

Batch Coloring Flat Graphs and Thin

Magnús M. Halldórsson¹ and Hadas Shachnai²

¹ School of Computer Science, Reykjavik University, 103 Reykjavik, Iceland.
E-mail: mmh@ru.is.

² Department of Computer Science, The Technion, Haifa 32000, Israel.
E-mail: hadas@cs.technion.ac.il.

Abstract. A batch is a set of jobs that start execution at the same time; only when the last job is completed can the next batch be started. When there are constraints or conflicts between the jobs, we need to ensure that jobs in the same batch be non-conflicting. That is, we seek a coloring of the conflict graph. The two most common objectives of schedules and colorings are the makespan, or the maximum job completion time, and the sum of job completion times. This gives rise to two types of *batch coloring problems*: *max-coloring* and *batch sum coloring*, respectively. 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 (unbounded) p -batch scheduling problem [6], we are given a set of n jobs where job J_j has processing time, or *length*, p_j . We need to partition the jobs into *batches*. All jobs in a batch start at the same time, and the batch is completed when its last job finishes. The length of a batch is then the length of the longest job in the batch.

Real-life scenarios frequently 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 $G = (V, E)$, with $V = \{1, 2, \dots, n\}$ where the length of vertex j is the processing time of J_j , and 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. Each batch forms an independent set in G , hence a valid schedule is a proper coloring of the graph.

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 times problem* (SBC), we minimize the sum, over all the jobs J_j , of the completion time of the batch that contains J_j . Let BSC refer to either of the batch scheduling problems under the sum measure: SJC or SBC.

In this paper we give the first *polynomial time approximation schemes (PTAS)*, as well as exact algorithms, for BSC on several classes of graphs that arise in applications. We focus on “thin” graphs, namely, graphs with bounded treewidth, and “flat” graphs, i.e., graphs that can (almost) be embedded on flat surface.¹ In particular, we consider paths, trees and partial k -trees (which are “thin”), as well as planar and other “flat” graphs.

We say that an 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 $1 + \epsilon$, for any constant $\epsilon > 0$. 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)*.

Our problems naturally arise in the following applications.

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, the same slots can be assigned to messages with non-intersecting paths [15]. 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., [21, 16]) 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.

Other applications include, e.g., batch production [14], dynamic storage allocation and memory management in wireless protocol stack (see, e.g., [12, 19].) The graph classes that we study here arise in these applications from tree-like (or planar) network topologies [17] ([18]), conflict graphs of processes generated for computer programs [21], and scheduling jobs with bounded resource requirements (which yield *pebbly* graphs; see in Section 3.4).

1.1 Related Work

There is a wide literature on batch scheduling, namely, our problems with empty conflict graph; see a comprehensive survey in [6]. 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 [3]. Constant factor approximations for SJC were given in [2] for, e.g., various classes of perfect graphs, with

¹ Formal definitions can be found, e.g., in [22].

improved constant factor obtained in [9]. The SBC variant was introduced in [9] and approximated on various graph classes. The common special case when all jobs are of unit length is known as the *sum coloring problem*. This implies that BSC is hard to approximate within factor $n^{1-\varepsilon}$, for any $\varepsilon > 0$ [2].

Max-coloring: The max-coloring problem was first considered in [12], where an $O(n^4)$ -time algorithm was given for paths. A quadratic algorithm was presented more recently in [10]. A PTAS is known for trees [19, 10] and for partial k -trees [10]. Constant factor approximation and NP-hardness results are known for various classes of perfect graphs (see the recent work of [8] for details).

Multicoloring problems: Batch coloring problems relate to certain multicoloring problems where vertices must be assigned a given number of contiguous colors, corresponding to a non-preemptive schedule. The difference is that in these problems, jobs are not restricted to start in batches. We shall be using some of the ideas developed for the non-preemptive sum multicoloring problem, NPSMC, defined as follows: Given a graph G with a length $x(v)$ for each vertex v , find an assignment f of starting times for the vertices such that no neighbor of v starts in the interval $[f(v), \dots, f(v) + x(v) - 1]$. For a comprehensive survey of known results for multicoloring problems with minsum objectives see, e.g., [11].

1.2 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.

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. The complexity of these schemes are of the form $2^{\epsilon^{-O(1)}} n$, when guaranteeing a $1 + \epsilon$ -approximation. By a result of [4, 7], this implies that BSC is fixed parameter tractable on these classes of graphs. Our scheme for planar graphs improves the running time of a PTAS proposed in [13] for the special case of the sum coloring problem to a linear time EPTAS, more precisely $2^{\epsilon^{-1} \log \epsilon^{-1}} n$ time. 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 [10].

