Weighted Sum Coloring in Batch Scheduling of Conflicting Jobs[*]

Leah Epstein[†]   Magnús M. Halldórsson[‡]   Asaf Levin[§]   Hadas Shachnai[¶]

Abstract

Motivated by applications in batch scheduling of jobs in manufacturing systems and distributed computing, we study two related problems. Given is a set of jobs $\{J_1, \ldots, J_n\}$, where J_j has the processing time p_j, and an undirected intersection graph $G = (\{1, 2, \ldots, n\}, E)$, with an edge (i, j) whenever the pair of jobs J_i and J_j cannot be processed in the same batch. We are to schedule the jobs in batches, where each batch completes its processing when the last job in the batch completes execution. The goal is to minimize the sum of job completion times. Our two problems differ in the definition of *completion time* of a job within a given batch. In the first variant, a job completes its execution when its batch is completed, whereas in the second variant, a job completes execution when its own processing is completed.

For the first variant, we show that an adaptation of the greedy set cover algorithm gives a 4-approximation for perfect graphs. For the second variant, we give new or improved approximations for a number of different classes of graphs. The algorithms are of widely different genres (LP, greedy, subgraph covering), yet they curiously share a common feature in their use of *randomized geometric partitioning*.

1 Introduction

Batching is defined as follows (see e.g. Chapter 8 in [3]). A batch is a set of jobs that can be processed jointly. The completion time of a batch is the last finishing time of a job in the batch. Usually, one defines the completion time of a job in a batch as the completion time of the batch that contains it. In the p-batch set of problems, the length of a batch is defined as the maximum processing time of any job in the batch. The s-batch set of problems has a different definition for the length of a batch, namely, it is partitioned into a setup time and the sum of the processing times of the jobs in the batch. In this paper we study p-batch problems.

Consider a communication network (e.g., an optical network), which consists of a set of vertices $V = \{1, \ldots, n\}$ interconnected by a given topology; any pair of vertices communicates through a prespecified path. Given is a set of requests $R_i = [a_i, b_i]$, where $a_i, b_i \in V$; each request R_i has a length p_i, meaning that during a period of p_i time units the communication links along the path connecting a_i and b_i must be dedicated to the processing of R_i. When communication is synchronous, time is divided into disjoint periods (or phases); during such a period, each link is dedicated to at most a single request, so the processing of this request

starts at the beginning of the period and ends at some point within the same period. When the processing of all requests assigned to the same time period is completed, this time period ends, and the system starts serving the requests of the next time period. A set of requests can be modeled as an undirected graph, in which the vertices represent the requests, and two vertices are adjacent if the paths of the corresponding requests contain common edges. In each time period, an independent set in this graph can be processed. In the special case where the network topology is a line, the intersection graph of the requests is an interval graph. Motivated by the above scheduling problem and other batch scheduling problems arising in manufacturing systems and in distributed computing (see below) we study two related problems.

Definitions. [1] In the *minimum sum of batch completion times problem* (**MSBCT**), we are given a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, where J_j has processing time (or *length*) p_j, [2] and an undirected intersection graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$; 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. A batch is completed when the last job in the batch finishes its processing, and all the jobs in a batch terminate once the batch is completed. The goal is to minimize the sum of job completion times, where the completion time of a job is the completion time of its batch. In other words, the goal is to partition V into independent sets, and to sort these independent sets, so that the weighted sum of batch completion times is minimized, where the weight of each batch (or, an independent set) $\mathcal{S}$ is the number of vertices in $\mathcal{S}$, and its processing time is equal to the maximum processing time of any job in $\mathcal{S}$.

The *minimum sum of job completion times problem* (**MSJCT**) is similar to **MSBCT**, except that each job can terminate as soon as its processing is completed, i.e., a job need not wait until the end of processing of the entire batch. However, jobs can still only be started when a batch starts.

Related problems. Our two batch scheduling problems generalize the *sum coloring problem* defined as follows. Given an input graph $G = (V, E)$, find a proper coloring f of V so that $\sum_{v \in V} f(v)$ is minimized, where a proper coloring of V is a function $f : V \to \mathbb{N}$, such that for all $(u, v) \in E$, $f(u) \neq f(v)$. Halldórsson et al. [14] considered a family of graph classes, denoted below by $\mathcal{F}$, defined as those for which the maximum ℓ-colorable induced subgraph problem is polynomial time solvable, for any value of ℓ. This family includes interval graphs, comparability graphs and co-comparability graphs (see [9, 27]). They presented a randomized 1.796-approximation algorithm and a deterministic $(1.796 + \varepsilon)$-approximation algorithm for the sum coloring problem on graphs belonging to classes in $\mathcal{F}$. This last result improves an earlier 2-approximation algorithm of Nicoloso, Sarrafzadeh and Song [21] for sum coloring interval graphs. Bar-Noy et al. [1] gave a 4-approximation for sum coloring perfect graphs, a 2-approximation for line graphs and a k-approximation for $k + 1$-claw free graphs (which are graphs that contain no induced star with $k + 1$ leaves). They also showed that sum coloring on general graphs is hard to approximate within factor $n^{1-\varepsilon}$, unless P = NP. We note that sum coloring is the special case of both **MSBCT** and **MSJCT** when all processing times are identical. Due to the hardness result for sum coloring general graphs, which applies also to our problems, we consider **MSBCT** and **MSJCT** only on special families of graphs.

---

[1] MMH3: How about shortening the names of the problems to **SBC** and **SJC**?

[2] MMH3: Should we specify the domain of the p_j?

Feige, Lovász and Tetali [8] extended the definition of sum coloring to the following variant of set cover, called *minimum sum set cover problem* (MSSC). We are given a collection of subsets $S_1, S_2, \ldots, S_m$ of a base set $S = \{1, 2, \ldots, n\}$. A feasible solution is an ordering π of a subset $\mathcal{S}'$ of $1, 2, \ldots, m$, such that $\bigcup_{\mathcal{X} \in \mathcal{S}'} \mathcal{X} = S$, and for each element j of the base set we incur a cost i, such that $j \in S_{\pi(i)}$ and $j \notin S_{\pi(k)}$ for all $k < i$. The goal is to find an ordering that minimizes the total cost. They obtained a 4-approximation algorithm for MSSC by showing that a greedy algorithm for sum coloring perfect graphs attains the same ratio for MSSC. In the context of a set system, this algorithm is equivalent to the well-known greedy algorithm for the classical set cover problem that selects in each iteration the set that covers the most number of uncovered elements [18, 19]. They further showed that this approximation ratio is best possible unless $P = NP$. A weighted generalization of this problem, motivated by database applications, was considered recently by Munagala et al. [20]. In this problem, each subset S_ℓ has a weight c_ℓ. For an element j of the base set, let i be an index such that $j \in S_{\pi(i)}$ and $j \notin S_{\pi(k)}$ for all $k < i$. The cost incurred by j is $\sum_{\ell=1}^{i} c_\ell$. They showed using linear programming that an application of the weighted greedy set cover algorithm (see [5]) gives a 4-approximation for this problem as well. Note that the above results can be applied to **MSBCT** on perfect graphs. In a graph, the sets S_i are given implicitly, however, in perfect graphs we can compute a maximum independent set in polynomial time. Recall also the following properties of perfect graphs (see e.g. [23]): a subgraph of a perfect graph is a perfect graph, and graph coloring is polynomial-time solvable in perfect graphs.

In the *maximum independent set (MIS)* problem, we are given an undirected graph $G = (V, E)$, and the goal is to find a subset U of V where E does not contain an edge between a pair of vertices in U, such that the size of U is maximized. This problem is well-known to be NP-hard on general graphs (see problem [GT20] in [11]). However, there are graph classes for which it is solvable in polynomial time (e.g., perfect graphs [13] and line graphs [7]), and there are graph classes for which there are efficient approximation algorithms. In this paper, we denote by $\rho \leq 1$ the best known approximation ratio of MIS on the graph class containing the graph in question.

