \max\{i \geq 1 : v \text{ is colored with } i\}$ be the highest color assigned to $v$, then we want to minimize $\sum_{v \in V} f(v)$, such that any two adjacent vertices are assigned disjoint sets of colors. We note that in batch coloring, we take a non-preemptive any batch schedule is also a non-preemptive coloring. Denote by NPSMC$(G)$, SJC$(G)$, SBC$(G)$ the optimal values for the corresponding problems on a given graph $G$. Then, we get that

$$\text{NPSMC}(G) \leq \text{SJC}(G) \leq \text{SBC}(G).$$

For a comprehensive survey of known results for multicoloring problems with minsum objectives see, e.g., [17]. The common special case where $x(v) = 1$ for all $v \in V$, known as the *sum coloring (*SC*)* problem, has been studied extensively (see, e.g., [3, 14, 20]). Halldórsson and Kortsarz [19] gave a PTAS for SC on planar graphs. We present an EPTAS with improved running time.

## 1.3   Our Results

In this paper we focus on batch coloring of several amenable classes of graphs that we could term either "flat" or "thin" (see in Section 3). Given the hardness of these problems on general graphs,

it is natural to seek out classes of graphs where effective solutions can be obtained efficiently. Let BSC refer to either of the batch scheduling problems under the sum measure: the sum of batch completion times, or job completion times.

**Batch coloring with minsum objective:** In Section 2 we describe our main approximation technique. In Section 3 we develop EPTASs for BSC on partial $k$-trees and later for planar graphs. By a result of [5, 9], this implies that BSC is fixed parameter tractable on these classes of graphs.[2] Our scheme for planar graphs improves the running time of a PTAS proposed in [19] for the special case of the sum coloring problem to a linear time EPTAS. In Section 3.3 we show how our results for planar graphs can be extended to other "flat" graphs. To the best of our knowledge, we give here the first approximation schemes for batch sum coloring on partial $k$-trees and planar (or more generally, "flat") graphs.

**Max coloring:** In Section 4 we give an $O(n \log n)$ exact algorithm for MAX-COL on paths. This improves upon the $O(n^2)$ algorithm of [15].

**Contribution:** Our main approximation technique combines technical tools developed in [19] for approximating NPSMC with efficient enumeration of batch length sequences. A key component in our technique is truncation of the number of batches in the schedule, so that all possible batch length sequences can be enumerated in polynomial time. This defines a general framework for solving batch coloring problems with minsum objective, for any graph class for which (*a*) the number of batches can be truncated with small harm to the objective function, and (*b*) given a sequence of batch lengths, a proper batch coloring with minimum total cost can be found efficiently.

A framework proposed earlier, for approximately solving the max coloring problem on certain classes of graphs, shares some similarities with ours (see [18, 26]); however, BSC differs from MAX-COL in two ways. (*i*) For many classes of graphs, we can bound the total number of batches used in a MAX-COL schedule using structural properties of the graph (e.g., the maximum degree of any vertex), while for batch coloring with *minsum* objective this number may depend on the distinct number of job lengths (see Lemma 1). (*ii*) Given a batch length sequence, the problem of finding a MAX-COL schedule reduces to finding a *feasible* batch coloring, while solving BSC involves optimizing over the set of feasible schedules. Thus, our BSC problems require the usage of different machinery.

We expect that our framework for solving BSC will find more uses for batch coloring with other (more general) minsum objective functions, as well as for solving BSC on other classes of graphs. In fact, as we show in Section 3, the approximation technique that we use for planar graphs is general enough to be applicable to graph classes that contain planar graphs as a subclass (such as bounded-genus graphs).

**Notation** Let $p(G) = \sum_{j \in V} p_j$ be the sum of the job lengths. Let $p_{max} = p_{max}(G) = \max_{j \in V} p_j$ be the maximum job length, $p_{min} = \min_{j \in V} p_j$ be the minimum job length, and $\tau = \tau(G) = p_{max}/p_{min}$ be the maximum ratio between two job lengths. We omit $G$ when clear from context.

The *size* or *weight* of a set of vertices is the sum of the vertex lengths. A *batch* is an independent set that is to be scheduled starting at the same time. The *length* of a batch is the largest length of a vertex in the batch. The *density* of a batch $B$ of length $\ell$ is $|B|/\ell$, or the number of jobs in $B$ per length unit. A *(batch) schedule* is a partition of the vertices into a sequence of batches.

The completion time of a batch $B$ in a schedule is the sum of lengths of the batches up to and including $B$. The completion time of a job $J_j$ in batch $B$ is the sum of the lengths of batches prior

---

[2]For the recent theory of fixed-parameter algorithms and parameterized complexity, see, e.g., [12].

to $B$, plus the length of $J_j$.

