Approximating Minimum Sum Coloring with Bundles

by

Seyed Parsa Darbouy

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

This study introduces a new problem called the Minimum Sum Coloring with Bundles. We are given an undirected graph $G = (V, E)$ and (not necessarily disjoint) bundles $V_1, V_2, \ldots, V_p \subseteq V$ with associated weights $w_1, \ldots, w_p \geq 0$. The goal is to give a proper coloring of G using positive integers to minimize the weighted average/total completion time of all bundles, where the completion time of a bundle is the maximum integer assigned to one of its nodes. This is a common generalization of the classic Minimum Sum Coloring problem, i.e. when all bundles are singleton nodes, and the classic Chromatic Number problem, i.e. the only bundle is all of V.

We provide the first constant-factor approximation in perfect graphs and, more generally, graphs whose chromatic number is within a constant factor of the maximum clique size in any induced subgraph.

Next, we extend our results to get constant-factor approximations for a general model where the bundles are disjoint (i.e. can be thought of as jobs brought by the corresponding client) and we are only permitted to color/schedule a bounded number of jobs from each bundle at any given time. Specifically, we get constant-factor approximations for this model for interval graphs and generalizations.

Preface

This dissertation is an original work by the author, Seyed Parsa Darbouy. This research project has been published in the Scandinavian Symposium and Workshops on Algorithm Theory (SWAT). The published paper was coauthored with Zachary Friggstad; P. Darbouy developed the technical aspects of the paper.

"Computer science is no more about computers than astronomy is about telescopes."

– Edsger Dijkstra

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Zachary Friggstad, for his unwavering support, insightful guidance, and constant encouragement throughout my journey. His expertise and dedication have been instrumental in shaping this research and my development as a scholar.

My sincere thanks extend to the members of my thesis committee for their valuable time, constructive feedback, and thoughtful questions. Their insights have significantly enhanced the quality of this dissertation.

I am eternally grateful for the love and support of my family. Their endless patience, unwavering belief in me, and countless sacrifices have made this achievement possible.

Contents

1 Introduction **1**
1.1 Results . . . 4
1.2 Preliminaries . . . 7
1.2.1 Approximation Algorithms . . . 7
1.2.2 Linear Programming . . . 7
1.2.3 Chordal Graphs . . . 8
1.2.4 Interval Graphs . . . 9
1.2.5 Perfect Graphs . . . 9
1.3 Related Work . . . 9
1.4 Organization . . . 11

2 Approximating MSCB in Perfect Graphs **12**
2.1 Challenges . . . 12
2.2 Algorithm . . . 15
2.3 Rounding Algorithm . . . 17
2.4 Analysis . . . 18
2.5 Extensions . . . 20

3 MSCB with Task Concurrencies **24**
3.1 Extensions . . . 27
3.2 **MSCB-TC** in Perfect Graphs - A Barrier . . . 28

4 Conclusion **30**

References **31**

List of Tables

1.1 Best Known Results of Approximation Algorithms for **MSC** in Special Graph Classes. 10

List of Figures

2.1 An example of the reduction from an instance of the Maximum Independent Set problem to a collection of intervals using $t = 2$. It is straightforward to verify a subset of nodes is independent if and only if each point on the underlying line is spanned by at most t intervals in the union of their bundles. . 14

3.1 Tightness of Lemma 2 for Interval Graphs. 26

Chapter 1

Introduction

Scheduling problems form a broad and extensively researched class of combinatorial optimization problems. They center around the allocation of limited resources to a set of tasks or jobs over time, subject to various constraints, with the goal of optimizing a defined objective. Scheduling problems typically involve several core components:

Resources, which can represent machines in a manufacturing environment, processors in a computing system, runways at an airport, personnel, or other limited assets; tasks or jobs, which are units of work with characteristics such as processing times, deadlines, release dates (when they become available), and possible precedence relationships; and constraints that define the rules and restrictions governing feasible schedules, such as resource capacity limitations, precedence relationships between tasks, and deadlines or due dates. The objective function quantifies the quality of a schedule, with common objectives including minimizing the makespan (total completion time), minimizing average job completion times, minimizing tardiness (how late jobs finish relative to their due dates), or maximizing the number of completed tasks.