MSJCT was introduced by Bar-Noy et al. [2], who presented a 16-approximation for perfect graphs, and, more generally, a $16/\rho$-approximation. No other results have been previously reported for this problem.

1.1 Applications of MSBCT and MSJCT

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 the goal is to find a production schedule which minimizes the average completion time of an order. Thus, 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. This yields an instance of **MSJCT**.

Scheduling in Distributed Systems Batch scheduling of conflicting jobs is used in distributed operating systems (see, e.g., [25]). In such systems, the scheduler identifies subsets of non-conflicting or cooperating processes that can run concurrently (e.g., because these pro-

cesses do not use the same resources or communicate frequently with each other); each subset is then executed simultaneously on several processors, until *all* the processes in the subset have completed. Thus, the problem of finding a schedule that minimizes the sum of completion times of all processes can be modeled as an instance of **MSBCT** with a general conflict graph.

1.2   Our Results

We describe below our main results for **MSBCT** and **MSJCT**. In Section 2 we define the extension of the greedy algorithm and its analysis (due to [8]) to obtain a $\frac{4}{\rho}$-approximation algorithm for **MSBCT**, for any graph class with a a ρ-approximation for MIS. Note that the case $\rho = 1$ also follows from the results of [20], but with a different proof technique.

In Section 3 we consider **MSJCT**. Let $e \approx 2.71828$ denote the base of the natural logarithm. We first present a $2e \approx 5.437$-approximation algorithm for the problem on interval graphs, and later obtain a better bound of $1.296e + 3/2 + \varepsilon \approx 5.022$ for any graph class that belongs to $\mathcal{F}$. The first two algorithms can be combined to obtain an improved bound of 4.912 for interval graphs. Then, we show a $\frac{4e}{\rho}$-approximation algorithm. We also show that the classical greedy algorithm (given in Section 2) provides an alternative $\frac{4e}{\rho}$-approximation for **MSJCT**. Thus, both of these algorithms yield combinatorial $4e \approx 10.873$-approximation for perfect graphs and line graphs, and a $(2ek + \epsilon)$-approximation for $(k + 1)$-claw free graphs (based on a $2/(k + \epsilon)$-approximation for MIS by Hurkens and Schrijver [15]). Finally, in Section 4 we present a general LP-based scheme for batch scheduling that yields a bound of 9.9 for perfect graphs, and $9.9 + o(1)$ for line graphs.

Our results for **MSJCT** yield significant improvements over previous bounds; in particular, the bounds that we derive for interval graphs, line graphs and perfect graphs improve upon the uniform bound of 16 obtained for these graph classes in [2]. We summarize the known results for **MSJCT** in Table 1. New bounds given in this paper are shown in boldface, with the previous best known bound given in parenthesis. The reference to previous results is [2].

Graph class	MSJCT
General graphs	$n/\log^2 n$
Perfect graphs	**9.9** (16)
Family $\mathcal{F}$	**5.022**
Interval graphs	**4.912** (16)
k-colorable graphs	$1.544k + 1$
Bipartite graphs	2.796
$k + 1$-claw free graphs	**2ek + ϵ**
Line graphs	**9.9 + o(1)** (16)

Table 1: Known results for batch scheduling of conflicting jobs under minsum criteria

Techniques: While the algorithms that we develop for various graph classes are inherently different and tailored to exploit the intrinsic properties of each graph class, the approaches at the core of these algorithms share a common thread in the crucial use of *randomized geometric partitioning.* This technique has an extensive history for non-graph problems; see [4] for a recent survey on its applications to online problems. Our main partitioning lemma (Lemma 3) uses randomized rounding to partition the jobs into length classes, which enables to improve

the bound of 16 obtained in [2] for **MSJCT**, to $4e$. Our algorithm for the family $\mathcal{F}$ (see in Section 3.4) randomizes on ℓ, and then finds an ℓ-colorable induced subgraph, from which to construct batches. Finally, our LP-based scheme (in Section 4) combines randomized partitioning in two different ways after solving a linear programming relaxation of the problem. We show that the resulting algorithms can be derandomized while preserving their approximation ratio within an additive ε, for any $\varepsilon > 0$. We believe that this powerful technique will find more applications in solving other scheduling and partitioning problems.

2 Approximating MSBCT

In this section we present the greedy algorithm for **MSBCT**, and show that it yields a $\frac{4}{\rho}$-approximation for the problem. Our proof follows a similar proof of [8] for the unweighted case, *MSSC*.

Greedy(G)
While G is non-empty do:
    For each $j = 1, 2, \ldots, n$ do:
      Let G_j be the induced subgraph of G over the vertex set $\{j' : p_{j'} \leq p_j\}$.
      Find a ρ-approximate independent set S_j in G_j.
    Let $j = \operatorname{argmin}_{j'=1,2,\ldots,n} \frac{p_{j'}}{|S_{j'}|}$.
    Schedule the independent set S_j in the next p_j time units.
    Remove S_j from G.

Theorem 1 GREEDY *yields a $\frac{4}{\rho}$-approximation for* **MSBCT**.

Proof. Let X_i be the independent set (S_j) of jobs found by GREEDY in iteration i, for $i = 1, 2, \ldots$. Let R_i be the set of jobs that are still not processed prior to the i-th iteration and note that $X_i \subseteq R_i$. Denote by $P_i = \max\{p_j : j \in X_i\}$ the processing time of the batch X_i. For each $j \in X_i$ we define the *price of job j* to be $\mathbf{price}(j) = \frac{|R_i| \cdot P_i}{|X_i|}$. Then, the cost of GREEDY is given both by $\sum_i P_i \cdot |R_i|$ and by $\sum_{j=1}^{n} \mathbf{price}(j)$.

Consider the following histogram corresponding to OPT. There are n columns, one for every job, where the jobs are ordered from left to right by the order in which they were processed by the optimal solution (breaking ties arbitrarily). The height of a column is the time step at which the job was completed by the optimal solution (i.e., the time in which the batch that contains the job is completed). Hence, we get a histogram with nondecreasing heights. The total area beneath this histogram is exactly OPT.

Consider now a different diagram corresponding to the greedy solution. Again there are n columns, one for every job, and in analogy to the previous diagram, the jobs are given in the order in which they were processed by GREEDY. But unlike the previous diagram, the height of a column is not the time step by which the job was completed, but rather its price. Hence, the heights are not necessarily monotone. The total area of the histogram is exactly the total price, i.e., the cost of the greedy solution. We would like to show that the area of the second histogram is at most $\frac{4}{\rho}$ times that of the first. To do this we shrink the second histogram by a factor of $\frac{4}{\rho}$ as follows. We shrink the height of each column by a factor of $\frac{2}{\rho}$. Hence, the column heights are $\frac{\mathbf{price}(j)\rho}{2}$. We also shrink the width of each column by a factor of two. Hence, the

5

total width of the second histogram is now $\frac{n}{2}$. We align the second histogram to the right. Namely, it now occupies the space that was previously allocated to columns $\frac{n}{2}+1$ up to n (assume for simplicity of notation and without loss of generality that n is even). Now we claim that this shrunk version of the second histogram fits completely within the first histogram, implying that its total area is no more than that of the first histogram. This suffices in order to prove the approximation ratio of GREEDY.

Consider an arbitrary point q in the original second histogram, let j be the job to which it corresponds, and let i denote the iteration of GREEDY in which we chose to process job j (i.e., $j \in X_i$). Then the height of q is at most $\mathbf{price}(j)$, and the distance of q from the right hand side boundary is at most $|R_i|$. The shrinking of the second histogram maps q to a new point q'. We now show that q' must lie within the first histogram. The height of q', which we denote by h, satisfies $h \leq \frac{|R_i| \cdot P_i \cdot \rho}{2|X_i|}$, and the distance of q' from the right hand side boundary (which we denote by r) satisfies $r \leq \frac{|R_i|}{2}$. For this point q' to lie within the first histogram, it suffices to show that by time step h, at least r jobs are still not completed by the optimal solution.