The *makespan* of a schedule is the completion time of the last job, or the sum of the lengths of all batches in the schedule. Let $\mu(G)$ be the minimum makespan of a batch schedule of $G$.

Let $d$ denote the number of distinct vertex lengths and $\Delta$ the maximum degree of the graph.

## 2    Techniques

Batch coloring problems introduce new difficulties to the classic *multicoloring* and *batch scheduling* problems, which are known to be hard when solved alone (see in Section 1.2). The fact that all jobs in the same batch must start at the same time limits the number of different starting times in a schedule, and thus can simplify the search for good schedules. However, as opposed to ordinary multicoloring, batch coloring introduces *non-locality*: the allowable starting times of a job depends not only on this job and its neighbors, but also implicitly on the jobs elsewhere in the graph that have been assigned to earlier batches. This implies, for example, that there are almost no non-trivial results known on the sum of completion times problems, even on paths, nor is there a known exact algorithm for the max coloring problem on trees.

Two difficult features are inherited from related multicoloring problems. One is that a large number of batches may be needed even on low-chromatic number graphs. In fact, as we show in the next result, $\Omega(n)$ batches may be required for optimal SJC coloring of paths.[3]

**Proposition 1** *There are* SJC *instances on paths and* SBC *instances on empty graphs for which the only optimal solution uses $n$ batches.*

*Proof.* Consider a path of $n$ vertices where the length of vertex $j$ is $p_j = n^j$. We claim that the only optimal solution is given by the *shortest processing time first (SPTF)* rule; namely, batch $i$ contains only vertex $i$, for $i = 1, 2, \ldots, n$.

Suppose there is a non-SPFT optimal SJC schedule $S$, and let $j$ be the smallest number such that vertex $j$ is not assigned alone in batch $j$. Consider now the following change to $S$, where we create a new batch, number it $j$ (increasing the index of later batches) and reassign $v_j$ to batch $j$. This may increase the completion times of $v_s$, for $s = j + 1, \ldots, n$, by a total of $(n - j) \cdot n^j < n^{j+1}$. It will, however, decrease the completion time of $v_j$ by the length of some later vertex, or at least $p_{j+1} = n^{j+1}$. Thus, this new schedule improves upon $S$, which is a contradiction.

The same set of lengths yields the same argument for SBC, even if the graph contains no edges. ∎

The way to get around the large number of batches is to analyze the cost of truncating the coloring early, and show that there are approximate colorings that use moderately many colors. Alternatively, we can use standard rounding of job lengths to powers of $1 + \epsilon$ and bound the number of batches of each length.

Another difficulty has to do with the large range of job lengths. An essential ingredient in any strategy for these problems is a method to break the instance into sub-instances of similar-length jobs. For max coloring, using a fixed geometric sequence is sufficient to reduce the problem to the ordinary coloring problem, within a constant factor [26]. For our batch coloring problems with minsum objective, BSC, the situation is not as easy, and it is necessary to partition according to the actual length distribution.

---

[3]The paper [26] gives a similar result for the max coloring problem of bipartite graphs.

We observe that a result of [19], originally stated for sum multicoloring, holds also for BSC. It shows that we can focus on the case where the ratio between the longest and the shortest job is small, if we also bound the makespan of the algorithm.

**Theorem 2** *Let $n$, $q = q(n)$ and $\sigma$ be given. Suppose that for any $G = (V, E)$ in a hereditary class $\mathcal{G}$ with $n$ vertices and $\tau(G) \leq q$, we can approximate BSC within a factor $1 + \epsilon(n)$ and with makespan of $\sigma \cdot p_{max}(G)$ in time $t(n)$. Then, we can approximate BSC on any graph in $\mathcal{G}$ within factor $1 + \epsilon(n) + \sigma/\sqrt{\ln q}$ with makespan $2\sigma \cdot p_{max}(G)$ in $O(t(n))$ time.*

This is based on finding a partition of the instance into length classes that depends on the actual distribution of the lengths. Then, the solutions produced on each length class by the assumed algorithm are simply concatenated in length order. Thus, if the solution produced on each length class is a batch schedule, so is the combined solution.

The techniques that we use for BSC builds on a technique developed in [19] for non-preemptive sum multicoloring (NPSMC) of partial $k$-trees and planar graphs. For the batch problems studied here, we have modified and replaced some of the parts, including the bounds on the number of batches needed. Most crucially, the central subroutines for handling subproblems with a limited number of job lengths were completely changed. The strategy here is to decide first the length sequence of the batches to be used, and then assign the jobs near optimally to those batches. By rounding the job lengths, we limit the number of possible batch lengths and at the same time reduce the number of batches needed.

# 3 Batch Sum Coloring

In this section we give EPTASs for BSC on both "thin" graphs (trees, partial $k$-trees) and "flat" graphs (planar and bounded-genus graphs).[4] First, we argue some general properties of exact and approximate solutions. Recall that $p(G)$ is the sum of vertex length, $p_{max}$ the length of the longest job, $\mu(G)$ the minimum makespan of a batch schedule, and $\chi$ the chromatic number of the graph.