Contribution: Our main approximation technique combines technical tools developed in [13] 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 [12, 19]); 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 ℓ 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 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 Δ the maximum degree of the graph.

Due to space constraints, some of the proofs and implementation details are omitted. The detailed results appear in [14].

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. 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 depend 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 for the sum of completion times problems, even on paths, nor is there a known exact polynomial time 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 SBC coloring of paths.²

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 $i = 1, 2, \dots, n$.

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 $s = j + 1, \dots, n$, 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, it suffices to use a fixed geometric sequence to reduce the problem to the ordinary coloring problem, within a constant factor [19]. 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.

We observe that a result of [13], 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.

² The paper [19] gives a similar result for the max coloring problem of bipartite graphs.

Theorem 1. *Let n , $q = q(n)$ and σ 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 [13] 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). 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 χ the chromatic number of the graph.

3.1 Properties and tools

Observation 2 *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 Δ 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 [20, 6]). (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 regarding batch coloring of χ -colorable graphs.

Lemma 1. *If G is χ -colorable, then $\text{SBC}(G) \leq \text{SJC}(G) \leq 3\chi \cdot p(G)$.*

Proof. Recall that τ is the maximal ratio between the processing times of any two vertices. Use a batch sequence with χ batches of each length $2^i p_{\min}$, for $i = 1, 2, \dots, \lceil \log \tau \rceil$, 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 $p(v) + \chi \sum_{j=0}^{\lceil \log \tau \rceil} p(v)/2^j + (\chi - 1)/2 \cdot 2p(v) = 3\chi p(v)$. Thus, the total schedule cost is at most $3\chi p(G)$. ■

The following lemma, which extends a result of [13], helps rein in the total length of our approximate schedule.

Lemma 2. *Let G be a χ -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 $\epsilon \cdot (\text{OPT}(G) - p(G))$.

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 ℓ . Then, following batch $\chi \cdot \epsilon^{-1}$ of length ℓ , we introduce χ batches of length ℓ , shifting all later batches by $\chi \cdot \ell$. We color all jobs that occurred in later batches of length ℓ using the χ 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 ℓ , only if it was already preceded by ϵ^{-1} times that many batches. Thus, each batch in the resulting coloring is delay by at most an ϵ -fraction of what previously came before it, or at most $\epsilon \cdot (\text{OPT}_J(G) - p(G))$.

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 1, 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 $2\chi \cdot p_{\max}$. ■

3.2 Thin graphs

We give here algorithms for the class of partial k -trees. For formal definition of this class, and that of the related tree decompositions, please refer to, e.g., [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 3. *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.*

Proof sketch. For each bag in the tree decomposition 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 2. *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 3 to search for restricted $(1 + \epsilon)$ -colorings guaranteed by Lemma 2. The time complexity of the algorithm of Lemma 3 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 2, using that $\mu(G) \leq \chi \cdot p_{max}$. ■

We now combine Proposition 2 with Theorem 1.

Theorem 3. *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 3 to search for restricted $(1 + \epsilon)$ -colorings guaranteed by Lemma 2 with these parameters. Theorem 1 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 2 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 polynomial time algorithm for BSC on trees remains open, it appears unlikely. We consider instead parametrizations that lead to efficient exact solutions. We treat the parameters d , the number of distinct job lengths, and Δ , the maximum degree.

Consider graphs of maximum degree Δ . Since there can be at most Δ 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 2) 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 [14].

Lemma 4. *The number of possible batch length sequences with d distinct lengths is at most $2^{(\Delta+1)d}$.*

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 4. *There is a $2^{O(\Delta \cdot k \cdot d)}$ -time algorithm for BSC in partial k -trees of maximum degree Δ 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 [1]. See [13] for details on the following specific version. A t -outerplanar graph has treewidth at most $3t - 1$ [5].

Lemma 5. *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 1, 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 [13], 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 [13] for NPSMC, whose running time is of the form $(\log n)^{\epsilon^{O(1)}} n$.

Lemma 6. *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 5 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 2, and find a schedule

S_2 of G_2 using Lemma 1. 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 ℓ 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 ℓ , for $i = 1, 2, 3, 4$. Then, each job in S is delayed at most $i\ell$ by batches of length ℓ 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 ϵ 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 1, for a total of $O(\epsilon^{-1} p_{max})$.

The time complexity is dominated by the time used by the algorithm of Proposition 2 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 6 with Theorem 1. The time complexity is $2^{\Theta(\epsilon^{-4} \cdot \log^3 \epsilon^{-1})} n$.