Scheduling problems exhibit an enormous variety based on the types of resources, task characteristics, constraints, and the objective function. Many classic scheduling problems are **NP**-hard, making the search for optimal solutions computationally intractable for large instances. This intractability motivates the development of approximation algorithms.

Graph coloring is a fundamental problem in combinatorial optimization

with far-reaching applications. It involves assigning colors to the vertices of a graph such that no two adjacent vertices share the same color. This assignment is called proper coloring.

Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a proper $k-$coloring assigns one of k colors to each vertex in V. Adjacent vertices (connected by an edge) must receive different colors. The chromatic number of a graph G, denoted by $\chi(G)$, is the smallest number of colors needed for a proper coloring.

Graph coloring has direct connections to various scheduling problems. Consider tasks that need to be scheduled on shared resources, where certain pairs of tasks conflict and cannot be scheduled simultaneously. We can construct a graph where vertices represent tasks, and edges represent conflicts. A proper coloring of this graph then corresponds to a valid schedule where no conflicting tasks overlap in time.

In a scheduling problem, a common objective is to minimize the sum of completion times of tasks, ensuring no constraint violations. As we said earlier a task can be represented by a vertex and edges represent conflicts between resources. To schedule these tasks we can use graph coloring to give each of them a number that represents a color. The objective, then, becomes minimizing the sum of these assigned color values. This is known as the Minimum Sum Coloring (**MSC**) problem. This is a well-studied problem lying at the intersection of scheduling theory and graph coloring.

Formally, In **MSC**, we are given an undirected graph $G = (V, E)$ on n nodes. The goal is to find a **proper coloring** $\chi : V \to \{1, 2, \ldots\}$ of nodes of V, i.e. $\chi(u) \neq \chi(v)$ for all $uv \in E$, with minimum total color $\sum_{v \in V} \chi(v)$.

While different than standard graph coloring where the goal is to simply minimize the number of distinct colors of the coloring, it is essentially just as hard to approximate. That is, unless $\mathbf{P} = \mathbf{NP}$, there is no $n^{1-\delta}$-approximation for any constant $\delta > 0$ [2]. However, in certain cases, it is possible to get improved approximations. For example, if it is possible to efficiently compute a maximum independent set in G and any of its induced subgraphs then greedily coloring by computing a maximum independent set I, coloring I with the

next unused integer, and then removing I from G yields a 4-approximation [2]. There is a large body of work on getting improved constant-factor approximation in more structured special cases or obtaining constant-factor approximations in other graph classes, see the related works section below for further discussion.

This work introduces two generalized versions of the Minimum Sum Coloring problem (**MSC**) that address task bundling. Unlike **MSC**, where tasks are considered individually, clients can now submit groups of tasks (bundles) and prioritize the completion time of all tasks within each bundle. The first version is as follows.

Definition 1. *In the* Minimum Sum Coloring with Bundles *problem (***MSCB***), we are given an undirected graph $G = (V, E)$ and collection of bundles $V_1, \ldots, V_p \subseteq V$ with associated weights $w_1, \ldots, w_p \geq 0$. The goal is to find a proper coloring $\chi : V \to \{1, 2, \ldots, |V|\}$ of V in a way that minimizes the total weighted makespan of all bundles. That is, the coloring should minimize $\sum_{i=1}^{p} w_i \cdot \max_{v \in V_i} \chi(v)$.*

By allowing bundles with multiple tasks, **MSCB** encompasses both **MSC** and finding the chromatic number. **MSC** is a single-tasks bundles version and finding the chromatic number is the single bundle with all tasks. Therefore, **MSCB** is a more general problem.