Consider now only the jobs in the set R_i. No independent set whatsoever can complete more than $\frac{|X_i|}{P_i \cdot \rho}$ jobs from R_i per time unit. Hence, in h time units the optimal solution could complete processing at most $\frac{h|X_i|}{P_i \rho} \leq \frac{|R_i| \cdot P_i \cdot \rho}{2|X_i|} \cdot \frac{|X_i|}{P_i \cdot \rho} = \frac{|R_i|}{2}$ jobs from R_i, leaving at least $\frac{|R_i|}{2}$ jobs of R_i that the optimal solution still has not completed. Hence, the point q' indeed lies within the first histogram. ∎

Corollary 2 GREEDY *yields a 4-approximation for* **MSBCT** *on perfect and claw-free graphs.*

3   Approximating MSJCT via Length Rounding

The outline of the algorithms is as follows: In the preprocessing phase, all of our algorithms partition the jobs into classes according to their processing times and round up the processing time of each job to the maximum processing time of a job in its class. Then, each batch that we generate is devoted to a single class of jobs. For each class of graphs, we design a specialized algorithm to find an approximate partition of each class into batches. Then, all of our algorithms find an optimal sequence of the resulting batches using Smith's rule.

3.1   Preprocessing

In this subsection we present our preprocessing step and analyze the degradation of the performances caused by the preprocessing. We first introduce a randomized version of the preprocessing, and derandomize it afterwards. W.l.o.g., we may assume that the minimum processing time is at least e.

Length Rounding
Pick uniformly a random number α in the range $[0,1)$, i.e., $\alpha \sim \mathbf{U}[0,1)$.
Partition the jobs into (length) classes according to their processing times,
i.e., let $\mathcal{J}_0 = \{j : p_j \leq e^\alpha\}$, and $\mathcal{J}_i = \{j : e^{i-1+\alpha} < p_j \leq e^{i+\alpha}\}$.
Denote by k the largest index of a non-empty length class.
For each $i = 0, 1, \ldots, k$ and each job j in $\mathcal{J}_i$,
round up the processing time of job j to $p'_j = e^{i+\alpha}$.

We now show that by partitioning the input into the length classes $\mathcal{J}_i$, rounding up the processing times (as done in LENGTH ROUNDING) and requiring that each batch contain only jobs from a single class, we lose a factor of at most e.

One might first wonder why it is important to use randomization. It may be helpful to consider the following related simplified problem. We have to come up with a number that is at least as large as a secret value. We will then guess some sequence until the last value is as large as the secret. Our cost will be the sum of all the numbers guessed. The best deterministic strategy is to keep doubling the guesses, starting with a sufficiently smallest initial guess. This is susceptible to the worst case when the secret is just incrementally smaller than the second-to-last guess, yielding a factor of 4. By randomizing the starting guess, we can avoid this worst case, reducing the expected cost. Increasing the doubling factor to e now gives the best possible tradeoff between the expected cost incurred by the previous guesses and the expected overshoot of the last guess. In the problem versions of this paper, the "secret" corresponds to the contribution of each node, on average, to the cost of the optimal solution. We now proceed with the analysis.

For a fixed value of α, denote by OPT_α an optimal solution to the instance in which we use the rounded up processing times p'_j and the solution must satisfy the additional requirement that for each batch the jobs scheduled in the batch have a common length class. Denote also by OPT_α the cost of OPT_α. Recall that we similarly let OPT refer to both a fixed optimal solution and its cost. Let $p(V) = \sum_{v \in V} p_v$ denote the sum of job processing times, and $p'(V) = \sum_{v \in V} p'_v$ be the sum of rounded processing times.

Lemma 3 $E[\mathrm{OPT}_\alpha] \leq e \cdot \mathrm{OPT}$, where the expectation is over the random choice of α.

Proof. We shall show how to replace each batch of OPT with a sequence of batches, each containing identical length class jobs, while satisfying the stated bound on the total cost.

Consider a fixed batch $\mathcal{B}$ of OPT with job set J. Let x be such that the maximum processing time of a job in J is e^x (i.e., $x = \ln \max_{j \in J} p_j$), and let $i_{max} = \lceil x - \alpha \rceil$. Note that $i_{max} + \alpha \geq x$. We replace $\mathcal{B}$ with a set of batches $B = \{\mathcal{J}_i \cap J : i = 0, 1, \ldots, i_{max}\}$. The total processing time of B is at most

$$\sum_{i=0}^{i_{max}} e^{i+\alpha} \leq e^{i_{max}+\alpha} \cdot \frac{e}{e-1}. \tag{1}$$

To bound the expected processing time of B, it suffices to show that $E[e^{i_{max}+\alpha-x}] = e - 1$, since the processing time of $\mathcal{B}$ is e^x. Since α is uniformly distributed in $[0, 1)$, so is the random variable $\lceil x - \alpha \rceil - (x - \alpha) = i_{max} + \alpha - x$. The claim regarding the length of the batch then follows by noting that for $u \sim \mathbf{U}[0, 1)$, $E[e^u] = \int_{u=0}^{1} e^u du = e^1 - e^0 = e - 1$.

We also need to show that the completion time of a job j within the set of batches replacing its batch also grows by an expected multiplicative factor of e. The proof is largely identical to the above. Indeed, suppose that a job j has original length e^y, so its processing time within the batch $\mathcal{B}$ was e^y. Letting $i' = \lceil y - \alpha \rceil$ and rounding the job length to $e^{i'+\alpha}$, the time until the new batch of j is completed is now at most $\sum_{i=0}^{i'} e^{i+\alpha} \leq e^{i'+\alpha} e/(e-1)$. Similar to the above, to get the ratio between the expected completion time of j in its new batch and its original completion time, it suffices to show that $E[e^{i'+\alpha-y}] = e - 1$, which gives the ratio of e. ∎

We next fix a value of α and show how to design algorithms that approximate OPT_α within constant factors. Then, using Lemma 3, the approximation ratio of the combined algorithms will be e times these constants. In fact, we can obtain a slight improvement.

Lemma 4 *Let G' be the graph with rounded processing times. Suppose an algorithm finds a solution of expected cost X on G' (with expectation taken over values of α). Then, the expected cost of the algorithm on G is $X - (p'(V) - p(V)) = X - (2 - e)p(V)$.*

Proof. We note that a job does not need to wait until the end of its rounded-up processing time, but ends as soon as the original processing time (from the start time of the job) elapses. Therefore, for each job v we gain $p'_v - p_v$ time units when we consider the output as a solution to the original instance of **MSJCT**, instead of a solution to the rounded-up instance. The reduction in the cost is then $p'(V) - p(V)$.

We claim that $E[p'(V)] = (e - 1)p(V)$. To see that, observe that for a given job j, the random variable $\frac{E[p'_j]}{p_j} = E\left[\frac{p'_j}{p_j}\right]$ is equivalent to e^u, where $u \sim \mathbf{U}(0, 1]$. We have already found in the proof of Lemma 3 that $E[e^u] = e - 1$, establishing the claim. ∎

3.1.1 Derandomization of the preprocessing step

A result of Lemma 3 is that there exists a value of α whose effect on the rounding and classification is an increase of the total cost by a factor of at most e.

In order to derandomize the algorithm we can use a finite number of values for α that are $\delta > 0$ apart, and obtain an increase of the approximation ratio of the preprocessing step within $\varepsilon > 0$ where ε is a linear function of δ. We refer to [14], for such approach.