## 3.1 Properties and tools

**Observation 3.1** *Any optimal BSC schedule satisfies the following properties.*
*(i)* (Non-increasing density) *Batch densities are monotone non-increasing.*
*(ii)* (Density reduction) *After $i(2\mu(G) + p_{max})$ steps, the total length of the remaining graph is at most $p(G)/2^i$.*
*(iii)* (Restricted batch length sequences) *Each batch is preceded by at most $\Delta$ longer batches.*

*Proof sketch.* (*i*) The claim holds since, otherwise, batches can be swapped to decrease the cost of the schedule (also known as Smith's rule; see in [27, 8]). (*ii*) Density must go down by half each $2\mu(G) + p_{max}$ steps; otherwise, we could schedule the whole remaining graph in the latter $\mu(G)$ steps, at lesser cost. (*iii*) This is true since, otherwise, the vertices in that batch can all be recolored with earlier batches. ∎

We now give two lemmas referring to batch coloring of $\chi$-colorable graphs.

**Lemma 3** *Let $G$ be a $\chi$-colorable graph. Then, $\text{SJC}(G) \leq \text{SBC}(G) \leq 3\chi \cdot p(G)$.*

---

[4]Informally, a graph is "flat" if it can almost be embedded on a flat surface; a graph is "thin" if it has bounded treewidth. Formal definitions can be found, e.g., in [29].

*Proof.* Recall that $\tau$ is the maximal ratio between the processing times of any two vertices. Use a batch sequence with $\chi$ batches of each length $2^i p_{min}$, for $i = 1, 2, \ldots, \lceil \log \tau \rceil$, in non-decreasing order. Order the batches of the same length in non-increasing order of size. The length of a job is at least half the length of its batch. Each job waits for all batches shorter than it; also, averaged over the jobs in batches of the same length, it waits for $(\chi - 1)/2$ batches of length equal to its batch length. Then, on average, the completion time of $v$ is at most $p(v) + \chi \sum_{j=0} p(v)/2^j + (\chi - 1)/2 \cdot 2p(v) = 3\chi p(v)$. Thus, the total schedule cost is at most $3\chi p(G)$. $\blacksquare$

The following lemma, which extends a result of [19], helps rein in the total length of our approximate schedule.

**Lemma 4** *Let $G$ be a $\chi$-colorable graph, and let $\epsilon > 0$. Then, there exists a $(1 + \epsilon)$-approximate* Bsc *schedule of $G$ satisfying the following constraints:*

1. *Batch lengths are powers of $1 + \epsilon$,*

2. *There are at most $t = t_{\chi,\epsilon}$ batches of each length, where $t = \chi(1 + \epsilon)\epsilon^{-1}$, and*

3. *The makespan of the schedule is at most $(2\mu(G) + p_{max}) \cdot \log(\chi/\epsilon) + 2\chi \cdot p_{max}$*

*Proof.* First, we consider the effect of rounding all job lengths to powers of $1 + \epsilon$. This increases the size of each batch by at most a factor of $1 + \epsilon$, which delays the starting time of each job by factor at most $1 + \epsilon$. Thus, the extra cost incurred for the optimal schedule is at most $\epsilon \cdot (OPT(G) - p(G))$.

Next, consider the optimal schedule $S^*$ for the rounded instance. Suppose that more than $\chi \cdot (1 + \epsilon)\epsilon^{-1}$ batches are used for some batch length $\ell$. Then, following batch $\chi \cdot \epsilon^{-1}$ of length $\ell$, we introduce $\chi$ batches of length $\ell$, shifting all later batches by $\chi \cdot \ell$. We color all jobs that occurred in later batches of length $\ell$ using the $\chi$ new batches, and delete those later batches. Each batch in the resulting coloring is delayed by $\chi \cdot \ell$, by the new batches of length $\ell$, only if it was already preceded by $\epsilon^{-1}$ times that many batches. Thus, each batch in the resulting coloring is delay by at most an $\epsilon$-fraction of what previously came before it, or at most $\epsilon \cdot (OPT_J(G) - p(G))$.

Finally, we analyze the effect of cutting a schedule short. By the density reduction property, the total size remaining is at most $\epsilon p(G)/\chi$ after $i(2\mu(G) + p_{max})$ rounds with $i = \log(\chi/\epsilon)$. By Lemma 3, there is a schedule of the remainder of cost at most $3\chi \cdot \epsilon p(G)/\chi \leq 3\epsilon p(G)$. The makespan of that coloring is at most $2\chi \cdot p_{max}$. $\blacksquare$

## 3.2 Thin graphs

A $k$-tree is a graph that can be reduced to a clique of size $k$ by deleting iteratively some vertices the neighborhood of which is a clique of size $k$. A partial $k$-tree is a subgraph of a $k$-tree. Partial $k$-trees are (polynomially) $k + 1$-colorable.[5]

We round the job lengths, obtaining an instance with only a limited number of distinct lengths. There are $d$ possible lengths for each of the $b$ batches, for a total of $d^b$ distinct batch length sequences. We solve the problem optimally for each such length sequence, using standard dynamic programming. We sketch it briefly.