The second general version introduces the concept of limited concurrent processing within bundles. For example, imagine clients with specific bandwidth or resource constraints, preventing them from delivering or receiving more than a certain number of tasks at a time. As in this case, clients have their own specific limits, and the tasks in different bundles cannot overlap as they come from different clients, i.e. bundles constitute a partition $V_1, \ldots, V_p$ of V. We consider the following generalization of **MSCB**

Definition 2. *In the* Minimum Sum Coloring with Bundles and Task Concurrency *problem (***MSCB-TC***), we are given a graph $G = (V, E)$ and a* **partition** *$V_1, \ldots, V_p$ of V with associated weights $w_1, \ldots, w_p \geq 0$. Further, for each $1 \leq i \leq p$ we are given a bound $d_i \geq 1$ on the number of jobs from*

bundle i that may be processed at any moment. The goal is still to find a proper coloring χ that minimizes the total weighted completion time of all bundles, but we further require $|\{v \in V_i | \chi(v) = t\}| \leq d_i$ for each $1 \leq i \leq p$ and each time step/color t.

1.1 Results

Interval graphs (check definition in 1.2.4) hold particular interest for scheduling problems due to their ability to naturally represent tasks with defined start and end times. Each vertex in an interval graph maps to a single task, and an edge connecting two vertices signifies that these tasks cannot be scheduled to overlap. This simple graph model captures the core constraint of scheduling – tasks cannot be done simultaneously if they conflict. This natural fit between interval graphs and scheduling problems motivated us to base our initial work on them. We further extended our research to encompass perfect graphs (check definition in 1.2.5).

Our first main result is a constant-factor approximation for **MSCB** in interval graphs which can be applied more generally to perfect graphs.

Theorem 1. MSCB *on perfect graphs admits a polynomial-time* 10.874*-approximation.*

Our presentation will first focus on proving Theorem 1 in the case of interval graphs. The linear programming (LP) relaxation we use in this case is simpler than in the general case of perfect graphs. After presenting the algorithm for interval graphs, we discuss the few necessary changes to extend it to perfect graphs in general.

Our techniques extend further to classes of graphs for which the chromatic number is approximately equal to the maximum clique number and these quantities can be approximated in polynomial time. Namely, we get the following extension. Recall a graph class $\mathcal{G}$ is **hereditary** if for any $G \in \mathcal{G}$ we have $G[U] \in \mathcal{G}$ as well for all induced subgraphs of G.

Corollary 1. *Let $\mathcal{G}$ be any hereditary graph class with the following properties: (a) the maximum clique size is within a constant factor of the chromatic number of any $G \in \mathcal{G}$, (b) given vertex weights $w_v, v \in V$ for a graph $G \in \mathcal{G}$, there is a constant-factor approximation algorithm for the maximum-weight clique of G, and (c) there is a constant-factor approximation for coloring any $G \in \mathcal{G}$ with the minimum number of colors. Then there is a constant-factor approximation for* **MSCB** *for graphs in $\mathcal{G}$.*

For example, we get $O(1)$-approximations for the following graph classes:

- Unit disc graphs: when vertices are associated with discs of radius 1 in the plane and edges indicate when two discs intersect. The classic algorithm in [6] for computing a maximum-size clique easily generalizes to compute a maximum-weight clique in polynomial time. It is possible to color any unit disc graph with maximum clique size K using fewer than $3 \cdot K$ colors in polynomial time [27].

- Circular-arc graphs: when vertices are associated with arcs of a circle and edges indicate when two such arcs intersect. A maximum weight clique can be found in polynomial time, e.g. [4]. One can efficiently $2 \cdot K$-color a circular arc graph with maximum clique size K by first coloring the arcs spanning one particular point with at most K colors. After removing these arcs, we are left with an interval graph which can also be colored with K additional colors since interval graphs are perfect.

To the best of our knowledge, **MSCB-TC** has not been previously studied even in special cases. We obtain constant-factor approximations for this problem, though in more restricted settings. Recall that a graph G is chordal if every cycle of length ≥ 4 has a chord. That is, if $v_1, v_2, \ldots, v_\ell, v_1$ is a cycle of length $\ell \geq 4$ then we also have $v_i v_j \in E$ for some $1 \leq i, j \leq \ell$ where i, j are not consecutive indices around the cycle, i.e. $1 \leq i \leq j - 1$ and if $i = 1$ then $j \neq \ell$. Interval graphs, an important class of graphs in scheduling, are chordal.