In the following, we argue that it is enough to test $O(n)$ values of α so that the performance guarantee of Lemma 3 will not be hurt. To see this, note to begin with that each job length can fall into at most two length classes, since α is from $[0, 1)$. Different values of α result in different class partitions if and only if at least one job moves from one class to another class. This means that there are at most n threshold values of α in which a job moves from one class to another class. Each value is related to at least one job. Moreover, the best value of α is a result of picking one of these n values, and choosing α, so that the job which relates to this threshold value gets a rounded value which is its exact original value. This holds since if no such job (that is not actually rounded) exists, then decreasing α to the closest such threshold will result in a solution which is not worse.

Picking the best solution among the at most n solutions (obtained for the different values of α that we test) is clearly at least as good as picking α randomly. Thus we obtain a derandomization technique without an increase of the performance guarantee of Lemma 3.

Remark 3.1 *A similar derandomization method can be applied to the randomized algorithm of [14] for the sum coloring problem on a graph that belongs to $\mathcal{F}$. This gives a deterministic 1.796-approximation algorithm, improving the deterministic $1.796 + \varepsilon$ approximation algorithm of [14].*

3.2 The final step of the algorithms

To approximate OPT_α, we use a different algorithm for each class of graphs, so the choice of algorithm depends on the smallest class to which the graph G belongs. In the next sections we present approximation algorithms for interval graphs, and for graphs that belong to $\mathcal{F}$, and finally we show that a greedy algorithm is a $\frac{4}{\rho}$-approximation algorithm. Recall that ρ is the best known approximation ratio for the maximum independent set problem on the smallest graph class that contains the input graph. However, the final step of the algorithms is identical and is therefore presented now.

We assume that the algorithm has determined a partition of the jobs of each length class into batches. Therefore, we have a set of batches, each is a set of jobs with a common (rounded-up) processing time, and we need to schedule the batches so as to minimize the sum of job completion times.

We note that such an ordering can be found optimally using Smith's rule [24, 3]. Sort the batches according to a non-decreasing order of the ratio of the weight of the batch divided by the (common) processing time of the jobs in the batch, where a weight of a batch is the number of jobs assigned to this batch.

Since we find an optimal ordering of the batches, in our analysis we can consider some fixed ordering (that may depend on the structure of the optimal solution), and note that our algorithm will return a better solution than the one indicated by some other (sub-optimal) order of the same batches. It remains to show how to partition the jobs of each length class into a set of batches.

3.3   Interval graphs

Nicoloso, Sarrafzadeh, and Song [21] designed a 2-approximation algorithm for sum coloring interval graphs. It computes, for all $\ell = 1, 2, \ldots, \chi(G)$, a maximum ℓ-colorable induced subgraph G_ℓ of G in a greedy fashion from left to right according to an interval representation of G (that can be found in polynomial time [12]). Given the output of such a process, they showed that for all $\ell > 1$, G_ℓ contains $G_{\ell-1}$ and the difference graph $G_\ell - G_{\ell-1}$ is 2-colorable. Thus, their algorithm simply colors G_1 using color 1, and colors the difference graph $G_\ell - G_{\ell-1}$ using colors $2\ell - 2$ and $2\ell - 1$.

For the rounded-up instance resulting from the preprocessing step we apply the algorithm of [21] on each length class $\mathcal{J}_i$ separately to partition it into batches (and then apply the final step of the algorithm described in Section 3.2).

Theorem 5 *The resulting algorithm for interval graphs is a $2e$-approximation algorithm. Moreover, the algorithm constructs a solution with cost at most $2e\mathrm{OPT} - 2p'(V) + p(V)$.*

Proof. The algorithm runs in polynomial time since a maximum size ℓ-colorable subgraph of an interval graph can be found in polynomial time, independent of ℓ [27]. The algorithm returns a partition of the job set, and therefore yields a feasible solution. It remains to prove its approximation ratio. We will first show that after fixing α to a constant value in $[0, 1)$, our algorithm approximates OPT_α within a ratio of two.

Given OPT_α we create another solution denoted as SOL whose odd numbered batches are the batches of OPT_α and even numbered batches are empty. The cost of SOL is then exactly twice the cost of OPT_α minus $p'(V)$. Using Lemmas 3 and 4, it suffices to show that our algorithm returns a solution that costs at most the cost of SOL.

Denote by G^i the subgraph of G induced by the jobs of length class $\mathcal{J}_i$. For any ℓ, consider the set L_i^ℓ of the first $2\ell - 1$ batches in SOL that were allocated to a particular length class $\mathcal{J}_i$. Of these, at most ℓ (the odd-numbered ones) are non-empty. Each one is an independent set and therefore the number of jobs n_ℓ that are executed during these batches is at most the size of a maximum ℓ-colorable subgraph of G^i. Therefore, the solution returned by the algorithm places in the first $2\ell - 1$ batches allocated to $\mathcal{J}_i$ at least n_ℓ jobs. This holds for any value of ℓ.

We define a sub-optimal order of the batches constructed by the algorithm (from all length classes). Batch k of length class i is placed in the exact spot that SOL places the kth batch of this class. At each point in time, the resulting solution completes at least as many jobs as

9

SOL does from every class. Therefore, the cost of the solution returned by the algorithm in this sub-optimal order of the batches is at most the cost of *SOL*. Since the algorithm returns the optimal order of the batches, we conclude that the algorithm returns a solution whose cost is at most the cost of *SOL*. ∎

3.4 Graphs that belong to $\mathcal{F}$

We recall that given a graph G that belongs to $\mathcal{F}$, for any $\ell \geq 1$, we are able to compute in polynomial time a maximum size induced subgraph of G that is ℓ-colorable.

We require the use of an algorithm $\mathcal{A}$ that is a fully polynomial time dual approximation scheme for the following variant of the Knapsack problem. In this variant, we are given a fixed budget B on the total size of all the packed items, each item can be packed at most n times, and packing ℓ times of item i yields a profit that is a monotonically non-decreasing integer valued function of ℓ. The solution returned by $\mathcal{A}$ must have total size within $(1 + \varepsilon)$ multiplicative factor of the budget (i.e., at most $(1 + \varepsilon)B$), and total profit of packed items of at least the total profit of the optimal solution that uses only total size that is at most the given budget. Further, $\mathcal{A}$ should run in time polynomial in n and $\frac{1}{\varepsilon}$. We will show how to implement $\mathcal{A}$ in the sequel.

For each length class $\mathcal{J}_i$, denote by G^i the subgraph of G induced by the vertices that correspond to the jobs of $\mathcal{J}_i$.

Our scheduling algorithm operates in iterations as follows. The algorithm creates a set of batches in each iteration. The *length* of the iterations (referring to the sum of the batch lengths formed in the iteration) grows geometrically between iterations. In each iteration, given the current bound on the length, we pack a maximum number of jobs into batches, using a total length of time within a $1 + \varepsilon$ factor of the required length.

Pack_Subgraphs

1. Apply LENGTH ROUNDING

2. Pick uniformly at random a number β from the range $[0, 1)$ (independently of α). I.e., $\beta \sim \mathbf{U}[0, 1)$. Set $t = 0$, and set q to a constant value to be determined later.

3. While G is not empty do:

    (a) For $i = 0$ to k do:
        $a_i = q^{i+\alpha}$.
        For $\ell = 1$ to $|\mathcal{J}_i|$ do:
            Let $W_{i,\ell}$ be the vertex set of a maximum size ℓ-colorable subgraph of G^i.

    (b) Form a Knapsack instance where the profit from packing ℓ copies of the i-th element is $c_i(\ell) = |W_{i,\ell}|$, the size of the i-th element is a_i, and the total budget B on the total size of elements placed in the knapsack is $q^{t+\beta}$. Apply Algorithm $\mathcal{A}$ to approximate this Knapsack instance.