**Lemma 5** *Given a partial $k$-tree $G$ and a sequence of $b$ batch lengths, there is an $O(b^{k+1}n)$-time algorithm to find an optimal* Bsc *coloring of $G$ into those batches, or determine that no such coloring exists.*

---

[5]For definitions and some properties of partial $k$-trees, see, e.g., [7] and the references therein.

*Proof sketch.* For each bag in the tree decomposition[6] of $G$, form a table of $b^k$ $k$-tuples, where a given $k$-tuple represents a particular batch assignment of the jobs in the given bag, and the entry corresponds to the minimum cost schedule constrained to assign the $k$-tuple in this given way. By having the children update the entries of the parents, and using that adjacent bags need only differ in only a single element, each entry is needed for a constant-time update of at most $b$ entries in its parent bag. Hence, the complexity is $O(b^{k+1}n)$. ∎

We first argue a PTAS for instances of restricted job lengths.

**Proposition 6** *There is a $(1+\epsilon)$-approximate algorithm for* BSC *on partial $k$-trees, with time complexity $2^{\Theta((\log \chi + \log \epsilon^{-1}) \cdot (k + \chi \cdot \epsilon^{-2} \cdot \log \tau))} n$ and makespan of $O(\chi \log \epsilon^{-1} p_{max})$.*

*Proof.* We use Lemma 5 to search for restricted $(1+\epsilon)$-colorings guaranteed by Lemma 4. The time complexity of the algorithm of Lemma 5 is $O(b^{b+k}n)$. The number $d$ of different batch lengths is $\log_{1+\epsilon} \tau = \Theta(\epsilon^{-1} \log \tau)$. The number $t$ of batches of each length is $\Theta(\chi\epsilon^{-1})$. Thus, the number $b$ of batches is at most $t \cdot d = \Theta(\chi \cdot \epsilon^{-2} \log \tau)$. The time complexity is therefore bounded by

$$O(2^{\log b \cdot b} n) = 2^{\Theta((\log \chi + \log \epsilon^{-1}) \cdot \chi \cdot \epsilon^{-2} \cdot \log \tau)} n.$$

The makespan is as promised by Lemma 4, using that $\mu(G) \leq \chi \cdot p_{max}$. ∎

We now combine Proposition 6 with Theorem 2.

**Theorem 7** *There is an EPTAS for* BSC *on partial $k$-trees, for any fixed $k$.*

*Proof.* We set $\sigma = \chi \cdot (\log(\chi/\epsilon) + 2)$ and $q = e^{(2\sigma \cdot \epsilon^{-1})^2}$, giving $\sigma/\sqrt{\ln q} = \epsilon/2$. We use Lemma 5 to search for restricted $(1+\epsilon)$-colorings guaranteed by Lemma 4 with these parameters. Theorem 2 then yields a $(1+\epsilon)$-approximate schedule of $G$.

Let us evaluate the parameters according to this scheme. Since we may assume $\epsilon^{-1} \geq \chi$, we have $\sigma = \Theta(\chi \cdot \log \epsilon^{-1})$. Thus, $\log \tau = \log q = (2\sigma \cdot \epsilon^{-1})^2 = \Theta(\chi^2 \cdot \epsilon^{-2} \cdot \log^2 \epsilon^{-1})$. Hence, the time complexity of the algorithm of Proposition 6 is bounded by

$$2^{O((\log \chi + \log \epsilon^{-1}) \cdot (k + \chi^3 \cdot \epsilon^{-4} \cdot \log^2 \epsilon^{-1}))} n.$$

Since the chromatic number of a partial $k$-tree is at most $k+1$, the time complexity is singly exponential in $1/\epsilon$ and $k$. The makespan is at most $2\sigma p_{max} = O(k \cdot \log \epsilon^{-1} p_{max})$. ∎

**Parametrization:** While the existence of an exact algorithm for BSC remains open, it appears unlikely. We consider instead parameterizations that lead to efficient exact solutions. We treat the parameters $d$, the number of distinct job lengths, and $\Delta$, the maximum degree.

Consider graphs of maximum degree $\Delta$. Since there can be at most $\Delta$ batches of the same length in an optimal schedule, there are at most $d^{\Delta d}$ different batch length sequences. However, this does not take the property of restricted batch length sequences (property *(iii)* in Observation 3.1) into account. Thus, for example, in a batch length sequence for a path, each batch can be preceded by at most two longer batches. We can capitalize on this to obtain improved bounds. The proof of the following lemma is given in the appendix.