Theorem 2. MSCB-TC *in chordal graphs admits a polynomial-time 16.31-approximation.*

The key property of chordal graphs that drives our algorithm is that they admit **perfect elimination orderings**. That is, it is possible to compute an ordering $v_1, v_2, \ldots, v_n$ of all nodes V such that the left-neighborhood $N_\ell(v_i) := \{v_j : j < i \text{ and } v_i v_j \in E\}$ is a clique for all $1 \leq i \leq n$. In fact, a graph is chordal if and only if it admits such an ordering and this ordering can be computed in linear time [14]. Our techniques extend more generally to other classes of graphs.

Corollary 2. *Let $\mathcal{G}$ be a hereditary graph class with the same properties as in Corollary 1. Further, for any $G \in \mathcal{G}$ suppose we can compute an ordering $v_1, \ldots, v_n$ of its nodes in polynomial time such that the left-neighborhood $N_\ell(v_i)$ can be covered by a constant number of cliques in polynomial time.* **MSCB-TC** *has a constant-factor approximation when restricted to inputs whose graphs lie in $\mathcal{G}$.*

For example, such an ordering exists for unit disk graphs (with each left-neighborhood being covered by ≤ 3 cliques) [27]. Such an ordering can be also found for circular arc graphs with each left-neighborhood being covered by ≤ 2 cliques, i.e. consider the coloring algorithm mentioned above: if one orders the arcs spanning the selected point and then orders the remaining arcs according to a perfect elimination ordering in the resulting interval graph.

The algorithm from Theorem 2 requires one additional structural result about coloring than the algorithm from Theorem 1, namely Lemma 2 in Section 3. Unfortunately this structural result fails to hold in perfect graphs, which is why Theorem 2 is only for chordal graphs. Still, we are able to recover the following.

Theorem 3. *There is an $O(\sqrt{n})$-approximation for* **MSCB-TC** *in perfect graphs.*

While the ratio is quite large, it is at least better than the lower bound in general graphs of $n^{1-\delta}$ for any constant $\delta > 0$, which is inherited from the same hardness for **MSC** [2].

1.2 Preliminaries

This section establishes the fundamental concepts and principles necessary to understand the methodologies and strategies explored throughout the remainder of the thesis.

1.2.1 Approximation Algorithms

Many real-world optimization problems are computationally intractable, meaning they cannot be solved to optimality within a reasonable time frame. This intractability often stems from the inherent difficulty of these problems, classified as **NP**-hard. Approximation algorithms offer a powerful approach to tackle these challenges by finding near-optimal solutions within polynomial time.

An approximation algorithm for an optimization problem has a provable approximation ratio (or performance guarantee). Let $OPT(I)$ represent the cost (or value) of an optimal solution for an instance I of the problem, and let $A(I)$ represent the cost (or value) of the solution produced by the approximation algorithm A. For a minimization problem, algorithm A has an approximation ratio of α if for all instances I, $A(I) \leq \alpha \cdot OPT(I)$. Conversely, for a maximization problem, the inequality is reversed: $A(I) \geq (1/\alpha) \cdot OPT(I)$. The value α is typically greater than or equal to 1, and the closer α is to 1, the better the approximation provided by the algorithm.

A classic example of an approximation algorithm is the Vertex Cover problem, where the goal is to find the smallest set of vertices in a graph such that every edge has at least one endpoint in the set. There is a 2-approximation algorithm for this problem. While not optimal, it finds a solution within twice the size of the smallest possible vertex cover in polynomial time.

1.2.2 Linear Programming

Linear Programming (LP) is a powerful optimization technique central to the design and analysis of approximation algorithms. It involves optimizing a linear objective function subject to a set of linear constraints.

$$\begin{aligned} \textbf{minimize}: \quad & c^T x \\ \textbf{subject to}: \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Here c denotes the vector of objective function coefficients, x represents the vector of decision variables, b is the vector containing the right-hand side constants of the constraints, and A is the matrix that holds the constraint coefficients.

A feasible solution to a linear program is an assignment of values to the decision variables (x) that satisfies all constraints of the problem. The values in x must adhere to the system of linear inequalities defined by $Ax \geq b$. This means any potential solution must fall within the boundaries specified by the constraints. Additionally, each element of the vector x must be greater than or equal to zero.