    (c) For $i = 0$ to k do:
        Assume that the approximate solution places b_i copies of the i-th element. Then, we use exactly b_i batches for the length class $\mathcal{J}_i$, and fill them by the jobs corresponding to a maximum size b_i-colorable induced subgraph of G^i (each batch formed by a color class of the subgraph).

10

(d) Remove the selected jobs from G and increase t by 1.

4. Apply the final step.

We now turn to analyze the performance of Pack_Subgraphs.

Lemma 6 Pack_Subgraphs *returns a feasible solution in polynomial time, for any $q > 1$.*

Proof. The algorithm terminates right after an iteration t that assigns the last remaining jobs in G. Observe that $t \leq \left\lceil \log_q \left(\sum_j p'_j \right) \right\rceil$, since this budget allows all jobs to be packed, e.g. each job in its own batch. Thus, the number of iterations is polynomial of the input size, since processing times are given in binary in the input. Each iteration is polynomial-time computable, since the graph G belongs to a graph class in $\mathcal{F}$. The feasibility of the returned solution follows since each batch packed is an independent set of G. ■

Consider the jobs sorted according to non-decreasing completion times in the solution returned by Pack_Subgraphs. Consider now the r-th job according to this order. Assume that this job belongs to the set of batches created in an iteration τ. We define the *modified completion time* of this job to be

$$\pi_r = (1 + \varepsilon) \cdot \left(\sum_{i'=0}^{\tau-1} q^{i'+\beta} + \frac{q^{\tau+\beta}}{2} \right).$$

Intuitively, the modified completion time will represent the halfway point in the processing of the set of batches containing the job. Define also $\pi(V) = \sum_{r=1}^{n} \pi_r$.

Instead of analyzing the solution we achieved, which orders all batches optimally by applying the final step, we analyze a solution which assigns the batches of each iteration consecutively. At each time the algorithm defines a set of batches for a new iteration, it assigns them right after the batches of the previous iteration. To assign the current batches, it is possible to use Smith's rule. In this way, the algorithm chooses the best order of these batches. Applying Smith's rule on the complete set of batches results in a solution that is not worse than this solution.

Lemma 7 *For any fixed value of α, the cost SOL_α of the solution returned by Pack_Subgraphs is at most $\pi(V) + p'(V)/2$.*

Proof. Consider the set S_τ of jobs selected in a given iteration $\tau = t$ of the while loop. By the performance guarantee of $\mathcal{A}$, we conclude that we use in this iteration a set of batches with total processing times of at most $(1+\varepsilon) \cdot q^{\tau+\beta}$. Therefore, the time σ_τ that a job waits for the sets of batches from previous iterations is at most $(1 + \varepsilon) \cdot \left(\sum_{i'=0}^{\tau-1} q^{i'+\beta} \right)$.

It remains to consider the time spent on waiting on other jobs selected within the same iteration. Define the *service time* of a job to be the intra-iteration completion time of the job, i.e. its completion time less σ_τ. Consider an arbitrary ordering of batches and its exact opposite order. The sum of the service time of a job j under the two orders equals the length of the batch plus the length of the job, or $(1 + \varepsilon) \cdot q^{\tau+\beta} + p'_r$. The ordering based on Smith's rule does no worse than the average of the two orders. Hence, the sum of the job service times in S_τ is at most $|S_\tau|(1 + \varepsilon) \cdot \frac{q^{\tau+\beta}}{2} + \frac{\sum_{j \in S_\tau} p'_j}{2}$. ■

Consider now the class $\mathcal{C}$ of solutions that given the length rounding parameterized by α, assign jobs of different rounded processing times to different batches. For any natural number

r, denote by d_r the minimum time in any solution from $\mathcal{C}$ that is needed to complete at least r jobs from V, and let

$$d(V) = \sum_{r=1}^{n} d_r.$$

It is straightforward to see that d_r is at most the completion time of the r-th job according to OPT_α. Summing up over all the vertices, we have the following observation.

Observation 8 $d(V) \leq \text{OPT}_\alpha$.

We note that the variables d_r, as well as OPT_α and p'_r are all random variables of α but independent of β. The variable π_r is also a random variable of α.

We next bound $\pi(V)$ in terms of $d(V)$.

Lemma 9 Let q be the root of the equation $\ln x = \frac{x+1}{x}$, that is $q \approx 3.591$. Then, for any fixed α, we have that $E_\beta\left[\pi(V)\right] \leq (1 + \varepsilon) \cdot 1.796 \cdot d(V)$.

Proof. Let $\delta_r = q^{\tau+\beta}$ for all r such that τ is the minimum index of an iteration such that we are able to pack at least r jobs during the first τ iterations. By definition, as noted above, the following inequality holds: $\pi_r \leq (1 + \varepsilon) \cdot \left(\sum_{i'=0}^{\tau-1} q^{i'+\beta} + \frac{q^{\tau+\beta}}{2}\right) \leq (1 + \varepsilon) \cdot q^{\tau+\beta} \cdot \left(\frac{1}{q-1} + \frac{1}{2}\right)$.

We now turn to bound $E[\frac{\delta_r}{d_r}]$ where the expectation is over the random selection of β. Let s be a real number such that $d_r = q^{s+\beta}$. Then, $\tau \leq \lceil s \rceil$, and $\frac{\delta_r}{d_r} = q^{\tau-s} \leq q^{\lceil s \rceil-s}$. Since $\beta \sim \mathbf{U}[0,1)$, so does $\lceil s \rceil - s$, and therefore $\frac{E[\delta_r]}{d_r} = E\left[\frac{\delta_r}{d_r}\right] \leq E\left[q^{\lceil s \rceil-s}\right] = \int_0^1 q^x dx = \frac{q-1}{\ln q}$. Therefore, we establish the following inequality:

$$E_\beta\left[\sum_{r=1}^{n} \pi_r\right] \leq (1 + \varepsilon) \cdot \left(\frac{1}{q-1} + \frac{1}{2}\right) \cdot E_\beta\left[\sum_{r=1}^{n} \delta_r\right]$$

$$\leq (1 + \varepsilon) \cdot \frac{q+1}{2(q-1)} \cdot \frac{q-1}{\ln q} \cdot \sum_{r=1}^{n} d_r$$

$$= (1 + \varepsilon) \cdot \frac{q+1}{2 \ln q} \cdot d(V) \ .$$

We choose $q \sim 3.591$ to minimize the expression $\frac{q+1}{2 \ln q} \sim 1.796$. ∎

Theorem 10 PACK_SUBGRAPHS *is a randomized* $(1.296e + \frac{3}{2} + \varepsilon)$*-approximation algorithm for* **MSJCT** *on graphs that belong to classes in* $\mathcal{F}$.

Proof. Consider first the cost SOL_α of the solution returned by the algorithm for a fixed value of α. By Lemma 7, the expected sum of *starting times* of the jobs, over the random choice of β, is at most $E_\beta[\pi(V)] - p'(V)/2$. Thus, by Lemmas 7 and 9 and Observation 8,

$$E_\beta\left[SOL_\alpha\right] \leq E_\beta\left[\pi(V)\right] - p'(V)/2 + p(V)$$

$$\leq (1 + \varepsilon) \cdot 1.796 \cdot d(V) + p(V) - p'(V)/2$$

$$\leq ((1 + \varepsilon) \cdot 1.796 \cdot e)\text{OPT}_\alpha + p(V) - p'(V)/2 \ .$$

We now bound the cost SOL of the algorithm solution in expectation over the random choices for α. We showed in the proof Lemma 4 that $E_\alpha[p'(V)] = (e - 1)p(V) \leq (e - 1)OPT$.