**Lemma 8** *The number of possible batch length sequences with $d$ distinct lengths is at most $2^{(\Delta+1)d}$.*

---

[6]A formal definition of *tree decomposition* is given, e.g., in [7].

By applying standard dynamic programming, we can therefore solve BSC efficiently on thin graphs of bounded-degree when the number of different lengths is bounded.

**Theorem 9** *There is a $2^{O(\Delta \cdot k \cdot d)} n$-time algorithm for BSC in partial $k$-trees of maximum degree $\Delta$ and $d$ different weights.*

## 3.3 Flat graphs

We first treat planar graphs, and then indicate how the approach can be generalized to other "flat" graphs.

We use a classical partitioning technique due to Baker [2]. See [19] for details on the following specific version. A $t$-outerplanar graph has treewidth at most $3t - 1$ [6].

**Lemma 10** *Let $G$ be a planar graph and $f$ be a positive integer. Then, there is a vertex-disjoint partitioning of $G$ into graphs $G_1$, which is $f$-outerplanar, and $G_2$, which is outerplanar with at most $2n/f$ vertices and a total vertex length of $2p(G)/t$.*

In view of Theorem 2, we focus on giving an efficient approximation in the case when the vertex lengths are within a limited range. Our method is similar to that of [19], but somewhat simpler. We can particularly take advantage of the feature of batch schedules that it is easy to insert color classes in between batches of a given schedule, which is not so easily done in non-preemptive multicoloring. The result is an EPTAS, as opposed to a PTAS of [19] for NPSMC, whose running time is of the form $(\log n)^{\epsilon^{O(1)}} n$.

**Lemma 11** *There is a $(1 + \epsilon)$-approximation algorithm for BSC on planar graphs, that runs in time $2^{O(\log \epsilon^{-1} \cdot \epsilon^{-2} \cdot \log \tau)} n$. The makespan of the schedule found is $O(\epsilon^{-1} \cdot p_{max})$.*

*Proof.* The algorithm proceeds as follows. Let $f$ be to be determined. We first apply Lemma 10 to partition $G$ into a partial $3f$-tree $G_1$ and a partial 2-tree $G_2$, where $G_2$ has at most $2n/f$ vertices and $2p(G)/f$ weight. We then find a $(1 + \epsilon)$-approximate schedule $S$ of $G_1$ using Proposition 6, and find a schedule $S_2$ of $G_2$ using Lemma 3. The issue that remains is how to insert the batches of $S_2$ into $S$ so as to limit the cost of the resulting schedule.

Let $z = d \cdot \epsilon^{-1}$, where $d = \lg \tau$ is the number of distinct batch lengths in $S_2$. Recall that there are at most 4 batches of each length in $S_2$. We shall fit the batches of each length $\ell$ in $S_2$ into the schedule $S$ as follows: Before the batch that starts execution at or right before step $i \cdot z \cdot \ell$, insert a batch of length $\ell$, for $i = 1, 2, 3, 4$. Then, each job in $S$ is delayed at most $i\ell$ by batches of length $\ell$ if its completion time in $S$ is at least $i \cdot z \cdot \ell$, or at most a $1/z$-fraction. Thus, summing over the different lengths, each job is delayed by at most a $d/z = \epsilon$-fraction by jobs from $S$. Now, each job in $S_2$ is delayed by at most a $z$ factor. So, the cost of scheduling the jobs of $G_2$ within the new schedule is at most

$$z \cdot p(G_2) \leq z \cdot p(G)/f,$$

which is at most $\epsilon p(G)$ if we choose $f$ to be $z/\epsilon = \epsilon^{-2} d$.

The combined cost of the coloring is then at most $1 + 3\epsilon$ times optimal. We scale $\epsilon$ to suit the claim. The makespan of the schedule is the sum of $O(\epsilon^{-1} \cdot p_{max})$ for scheduling $G_1$ and $O(p_{max})$ for $G_2$, as argued in Lemma 3, for a total of $O(\epsilon^{-1} p_{max})$.

The time complexity is dominated by the time used by the algorithm of Proposition 6 for $G_1$. Here, $\chi \leq 4$, but $k = O(f) = O(\epsilon^{-2} \log \tau)$, which is asymptotically equivalent to the number $b$ of batches. Hence, the combined time complexity is $2^{O(\log \epsilon^{-1} \cdot \epsilon^{-2} \cdot \log \tau)} n$. ∎

The following theorem now follows straightforwardly by combining Lemma 11 with Theorem 2. The time complexity is $2^{\Theta(\epsilon^{-4} \cdot \log^3 \epsilon^{-1})}n$.