Solving linear programs with integer constraints (Integer Linear Programming, ILP) can be computationally expensive. A common approach to tackle ILP problems is to use LP relaxation, where the integer constraints are relaxed, allowing the variables to take on continuous values. This relaxed problem is solvable using algorithms like the simplex method or ellipsoid method. The time complexity of the simplex method is not polynomial; in practice, however, the simplex method is very fast in most cases. LPs can be solved in polynomial time using the ellipsoid method [15]. It provides a lower bound (for minimization) or upper bound (for maximization) on the optimal objective value of the original ILP problem. The solution to the relaxed problem can then be used to guide approximation algorithms.

1.2.3 Chordal Graphs

A chordal graph, in the realm of graph theory, is defined as a graph where every cycle of four or more vertices contains a chord. A chord is an edge that connects two non-consecutive vertices within the cycle but is not itself part of the cycle. Chordal graphs possess various interesting properties and have numerous subclasses, including interval graphs, split graphs, and Strongly chordal graphs, each with their own specific characteristics and applications.

1.2.4 Interval Graphs

An interval graph, denoted by $G = (V, E)$, is an undirected graph where each vertex $v \in V$ corresponds to a closed interval I_v on the real line. Two vertices, u, and v, are connected by an edge $(u, v) \in E$ if and only if their corresponding intervals I_u and I_v intersect.

1.2.5 Perfect Graphs

A graph G is **perfect** if the maximum clique size in $G[U]$ equals the chromatic number of $G[U]$ for any $U \subseteq V$ where $G[U] = (U, \{uv \in E : u, v \in U\})$ denotes the subgraph of G induced by U. Examples of perfect graphs include bipartite graphs, line graphs of bipartite graphs, interval graphs, comparability graphs, split graphs, permutation graphs, and chordal graphs as well as the edge-complements of these graphs. For a broader discussion of perfect graphs and other graph classes that we mentioned, we refer the reader to Golumbic's book [14].

1.3 Related Work

MSCB has been studied in certain special cases. The most notable example is Coflow Scheduling, which is equivalent to **MSCB** when the input graph is the line graph[1] of a bipartite graph (i.e. at any given time step a matching of the edges/jobs is scheduled). This problem was first introduced in [28] where a 22.34-approximation was given. Later, 4-approximations were given for Coflow Scheduling and generalizations with release times for the jobs [1], [11], [30].

In a matroid setting (see [26] for a full definition), the jobs are given as items in the ground set X of a matroid rather than as vertices in a graph and bundles are subsets of items in X. The set of jobs scheduled at any given time must form an independent set of the matroid. A 2-approximation for the problem of minimizing the total weighted completion time of all bundles was

[1]Recall the line graph of $G = (V, E)$ is the graph $L(G) = (E, F)$ where two edges $e, f \in E$ are considered adjacent in $L(G)$ if they share a common endpoint.

	MSC	
Graph Class	upper bounds	lower bounds
General graphs	$O(n/\log^3 n)$ [2], [9]	$O(n^{1-\epsilon})$ [2], [10]
Perfect graphs	3.591 [12]	**APX** [3]
Chordal graphs	1.796 [8]	**APX** [25]
Interval graphs	1.796 [12]	**APX** [25]
Bipartite graphs	27/26 [24]	**APX** [3]
Partial k-trees	1 [22]	
Planar graphs	PTAS [18]	**NP**-complete [18]
Trees	1 [23]	
Line graphs	1.8298 [19]	**NP**-complete [2]
Line graphs of trees	1	
$[k+1]$-claw free graphs	$O(k)$	**NP**-complete [2]
Intersection of k-sets	$O(k)$ [2]	**NP**-complete [2]
Unit disc graphs	2 [5]	**NP**-complete [5]

Table 1.1: Best Known Results of Approximation Algorithms for **MSC** in Special Graph Classes.

given in [21].

MSC itself is much more well studied. As mentioned earlier, a 4-approximation is known in settings where the maximum independent set of the graph (and any induced subgraph) can be computed in polynomial time [2]. Special attention has been given to particular graph classes, in particular a 1.796-approximation is known in interval graphs [17] which was recently extended to an algorithm with the same approximation guarantee for chordal graphs [7]. In the more general setting of perfect graphs, a 3.591-approximation is known [12]. A broader summary of approximation algorithms for **MSC** in special graph classes can be found in the survey article by Halldórsson and Kortsarz [16]. The following table from the survey presents results for **MSC** applied to special graph classes.