12

The optimum cost is also easily bounded from below by both $p(V)$ and by OPT_α/e. Hence, the expected cost of the algorithm's solution is bounded by

$$E[SOL] \le E_\alpha[((1+\varepsilon) \cdot 1.796 \cdot e)\mathrm{OPT}_\alpha + p(V) - p'(V)/2]$$

$$\le (1+\varepsilon) \cdot 1.796 \cdot OPT + \left(1 - \frac{e-1}{2}\right) p(V)$$

$$\le \mathrm{OPT} \cdot \left((1+\varepsilon) \cdot 1.796 \cdot e + \frac{3}{2} - \frac{e}{2}\right) .$$

The claim then follows by adjusting the definition of ε appropriately. ∎

In order to derandomize the algorithm, we could use a finite number of values for β that are $\delta > 0$ apart, and obtain an increase of the approximation ratio of the resulting algorithm within $\varepsilon(\delta) > 0$ where $\varepsilon(\delta) = e^\delta$. This is so because by picking the best value of β denoted as β^* from the interval $[0, 1)$ we can only improve the approximation ratio of the resulting algorithm, and since we test a value that is larger than β^* by at most δ, then the budget used for each iteration increases by a multiplicative factor of at most e^δ, and the claim follows. By scaling ε and δ accordingly, we can derandomize the algorithm without increasing the approximation ratio. Hence, we establish the following.

Proposition 11 *There is a deterministic $(1.296 \cdot e + \frac{3}{2} + \varepsilon)$-approximation algorithm for* **MSJCT** *on graphs that belong to $\mathcal{F}$.*

Note that this also improves the bound of Theorem 5 for interval graphs to 5.022.

We next analyze an improved algorithm for interval graphs, that combines the two algorithms.

Theorem 12 *Consider the algorithm that runs* PACK_SUBGRAPHS *and the algorithm for interval graphs of Section 3.3, and picks the better solution. It achieves a 4.912-approximation for* **MSJCT** *on interval graphs.*

Proof. By Theorem 5, the cost of the solution is at most $2e\mathrm{OPT} - 2p'(V) + p(V)$. Since we have already established that $E[p'(V)] = (e - 1)p(V)$, we have that the expected cost $E[SOL]$ of the returned solution (whose cost is the minimum of the two outputs) satisfies $E[SOL] \le 2e\mathrm{OPT} - (2e-3)p(V)$. By the proof of Proposition 11, we conclude that $E[SOL] \le (1.796e + \varepsilon)\mathrm{OPT} + \left(\frac{3-e}{2}\right)p(V)$. We multiply the first inequality by $\frac{3-e}{3e-3}$ and the second inequality by $\frac{4e-6}{3e-3}$. Then, we obtain $E[SOL] \le (1.807e + \varepsilon)\mathrm{OPT} \approx 4.912 \cdot \mathrm{OPT}$. ∎

We note that since both algorithms can be derandomized without affecting the approximation ratio, the same applies to this combined algorithm.

Implementing Algorithm $\mathcal{A}$ It remains to show how to implement Algorithm $\mathcal{A}$ for our Knapsack problem. Our algorithm is similar to the dual fully polynomial time approximation scheme for the 0-1 Knapsack problem [16]. We fix a value of ε such that $0 < \varepsilon < 1$.

We first present an exponential time dynamic programming procedure which produces an optimal solution of the problem. The input is an integer budget B, a set of k integer sizes $a_1, \ldots, a_k$ of items, and a profit function $c_i(\ell)$ for each size a_i (for $1 \le \ell \le n$). For $1 \le i \le k$ and $0 \le b \le B$, we denote by $F_{i,b}$ the maximum possible total profit from packing the items $i, i+1, \ldots, k$ using a total size of at most b (our goal is to compute $F_{1,B}$).

> **KnapDP**
>
> 1. for $b = 0$ to B set $F_{k+1,b} = 0$.
>
> 2. for $i = k$ downto 1 do:
>   for $b = 0$ to B do:
> $F_{i,b} = \max_{\ell=0,1,\ldots,n}\{F_{i+1,b-\ell a_i} + c_i(\ell)\}$, where $F_{i+1,b} = -\infty$ if $b < 0$.

Given the values of $F_{i,b}$ for all i and b, we compute an optimal solution by back-tracking the optimal path that leads us to the value of $F_{1,B}$.

We next follow the method of trimming the state-space (see [26]) in order to transform KNAPDP into a dual fully-polynomial time approximation scheme. To do so, we define a set of budget sizes $\mathcal{B} = \left\{-1, 0, 1, (1 + \frac{\varepsilon}{2k}), (1 + \frac{\varepsilon}{2k})^2, \ldots, (1 + \frac{\varepsilon}{2k})^{\lfloor \log_{1+\frac{\varepsilon}{2k}} B \rfloor}, B\right\}$. We next note that for a fixed value of i the sequence $F_{i,-1} = -\infty, F_{i,0}, F_{i,1}, \ldots, F_{i,B}$ is monotonically non-decreasing.

In our approximated scheme we will follow KNAPDP and compute $F_{i,b}$ for all i and for all $b \in \mathcal{B}$, and when referring to values of $F_{i+1,b}$ such that $b \notin \mathcal{B}$ we use the smallest value of $\mathcal{B}$ that is larger than b instead. For an integer number $b \leq B$ we denote $next(b)$ the minimum element of $\mathcal{B}$ that is at least b, i.e., $next(b) = \min\{b' \geq b : b' \in \mathcal{B}\}$. Our algorithm is stated as follows.

> **KnapDualFPTAS**
>
> 1. for all $b \in \mathcal{B} \setminus \{-1\}$, set $\tilde{F}_{k+1,b} = 0$ and set $\tilde{F}_{k+1,-1} = -\infty$.
>
> 2. for $i = k$ downto 1 do:
>   for all $b \in \mathcal{B}$ do:
> $\tilde{F}_{i,b} = \max_{\ell=0,1,\ldots,n} \quad \{\tilde{F}_{i+1,next(b-\ell a_i)} + c_i(\ell)\}$.

We can extend the definition of $F_{i,b}$ to non-integer values of b to denote the maximum profit of items $i, \ldots, k$ using a total size of at most b. Clearly $F_{i,b} = F_{i,\lfloor b \rfloor}$.

Lemma 13 $F_{i,b} \leq \tilde{F}_{i,b}$, for any $i = k+1, k, \ldots, 1$ and any $b \in \mathcal{B}$.

Proof. The proof is by induction on i. The base case of $i = k+1$ trivially holds as $F_{k+1,b} = \tilde{F}_{k+1,b}$ for any $b \in \mathcal{B}$. We assume that the claim holds for $i+1$ and show the claim for i.

$$\tilde{F}_{i,b} = \max_{\ell=0,1,\ldots,n}\{\tilde{F}_{i+1,next(b-\ell a_i)} + c_i(\ell)\} \geq \max_{\ell=0,1,\ldots,n}\{F_{i+1,next(b-\ell a_i)} + c_i(\ell)\}$$

$$\geq \max_{\ell=0,1,\ldots,n}\{F_{i+1,b-\ell a_i} + c_i(\ell)\} = F_{i,b} ,$$

where the first inequality holds by the assumption, and the second inequality holds by the monotonicity of $F_{i+1,b}$. ■

Therefore, we conclude that the solution that KNAPDUALFPTAS returns (via backtracking the path of $\tilde{F}_{1,B}$) has a total profit that is at least $F_{1,B}$ and therefore at least the profit of the optimal solution (that packs items with total size at most B). It remains to show that the total size of the packed items is at most $(1 + \varepsilon)B$.