**Theorem 12** *There is an EPTAS for* BSC *on planar graphs.*

Recall that sum coloring (SC) is the common special case of SJC and SBC when all vertices have the same length. It was shown in [19] that SC is strongly NP-hard for planar graphs. This implies that our results are best possible, in that no FPTAS is possible.

We can obtain a more efficient scheme in the case of small values of $p_{max}$, and in particular for the sum coloring problem. This improves on the algorithm of [19] that has time complexity $exp(O(\ln \ln n \cdot \epsilon^{-1} \log \epsilon^{-1})) \cdot n$.

**Theorem 13** *There is an EPTAS for* SC *on planar graphs with running time* $(\log \epsilon^{-1})^{O(\epsilon^{-1})}n$.

**Minor-closed graphs of locally bounded treewidth**  Two properties of planar graphs were essential for our strategy: The partition into outerplanar sets that allowed for a "shifting" strategy, and the bounded chromatic number.

Suppose we form the layers $L_0, L_1, \ldots, L_h$ where $L_i$ consists of nodes of level $i$ in a breadth-first search, starting at an arbitrary vertex. Demaine and Hajiaghayi [10] showed that the treewidth of any $r$ consecutive layers in a graph from a minor-closed class of graphs of locally bounded treewidth is at most $cr + d$, for some constants $c$ and $d$. For $j = 0, \ldots, r-1$, let $V_j = V \setminus \cup_i L_{i(r+1)+j}$. Then, one of these $V_j$ yields a $G_1$ satisfying Lemma 10, within constant factors. We can use an algorithm of Amir [1] to find $O(r)$-tree decompositions in time $2^{O(r)}n^2$. Recall that we need $r = \theta(\epsilon^{-2})$.

Such minor-closed classes are known to be exactly the apex-minor free classes, where a graph is apex if deleting one vertex produces a planar graph. This includes planar graphs, bounded-genus graphs, single-crossing-minor-free graphs, and single-crossing graphs. A general upper bound on the chromatic number of minor-free graphs is open, but for graphs of bounded genus $g$, Heawood's formula gives $\chi(G) = O(\sqrt{g})$ [29].

**Theorem 14** *There is an EPTAS for* BSC *on bounded-genus graphs.*

## 3.4   Pebbly graphs

As implied by Proposition 1, SBC is non-trivial for arbitrary job lengths, even when the jobs are independent (i.e., when the graph contains no edges). We give here a strongly polynomial time algorithm for a meaningful simple class of pebbly graphs: those consisting of disjoint cliques of bounded size. Disjoint cliques correspond naturally to the case of single-resource constraints, namely, each job needs to use a single resource to be processed; thus, all jobs that compete for the same resource form a clique.

**Proposition 15** *There is an algorithm for* BSC *on disjoint collection of cliques, that runs in time* $\min(h^{O(d)}, 2^{O(h)})n$, *where* $h$ *is the size of the largest clique and* $d$ *is the number of distinct lengths.*

*Proof sketch.* Given a subset $S$ of the lengths of the longest already colored batches, we compute the minimum cost of batch coloring the remaining graph. We do so for all $h$-bounded length sets, i.e. sets of size up to $h$. Namely, given lengthiest $S$, it is easy to determine for a particular clique $C$ which vertices have already been colored; the greedy choice of assigning to each batch the longest

vertex that fits in the batch is an optimal strategy. Denote by $v(S)$ the subset of vertices (in all cliques) contained in the set of batch lengths $S$. Also, let $a(C, S)$ denote the subset of vertices in the clique $C$ scheduled in $S$, and $r(C, S) = v(S) \setminus a(C, S)$ are the remaining vertices from $S$. Our dynamic programming table, indexed by $S$, stores the sum of the costs of scheduling $r(C, S)$, minimized over all batch colorings.

We can compute this by trying all possible values for extending $S$, namely values whose addition (and possibly dropping the smallest value) yields an $h$-bounded sequence that dominates $S$.

The number of $h$-bounded length sets is bounded by the number of subsets of the cliques in the graphs, or $\sum_{C \in G} 2^{|C|} \leq 2^h n / h$. When the number of lengths $d$ is fixed, we can also bound this number by $h^d$, by counting how many times each length $\ell$ occurs in the set. To compute each entry of the table we need to consult at most all dominated entries, for a total time complexity at most square of the table size. ∎

## 4   Max Coloring Paths

We present an efficient exact algorithm for the max coloring problem on paths. Our algorithm uses the observation that any non-trivial solution for MAX-COL on paths uses at most three batches (see, e.g., [8]). The proof of the next result is given in the appendix.

**Theorem 16** *There is an $O(n \log n)$ algorithm for* MAX-COL *on paths.*

## References

[1] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proc. 17th Conf. on Uncertainty in Artif. Intell.*, 7–15, 2001.