In the table provided, n denotes the number of vertices. Also, **APX** signifies that it is **NP**-hard to approximate within some constant factor greater than 1. While we do not define the special graph classes in this table, you can find their definitions in [16].

1.4 Organization

After discussing some challenges in adapting previous **MSC** and Coflow Scheduling algorithms to **MSCB**, Section 2 presents our algorithm for **MSCB** and the proofs of Theorem 1 and Corollary 1. Section 3 extends these techniques to **MSCB-TC** and proves Theorem 2 and Corollary 2. We also discuss why our **MSCB-TC** algorithm does not extend to perfect graphs in general, point out that we can at least get an $O(\sqrt{n})$-approximation in perfect graphs for **MSCB-TC**, and leave further improvements for future work.

Chapter 2

Approximating MSCB in Perfect Graphs

As mentioned in Chapter 1, **MSC** is a well-studied problem with several approximation algorithms. One simple approach employs a greedy strategy. We iteratively identify the largest set of vertices that can be assigned the same color, color them, and remove them from the remaining graph. This process repeats until no such sets remain. This greedy approach can be formalized by finding the maximum independent set in the graph [16]. This greedy algorithm can be applied to any graph class where maximum independent sets can be computed efficiently. This achieves a 4-approximation for **MSC** [2].

One more complicated strategy is to find 2^k-colorable subgraphs, where k is iteratively increased from 0. This strategy achieves a constant approximation ratio for any graph class where their maximum k-colorable subgraphs can be computed. Interval graphs are an example where this computation is feasible [17]. We can improve the approximation ratio of this strategy by employing a geometric base other than 2, introducing a random geometric offset ($2^{k+\alpha}$), and randomly permuting the order of colors in a 2^k coloring.

2.1 Challenges

While **MSCB** is a common generalization of **MSC** and Coflow Scheduling, it turns out most techniques used to address these problems fail. For example, the 4-approximation for **MSC** from iteratively finding a maximum

independent set in the graph of unscheduled tasks in each time step. But for the classic problem of computing the chromatic number of a graph, i.e. the special case of **MSCB** where all nodes are in a single bundle, this can be as bad as an $\Omega(\log n)$-approximation even if the underlying graph is a tree.

We point this out to show that the greedy 4-approximation for unweighted **MSC** does not extend to our setting which includes, as a special case, computing the chromatic number of a graph. While this seems to be well known in the community, we are unaware of a particular reference for such an example so we provide a simple construction here for completeness.

Let T_0 be the trivial tree with a single node. Inductively for $i \geq 1$ let T_i be constructed by attaching 2 leaf nodes to each node of T_{i-1}. So the number of nodes in T_i is 3^i.

The only maximum independent set in T_i is the set of all leaves in T_i (which clearly forms an independent set). To see this, let I be an independent set that includes an internal vertex of T_i (i.e. a node of T_{i-1}). Neither leaf that was attached to v to form T_i is in I because I is an independent set. But then we get a larger independent set by removing v from I and adding in the two associated leaves.

The greedy algorithm to compute a maximum independent set in T_i will first pick all of its leaves. Removing them leaves tree T_{i-1}. So by induction, with the base case $i = 0$ clearly requiring a single iteration to color, the number of iterations will be $i+1$. Since $i+1$ is logarithmic in the size of T_i (i.e. $n := 3^i$) and since the chromatic number of $T_i, i \geq 1$ is 2 (as is true for any tree with at least one edge), this is an $\Omega(\log n)$-approximation for the chromatic number of a tree. Therefore, it is not an $O(1)$-approximation for **MSCB**.