Lemma 14 *The total size of the packed items in our solution is at most $(1 + \varepsilon)B$.*

Proof. The total size in each iteration of the outer loop of Step 2 increases at most by a multiplicative factor of $1 + \frac{\varepsilon}{2k}$. Therefore, the total size of the packed items is at most $B \cdot \left(1 + \frac{\varepsilon}{2k}\right)^k \leq B \cdot e^{\frac{\varepsilon}{2}} \leq B \cdot \left(1 + \frac{\varepsilon}{2} + \frac{\varepsilon^2}{4}\right) \leq B \cdot (1 + \varepsilon)$, where the inequalities hold since $\varepsilon < 1$ and $e^x \leq 1 + x + x^2$ for $x < 1.79$. $\blacksquare$

By lemmas 13 and 14 we conclude that KNAPDUALFPTAS is an implementation of Algorithm $\mathcal{A}$. Therefore, we establish the following proposition.

Proposition 15 *There is a polynomial time implementation of Algorithm $\mathcal{A}$.*

Proof. To prove this, we only need to show that $|\mathcal{B}|$ is of polynomial size, which is equivalent to showing that $\log_{1+\frac{\varepsilon}{2k}} B$ is polynomially bounded. Using $\frac{x}{1+x} \leq \ln(1+x)$, which holds for all positive values of x, we get $\log_{1+\frac{\varepsilon}{2k}} B = \frac{\ln B}{\ln 1 + \frac{\varepsilon}{2k}} \leq \frac{2k(1+\frac{\varepsilon}{2k}) \ln B}{\varepsilon}$. Using $\varepsilon < 1$, we get an upper bound of $\frac{4k \ln B}{\varepsilon}$, which is polynomial in the binary representation of B and in the size of the input. $\blacksquare$

3.5 A $\frac{4}{\rho}$-approximation algorithm

In this section we analyze the greedy algorithm from Section 2, to approximate the sum coloring instance defined as the set of jobs from $\mathcal{J}_i$. I.e., we first apply the preprocessing step and then each class is approximated separately. Then, we order the resulting batches according to our final step using Smith's rule.

Theorem 16 *There is a $\frac{4e}{\rho}$-approximation algorithm for problem **MSJCT** .*

Proof. By Lemma 3, the expected cost of OPT_α is at most e times the cost of OPT. On the rounded-up instance, both our greedy algorithm and OPT_α construct batches where each batch has a common processing time for all its jobs. Therefore, on the rounded-up instance, the two problems **MSBCT** and **MSJCT** are identical. Hence, by Theorem 1, the cost of the greedy solution is at most $\frac{4}{\rho}$ times the cost of OPT_α, and therefore it is a $\frac{4e}{\rho}$-approximation algorithm for **MSJCT**. $\blacksquare$

Recall that MIS can be solved in polynomial time on line graphs, and within $2/k - \epsilon$ factor on $k + 1$-claw free graphs for any $\epsilon > 0$ [15]. Thus, we have

Corollary 17 **MSJCT** *can be approximated within factor $4e$ on line graphs and $2ek + \epsilon$ for any $\epsilon > 0$ on $k + 1$-claw free graphs.*

We now show that the greedy algorithm applied directly gives the same ratio.

3.5.1 The greedy algorithm

In this section we show that the greedy algorithm of Section 2 provides the same approximation ratio for **MSJCT** as the above algorithm. We denote by OPT_J the optimal cost for **MSJCT**, and by OPT_B the optimal cost for **MSBCT** on the same instance.

We know that the cost of the solution returned by the greedy algorithm when considering the objective of **MSJCT**, denoted as $\mathbf{greedy}_J$, is at most the cost of the same solution when considering the objective of **MSBCT**, denoted as $\mathbf{greedy}_B$. By Theorem 1, we conclude that

greedy$_J \leq$ **greedy**$_B \leq \frac{4}{\rho} \cdot \mathrm{OPT}_B$. We note also that $\mathrm{OPT}_J \leq \mathrm{OPT}_B$. In order to show that the greedy algorithm yields a constant approximation for **MSJCT**, we prove the following lemma.

Lemma 18 $\mathrm{OPT}_B \leq e\mathrm{OPT}_J$.

Proof. Consider OPT_α which was defined in the statement of Lemma 3 to be an optimal solution to an instance in which we use the rounded up processing times p'_j. We further required that the solution must satisfy an additional requirement that for each batch, the jobs scheduled in the batch have a common class. Note that OPT_α is the same for both problems, since in each batch, all jobs have the exact same rounded processing time. Since in this solution jobs are only rounded up, we have $\mathrm{OPT}_B \leq \mathrm{OPT}_\alpha$. According to Lemma 3, choosing α randomly gives $E[\mathrm{OPT}_\alpha] \leq e \cdot \mathrm{OPT}_J$. Clearly, this means that there exists a value of α which we denote by α_1 for which $\mathrm{OPT}_{\alpha_1} \leq e \cdot \mathrm{OPT}_J$. Combining the two inequalities we get $\mathrm{OPT}_B \leq \mathrm{OPT}_{\alpha_1} \leq e\mathrm{OPT}_J$. ∎

We summarize in the next result.

Theorem 19 *The Greedy algorithm of Section 2 yields a $\frac{4e}{\rho}$-approximation for* **MSJCT** *.*

4   Approximating MSJCT via Linear Programming

In the following we describe a general scheme based on solving a linear programming formulation of our batch scheduling problems. We apply this scheme to obtain improved ratios for **MSJCT** in perfect graphs and in line graphs.

4.1   A Scheme for Non-preemptive Scheduling

Our scheme builds on one presented in [10] for the problem of *minimum sum multicoloring* graphs. The sum multicoloring problem is essentially the same as **MSJCT**, only without the restriction that jobs be started in batches. Namely, we are to find a non-preemptive schedule of the jobs, satisfying the non-concurrency of conflicting jobs, so as to minimize the sum of completion times of all jobs. For the sake of completeness, we first give an overview of the scheme of [10].

We first consider the following integer programming formulation. For each edge $(u, v) \in E$, there is a pair of variables $\delta_{uv}, \delta_{vu} \in \{0, 1\}$ such that $\delta_{uv} = 1$ if u precedes v in the schedule, and 0 otherwise. Let N_v denote the set of neighbors of v in G. We denote by $C_1, \ldots, C_{M_v}$ the set of maximal cliques in the subgraph induced by N_v.

$$(IP) \qquad \text{minimize} \sum_{v \in V} f_v$$

$$\text{subject to:} \quad \forall v \in V, \ \forall r, 1 \leq r \leq M_v \ : \ f_v \geq p_v + \sum_{u \in C_r} p_u \delta_{uv} \qquad (2)$$

$$\forall uv \in E \qquad : \ \delta_{uv} + \delta_{vu} = 1$$

$$\forall uv \in E \qquad : \ \delta_{uv} \in \{0, 1\}$$

In the linear relaxation of IP denoted as LP we replace the binary constraint with $\delta_{uv} \geq 0$. Let f^*_v denote the completion time of job J_v in the optimal solution for LP. Let $OPT^* = \sum_v f^*_v$ be the cost of an optimal solution for LP. Recall that $p(V) = \sum_{v \in V} p_v$ and $p'(V) = \sum_v p'_v$.

After solving LP,[3] we partition the jobs to *blocks* of increasing f_v^* values as follows. Let α be a value chosen uniformly at random from $[0,1)$, and $\beta = 3.5911$. Let L be the smallest value such that $\beta^{\alpha+L} \geq p(V)$. Define $V_\ell = \{v \in V : \beta^{\alpha+\ell-1} < f_v^* \leq \beta^{\alpha+\ell}\}$, $\ell = 0, \ldots, L$, to be the subset of vertices whose f_v^* values fall in the interval $(\beta^{\alpha+\ell-1}, \beta^{\alpha+\ell}]$.

We apply to each block V_ℓ an algorithm $\mathcal{A}$ for non-preemptive multicoloring (i.e., scheduling) that minimizes the total number of colors used (i.e., the makespan of the schedule). Finally, we concatenate the schedules obtained for the blocks, taking first the schedule for V_0, then the schedule for V_1 and so on, ending with the schedule for V_L.