[2] B. S. Baker. Approximation algorithms for NP-hard problems on Planar Graphs. *J. of the ACM*, 41:153–180, 1994.

[3] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

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

[5] C. Bazgan. Approximation schemes and parameterized complexity. *PhD thesis*, INRIA, Orsay, France, 1995.

[6] H. L. Bodlaender. A partial $k$-arboretum of Graphs with Bounded Treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[7] H. L. Bodlaender and A. M. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 2007, `doi:10.1093/comjnl/bxm037`

[8] P. Brucker. *Scheduling Algorithms, 4th ed.* Springer, 2004.

[9] M. Cesati and L. Trevisan. On the Efficiency of Polynomial Time Approximation Schemes. *Information Processing Letters* 64:165–171, 1997.

[10] E. D. Demaine and M. Hajiaghayi. Equivalence of Local Treewidth and Linear Local Treewidth and its Algorithmic Applications. In *Proc. SODA*, 833–842, 2004.

[11] M. Demange, D. deWerra, J. Monnot, and V. Th. Paschos. Time slot scheduling of compatible jobs. *J. of Scheduling*, 10:111–127, 2007.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.

[13] L. Epstein, and A. Levin. On the max coloring problem. In *Proc. of WAOA'07*.

[14] L. Epstein, M. M. Halldórsson, A. Levin, and H. Shachnai. Weighted Sum Coloring in Batch Scheduling of Conflicting Jobs. To appear in *Algorithmica*.

[15] B. Escoffier, J. Monnot, V. Th. Paschos. Weighted Coloring: Further complexity and approximability results. *Inf. Process. Lett.* 97(3):98–103, 2006.

[16] U. Feige and J. Kilian. Zero Knowledge and the Chromatic number. *Journal of Computer and System Science*, 57:187–199, 1998.

[17] R. Gandhi, M. M. Halldórsson, G. Kortsarz and H. Shachnai. Improved Bounds for Sum Multicoloring and Scheduling Dependent Jobs with Minsum Criteria. To appear in *ACM Transactions on Algorithms*.

[18] D. J. Guan and X. Zhu. A Coloring Problem for Weighted Graphs. *Inf. Process. Lett.* 61(2):77–81, 1997.

[19] 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.

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

[21] C.-C. Han, C.-J. Hou and K.J. Shin. On slot reuse for isochronous services in DQDB networks. Proc. of *16th IEEE Real-Time Systems Symposium*, 222–231, 1995.

[22] H. Liu, M. Beck and J. Huang. Dynamic Co-Scheduling of Distributed Computation and Replication. In Proc. of *CCGRID* 2006, 592–600.

[23] M. Mihail, C. Kaklamanis and S. Rao. Efficient Access to Optical Bandwidth, In Proc. of *FOCS'95*, 548–557.

[24] B. R. Peek, High performance optical network architecture In *All-Optical Networking: Existing and Emerging Architecture and Applications*, 2002.

[25] S. V. Pemmaraju, R. Raman, and K. R. Varadarajan. Buffer minimization using max-coloring. In Proc. of *SODA* 2004, 562–571.

[26] S. V. Pemmaraju and R. Raman. Approximation Algorithms for the Max-coloring Problem. In Proc. of *ICALP* 2005, 1064–1075.

[27] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[28] A. S. Tanenbaum. *Distributed Operating Systems.* Prentice-Hall, 1995.

[29] D. B. West, *Graph Theory.* 2nd Ed., Prentice-Hall, 2001.

# A    Some Proofs

**Proof of Lemma 8:** We represent a batch length sequence as a word over an alphabet whose size is the number of distinct job lengths. Define a linear ordering on the batches by length (in non-decreasing values).

**Definition 17** *Let $\Sigma$ be a finite alphabet with a linear ordering $\preceq$ on the symbols. Given a word $w = (w_1, w_2, \ldots, w_t)$, the* rank $r_i(w)$ *of a symbol $w_i$ in $w$ is the number of symbols that are ordered equally or after $w_i$ under $\preceq$ and appear in $w$ in positions $1, \ldots, i$, namely,*

$$r_i(w) = |\{\nu : w_i \preceq w_\nu, \nu \le i\}|.$$

*We say that the word $w$ is $k$-restricted if $r_i(w) \le k$, for all $i = 1, 2, \ldots, t$.*

**Lemma 18** *The number of $k$-restricted words from an alphabet $\Sigma$ is at most $2^{k|\Sigma|}$.*

*Proof.* Consider more generally the problem where given a vector $\mathbf{a} = (a_1, a_2, \ldots, a_s)$ of non-negative integers, with $s \le |\Sigma|$, and $a_1 \le a_2 \le \cdots \le a_s$, we seek the number of words such that for each $1 \le j \le s$, the $j$-th symbol has rank at most $a_j$. We give a recurrence relation for this number, denoted by $t(a_1, a_2, \ldots, a_s)$.