Another strategy that is used to get refined approximations for **MSC** is to compute maximum t-colorable subgraphs of G for a geometric sequence of values for t as in [7], [17]. For **MSCB**, one could consider an algorithm that for a geometric sequence of values t will compute a maximum-size subset of bundles $\mathcal{P} \subseteq \{1, 2, \ldots, p\}$ such that all nodes in these bundles, i.e. $\cup_{j \in \mathcal{P}} V_j$ can be scheduled without conflict in t steps. That is, we do not just compute a maximum-size t-colorable induced subgraph of G itself, rather we are con-

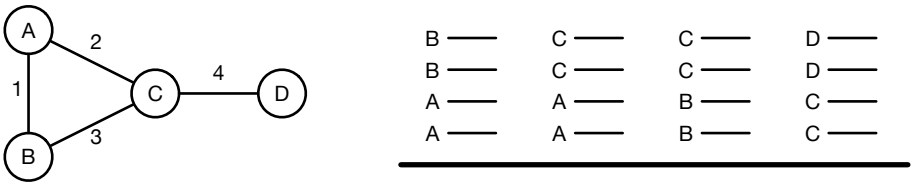

Figure 2.1: An example of the reduction from an instance of the MAXIMUM INDEPENDENT SET problem to a collection of intervals using $t = 2$. It is straightforward to verify a subset of nodes is independent if and only if each point on the underlying line is spanned by at most t intervals in the union of their bundles.

cerned with how many clients can have their bundles completed within t steps. This would lead to an $O(1)$-approximation for **MSCB**.

Unfortunately, even in the special case where G is an interval graph, this seems impossible to approximate well.

Lemma 1. *Let $G = (V, E)$ be an interval graph and $V_1, V_2, \ldots, V_p$ a partition of V. For any $t \leq |V|$ and any constant $\delta > 0$, there is no $O(|V|^{1/3-\delta})$-approximation algorithm for computing the maximum size $\mathcal{P} \subseteq \{1, 2, \ldots, p\}$ such that $\cup_{j \in \mathcal{P}} V_j$ can be scheduled within t steps unless $\mathbf{P} = \mathbf{NP}$.*

Proof. We reduce from the MAXIMUM INDEPENDENT SET problem. Let $H = (U, F)$ be an undirected graph. Order F arbitrarily as $e_1, e_2, \ldots, e_{|F|}$. We build an interval graph over the line $[1, 2 \cdot |F|]$. Namely, for each $v \in U$ and each e_i having v as an endpoint, add t copies of the interval $[2 \cdot i - 1, 2 \cdot i]$. Let U_v denote all intervals created from $v \in U$ this way. Note, there are exactly $2 \cdot t$ intervals of the form $[2 \cdot i - 1, 2 \cdot i]$ for each $1 \leq i \leq |F|$, one for each endpoint of e_i. Let G be the resulting interval graph whose nodes/intervals are partitioned as $\{U_v : v \in U\}$. Figure 2.1 illustrates this reduction.

Let $\mathcal{I}$ be a subset of bundles. It is straightforward to verify the intervals in $\cup_{U_v \in \mathcal{I}} U_v$ can be t-colored (i.e. can be scheduled within t time steps) if and only if $\{v \in U : U_v \in \mathcal{I}\}$ is an independent set in H.

Finally, recall for any constant $\delta > 0$, there is no $|U|^{1-\delta}$-approximation for the maximum independent set problem unless $\mathbf{P} = \mathbf{NP}$ [31]. Since $t \leq n$, then G has $2 \cdot |F| \cdot t \leq O(|U|^3)$ vertices, as required. Thus, an $\alpha = o(|V|^{1/3-\delta/3})$-

approximation for the largest number of parts that can be scheduled in t time steps would yield a $o(|U|^{1-\delta})$-approximation for the maximum independent set problem in H. Replacing δ by 3δ (again a constant) yields the result. □

Since **MSC** techniques seem to fail for **MSCB**, one could look to techniques successfully used for approximating Coflow Scheduling. Recall Coflow Scheduling is the special case of **MSCB** when the graph G is the line graph $L(H)$ of an undirected graph. A property of Coflow Scheduling that is commonly leveraged to design approximation algorithms is that for each edge e in H (i.e. a job), all other edges that conflict with e share one of two endpoints with e. Interval graphs, however, do not have this property.