For a block V_ℓ, let $\omega(V_\ell)$ be the maximum size of a clique in the subgraph induced by V_ℓ. We require that the coloring algorithm $\mathcal{A}$ gives an approximation in terms of the maximum clique size. Namely, let $\bar{\rho}$ be such that

$$\mathcal{A}(V_\ell) \leq \bar{\rho} \cdot \omega(V_\ell), \text{ for } \ell = 0, 1, \ldots, L.$$

The following result is given in [10].

Theorem 20 *Let G be a graph with processing times p_v, and let $\mathcal{A}$ and $\bar{\rho}$ be given as above. Then, the LP schema gives a non-preemptive schedule of G, whose sum of completion times of the jobs is at most $3.591\bar{\rho}OPT^* + p(V)/2$.*

4.2 The Scheme for MSJCT

Consider the following algorithm for solving an instance of MSJCT.

Algorithm Job_Batch (JB)

1. Apply LENGTH ROUNDING (of Section 3.1) for partitioning $\mathcal{J}$ to length classes by rounded processing times.

2. For any pair of jobs J_i, J_j that belong to different classes, add an edge (i,j) in the intersection graph G. Denote the resulting graph G'.

3. Solve LP for the input jobs with rounded processing times and intersection graph G'.

4. Partition the jobs in the input into blocks $V_0, V_1, \ldots, V_L$, by their LP completion times.

5. Schedule the blocks in sequence using for each block a coloring algorithm for unit length jobs.

Note that in Step 5 we use for each of the blocks an ordinary coloring algorithm and not a multi-coloring algorithm.

Theorem 21 JB *approximates* **MSJCT** *within a factor of $9.761\bar{\rho} + (1 - \frac{e-1}{2})$.*

Proof. Observe that after adding the edges to G and rounding up the lengths, any non-preemptive solution to G' is a solution for **MSJCT** on G' (and on G). Let $OPT(G')$ denote

---

[3]We refer below to graph classes for which LP is solvable in polynomial time.

the cost of a minimum cost solution to **MSJCT** on G' with rounded processing times. Note that $OPT(G') = OPT_\alpha$. By Theorem 20, the sum of completion times in the solution output by JB is at most $3.591\overline{\rho}OPT^* + p'(V)/2$. Observe that $OPT^* \leq OPT(G')$ and, by Lemma 3 and its derandomization, $OPT(G') \leq e \cdot OPT(G)$. It follows that the sum of completion times of all jobs with non-rounded processing times is at most $3.591\overline{\rho}eOPT(G) - \frac{p'(V)}{2} + p(V)$. Thus,

$$E[JB] \leq 3.591\overline{\rho}e \cdot OPT(G) + p(V) - \frac{E[p'(V)]}{2} \leq 9.761\overline{\rho} \cdot OPT(G) + p(V)(1 - \frac{e-1}{2}) \ .$$

The second inequality follows from the fact that $E[p'(V)] = (e-1)p(V)$, and since $OPT(G) \geq p(V)$, this gives the statement of the theorem. ∎

We note that in general LP may not be solvable in polynomial time on G'. Indeed, this is due to the fact that the number of maximal cliques in the neighborhood of each vertex in G' may be exponential, leading to an exponential size LP. In the following we give a condition that will be used for applying JB to certain graph classes.

Lemma 22 *If the maximum weight clique problem is polynomial-time solvable on G, then LP can be solved in polynomial time for G'.*

Proof. We give a polynomial-time separation oracle which, given a solution to LP, tests whether all constraints are satisfied (with arbitrary precision). This leads to a polynomial time algorithm for solving LP.

Given a solution for LP and a vertex $v \in V$, we set for each vertex $u \in N_v$, $p'_u = p_u \delta_{uv}$. Compute the maximum weight clique in each length class in N_v (in G) with respect to p' using the assumed clique algorithm for G, and take the union of these cliques. Given the way G' is constructed, this gives a maximum weight clique in N_v. We can then simply test whether f_v satisfies the constraint (2) by checking whether the inequality holds for this maximum weight clique. Testing the other constraints is trivial. ∎

In particular, the weighted clique problem and the coloring problem are polynomial-time solvable on perfect graphs [13].

Corollary 23 JB *is a 9.9-approximation algorithm for* **MSJCT** *on perfect graphs.*

The number of maximal cliques of a line graph is at most n, thus LP is solvable in polynomial time. Each block can be colored by Vizing's theorem (see e.g., in [17]) using at most $\omega(V_\ell) + 1$ colors. Therefore, we have that $\overline{\rho} = 1 + o(1)$. We conclude the following theorem.

Theorem 24 JB *is a $9.9 + o(1)$-approximation algorithm for* **MSJCT** *on line graphs.*

More generally, if we have line multigraphs, we can apply the recent polynomial time approximation scheme of Sanders and Steurer [22] for edge coloring multigraphs. This gives us $\overline{\rho} = 1 + \epsilon$, for any $\epsilon > 0$, and a ratio of $9.9 + \epsilon$ for **MSJCT**.

5 Concluding Remarks

We have shown the usefulness of combining various forms of randomized geometric partitioning to obtain improved approximation ratios for generalizations of weighted sum coloring, such as batch scheduling with minsum objectives. We expect that such combined techniques would be useful in solving other scheduling problems involving graph coloring constraints. We have

found that certain randomization techniques are more suitable for certain classes of graphs. Thus, in applying these techniques, we distinguish, e.g., between interval and comparability graphs, and the classes of line and perfect graphs.

Our results leave open the approximability (and hardness) of **MSBCT** and **MSJCT** on several natural classes of graphs, including k-trees, trees, or even paths. On the other hand, we are not aware of any non-trivial graph class that is polynomially solvable (beyond stars).

References

[1] 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.

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

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

[4] M. Chrobak, C. Kenyon-Mathieu. Online algorithm column 10: Competitiveness via doubling. *SIGACT News*, Dec. 2006.

[5] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[6] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson and A. S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.* **14**:744–780, 1985.

[7] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math* **17**:449–467, 1965.

[8] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[9] A. Frank. On chain and antichain families of a partially ordered set. *Journal of Combinatorial Theory Series B*, 29:176–184, 1980.

[10] R. Gandhi, M. M Halldórsson, G. Kortsarz and H. Shachnai, Improved Bounds for Sum Multicoloring and Scheduling Dependent Jobs with Minsum Criteria. *2nd Workshop on Approximation and Online Algorithms (WAOA)*, Bergen, September 2004.

[11] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman & Co., New York, 1979.

[12] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[13] M. Grötschel, L. Lovász and A. Schrijver. Polynomial algorithms for perfect graphs, *Topics on Perfect Graphs*, 325–356, North-Holland Math. Stud., 88, North-Holland, Amsterdam, 1984.

[14] M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37(3):187–209, 2003.

[15] C. A. J. Hurkens, and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discrete Math.*, vol. 2, 1989, pp. 68–72.

[16] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[17] T. R. Jensen and B. Toft. Graph Coloring Problems. Wiley-Interscience Series in Discrete Optimization, 1995.

[18] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[19] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[20] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. In *Proc. of the 10th International Conference on Database Theory (ICDT2005)*, pages 83–98, 2005.

[21] S. Nicoloso, M. Sarrafzadeh, and X. Song. On the sum coloring problem on interval graphs. *Algorithmica*, 23(2):109–126, 1999.

[22] P. Sanders, D. Steurer. An Asymptotic Approximation Scheme for Multigraph Edge Coloring. In *Proc. of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 897–906, 2005.

[23] A. Schrijver. *Combinatorial optimization: Polyhedra and efficiency*. Algorithms and Combinatorics Series Vol. 24. Springer, 2003.

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

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

[26] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.

[27] M. Yannakakis and F. Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Information Processing Letters*, 24(2):133–137, 1987.