If $a_1 = 0$ then $t(a_1, a_2, \ldots, a_s) = t(a_2, \ldots, a_s)$. Otherwise, there are $s$ possible symbols that can be the first symbol of the word. The number of valid words when the $j$-th symbol is first is $t(a_1 - 1, a_2 - 1, \ldots, a_j - 1, a_{j+1}, \ldots, a_s)$, the reason being that this reduces the possible number of occurrences of a given symbol in the remaining words by one, if that symbol is ranked $j$-th or earlier. Then, the total number of valid words is

$$t(a_1, a_2, \ldots, a_s) = \sum_{j=1}^{s} t(a_1 - 1, \ldots, a_j - 1, a_{j+1}, \ldots, a_s). \tag{1}$$

Let $t(z) = \max_{z = \sum_i a_i} t(a_1, a_2, \ldots, a_s)$ be the maximum number of valid words corresponding to vectors $\mathbf{a}$ whose entries sum up to $z$. Then, for $z = \sum_i a_i$, it follows from (1) that

$$t(z) \le \sum_{j=1}^{s} t(z - j), \tag{2}$$

where $t(0) = t(1) = 1$. We note that (2) is satisfied by $t(z) = 2^{z-1}$. (This solution for $t(z)$ is tight; indeed, in the special case where $s = z$ we get in (2) equality.) Observe that $k$-restricted words satisfy $z \le k|\Sigma|$, this gives the lemma. ∎

By Observation 3.1, in a graph of maximum degree $\Delta$, preceding a batch of a given length are at most *Delta* batches of longer or equal length. In other words, any word $w$ corresponding to a batch length sequence for such graphs is $\Delta$-restricted. Thus, we have a bound of $2^{(\Delta+1)d}$ on the number of batch sequences. ∎

**Proof of Theorem 16:** Let $x_s$ be the height of the $s$-th batch in some optimal solution, $1 \leq s \leq 3$. Note that $x_1 = p_{max}$, the length of the longest job. For each $s, t \in \{1, 2, 3\}$ and $1 \leq i \leq j \leq n$, we compute a vector $h(i, s, j, t)$ of length $(j - i + 1)$ which gives the minimum value of the third length $(x_3)$, for each possible value of the second length $(x_2)$, for vertices on the subpath between $i$ and $j$, when $i$ is scheduled in batch $s$ and $j$ in batch $t$. The possible values of $x_2$ are the lengths of the vertices on this subpath, that is, $x_2 \in \{p_i, p_{i+1}, \ldots, p_j\}$; the entries of the vector $h$ are sorted in non-increasing order by the value of $x_2$. The vector $h(i, s, j, t)$ can be calculated recursively, using the following steps.

1. Let $k = i + \lfloor \frac{j-i+1}{2} \rfloor$. For all $1 \leq \nu \leq 3$, calculate the vectors $h(i, s, k, \nu), h(k, \nu, j, t)$.

2. Initialize the sorted vector $h(i, s, j, t)$ with the lengths of all the vertices between $i$ and $j$, as the possible values of $x_2$.

3. The value of $x_3$ in each entry of $h(i, s, j, t)$ is calculated as follows. Denote by $(x_3 \in h(i, s, j, t) | x_2)$ the value of $x_3$ for a given value of $x_2$ in the vector $h(i, s, j, t)$. For each value of $x_2$ in $h(i, s, j, t)$, let $x_2^1$ $(x_2^2)$ be the maximal length of a vertex on the subpath between $i$ and $k$ ($k$ and $j$) satisfying $x_2^1 \leq x_2$ $(x_2^2 \leq x_2)$. Define

$$x_3^1 = \min_{1 \leq \nu \leq 3} (x_3 \in h(i, s, k, \nu) | x_2^1)$$

and

$$x_3^2 = \min_{1 \leq \nu \leq 3} (x_3 \in h(k, \nu, j, t) | x_2^2)$$

Now, we take

$$x_3 = \max\{x_3^1, x_3^2\}. \tag{3}$$

We find an optimal solution for max coloring by calculating $h(1, s, n, t)$ for all $1 \leq s, t \leq 3$, and selecting (within these 9 vectors) the entry which gives the minimum makespan.

The schedule is obtained by keeping throughout the recursion the values of $\nu$ that yield the minimum values of $x_3$ (in (3)).

We now analyze the running time of the algorithm. First, note that each pair of vectors is merged in linear time, since the vectors are sorted by the $x_2$ values. Now, we count the number of subproblems to be solved in the recursive calls. We need to solve 9 problems of length $n$. In the sequel, we solve $9 \cdot 2^i$ problems of length $n/2^i$, whose total size is $O(n)$. Since $1 \leq i \leq \log n$, we get that the overall running time of the algorithm is $O(n \log n)$. ∎