Theorem 5. *There is an EPTAS for BSC on planar graphs.*

Recall that sum coloring (SC) is the common special case of SBC and SJC when all vertices have the same length. It was shown in [13] 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 [13] that has time complexity $\exp(O(\ln \ln n \cdot \epsilon^{-1} \log \epsilon^{-1})) \cdot n$.

Theorem 6. *There is an EPTAS for SC on planar graphs with running time $(\log \epsilon^{-1})^{O(\epsilon^{-1})} n$.*

In the full version [14], we indicate how the result of Theorem 5 can be extended, e.g., to the larger class of 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 graphs: those consisting of disjoint cliques of bounded size. This class is neither thin nor flat; instead, we can say it is *pebbly*. 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 3. *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. We give a dynamic programming formulation. For each (multi)set S of possible job lengths, we compute the minimum cost of coloring a certain subgraph. Namely, we assume that a part of the graph has already been colored, and that S contains the lengths of the longest batches already colored. We therefore compute, for each such S , the minimum cost of batch coloring the best possible remaining graph. Given such a set S , it is easy to tell which nodes must have been already colored to leave the best possible remainder; the greedy choice of assigning to each batch the longest vertex that fits in the batch is an optimal strategy.

Hence, we have a table indexed by h -bounded multisets (sets of size at most h). We can fill into this table, bottom up, by trying all possible values for extending the given set S , namely values whose addition (with the possible deletion of the smallest value) yields an h -bounded sequence dominating S .

The number of h -bounded length sets is bounded by the number of subsets of the cliques in the graphs, or $\sum_{C \in \mathcal{G}} 2^{|C|} \leq 2^h n/h$. When the number of lengths d is fixed, we can also bound this number by h^d , by counting how many times each length ℓ 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 sketch below an improved exact algorithm for the max-coloring problem on paths. It uses the observation that any non-trivial solution for MAX-COL on paths uses at most three batches (see, e.g., [6]).

Theorem 7. *There is an $O(n \log n)$ algorithm for MAX-COL on paths.*

Let x_s be the height of the s -th batch in some optimal solution, $1 \leq s \leq 3$ (note that $x_1 = p_{max}$). For each $s, t \in \{1, 2, 3\}$ and $1 \leq i \leq j \leq n$, where $j - i$ is a power of 2 dividing both i and j , 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, $x_2 \in \{p_i, p_{i+1}, \dots, p_j\}$; 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.

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.

References

1. B. S. Baker. Approximation algorithms for NP-hard problems on Planar Graphs. *J. of the ACM*, 41:153–180, 1994.
2. 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.
3. 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.
4. C. Bazgan. Approximation schemes and parameterized complexity. *PhD thesis*, INRIA, Orsay, France, 1995.
5. H. L. Bodlaender and A. M. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 2007, doi:10.1093/comjnl/bxm037
6. P. Brucker. *Scheduling Algorithms*, 4th ed. Springer, 2004.
7. M. Cesati and L. Trevisan. On the Efficiency of Polynomial Time Approximation Schemes. *Information Processing Letters* 64:165–171, 1997.
8. L. Epstein, and A. Levin. On the max coloring problem. In *Proc. of WAOA'07*.
9. L. Epstein, M. M. Halldórsson, A. Levin, and H. Shachnai. Weighted Sum Coloring in Batch Scheduling of Conflicting Jobs. To appear in *Algorithmica*.
10. B. Escoffier, J. Monnot, V. Th. Paschos. Weighted Coloring: Further complexity and approximability results. *Inf. Process. Lett.* 97(3):98–103, 2006.
11. 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*.
12. D. J. Guan and X. Zhu. A Coloring Problem for Weighted Graphs. *Inf. Process. Lett.* 61(2):77–81, 1997.
13. 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.
14. M. M. Halldórsson and H. Shachnai. Batch Coloring Flat Graphs and Thin. full version. http://www.cs.technion.ac.il/~hadas/PUB/batch_col.pdf.
15. 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.
16. H. Liu, M. Beck and J. Huang. Dynamic Co-Scheduling of Distributed Computation and Replication. In *Proc. of CCGRID 2006*, 592–600.
17. M. Mihail, C. Kaklamanis and S. Rao. Efficient Access to Optical Bandwidth, In *Proc. of FOCS'95*, 548–557.
18. B. R. Peek, High performance optical network architecture In *All-Optical Networking: Existing and Emerging Architecture and Applications*, 2002.
19. S. V. Pemmaraju and R. Raman. Approximation Algorithms for the Max-coloring Problem. In *Proc. of ICALP 2005*, 1064–1075.
20. W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
21. A. S. Tanenbaum. *Distributed Operating Systems*. Prentice-Hall, 1995.
22. D. B. West, *Graph Theory*. 2nd Ed., Prentice-Hall, 2001.