For example, the algorithm in [1] solves a time-indexed LP relaxation and uses the familiar trick of greedy scheduling jobs according to their fractional completion time. Their analysis relies on the fact that there are only 2 "reasons" an edge may not be scheduled at a time step (i.e. one of the two endpoints already has an edge at that time step). Also, in [11] a hypergraph matching result by Haxell [20] is used to demonstrate that a good schedule of jobs exists. However, the way this matching result is used in [11] crucially relies on the fact that the formed hyperedges only have 3 nodes, which comes from the fact that each edge of H has only two endpoints.

2.2 Algorithm

Despite these challenges, we are still able to design a constant-factor approximation for **MSCB** in perfect graphs. Our techniques can be seen as a workaround to the problem highlighted in Lemma 1, though we also need to use LPs to do so. That is, we use an LP-relaxation to fractionally schedule the jobs. For a geometrically-increasing sequence of values t, we consider the jobs that are completed to an extent of at least $1/2$ by time t in the fractional solution. The LP constraints will witness that the size of the largest clique among these jobs is $O(t)$. The fact the graph is perfect then allows us to color these jobs with $O(t)$ colors, i.e. to schedule them all within $O(t)$ time steps.

For simplicity of presentation, we suppose $G = (V, E)$ is an interval graph

with $n = |V|$ nodes and that $V = \{V_1, V_2, \ldots, V_p\}$. This allows us to write a polynomial-size LP relaxation. The straightforward extension to perfect graphs (albeit with an exponential-size LP) will be discussed at the end of this chapter. Thus, we suppose each vertex $v \in V$ is associated with an interval $[s_v, t_v] \subseteq \mathbb{R}$. It is a folklore result that we may further assume, without loss of generality, that each endpoint of each interval lies in the set $\{1, 2, \ldots, 2 \cdot n\}$. For each $1 \leq i \leq 2 \cdot n$ the set $C_i := \{v \in V : i \in [s_v, t_v]\}$ is easily seen to be a clique in G. It is also known that every clique of G is a subset of C_i for some $1 \leq i \leq 2 \cdot n$ [14].

Like some previous work in **MSC** [7] and Coflow Scheduling [1], [11], we consider a time-indexed LP relaxation for **MSCB**. For each $v \in V$ and $1 \leq t \leq n$ we let $x_{v,t}$ be a variable indicating v should be colored t and for each $1 \leq k \leq p$ we let f_k be a variable that is intended to be the largest color used to color nodes in V_k (i.e. when all jobs for bundle k are completed). Throughout this section, we adhere to the following indexing conventions: k will be used for bundles, t for time steps/colors, i for points on the underlying line $\{1, 2, \ldots, 2 \cdot n\}$ in the interval graph, and j for indexing geometric groupings of jobs defined in the algorithm.

$$
\begin{array}{rrcll}
\textbf{minimize}: & \sum_{k=1}^{p} w_k \cdot f_k & & & \\
\textbf{subject to}: & f_k & \geq & \sum_{t=1}^{n} t \cdot x_{v,t} & \forall\ 1 \leq k \leq p, v \in V_k \\
& \sum_{v \in C_i} x_{v,t} & \leq & 1 & \forall\ 1 \leq t \leq n, \forall\ 1 \leq i \leq 2 \cdot n \\
& \sum_{t=1}^{n} x_{v,t} & = & 1 & \forall\ v \in V \\
& x, f & \geq & 0 &
\end{array}
$$

(**LP-MSCB**)

The first constraint says that bundle k is considered finished only after each $v \in V_k$ is completed, and the second constraint ensures that at most one vertex from any clique in the interval graph can be processed at a time. The third ensures each job is completed at some point. Clearly, the optimum value of (**LP-MSCB**) is at most the optimum value of the **MSCB** instance since the natural integer solution corresponding to the optimal **MSCB** solution is feasible for this LP.

2.3 Rounding Algorithm

Leveraging the solution from the LP relaxation (**LP-MSCB**) as a heuristic offers valuable insight. However, the relaxed LP solution does not directly indicate the optimal processing order for tasks. To address this, we introduce a geometric grouping inspired by the concept of maximum k-colorable subgraphs discussed earlier in this chapter. This approach ensures that for any given time t, all tasks that are "at least half done" according to the LP solution can be completed within the next $2 \cdot t$ time steps.

After computing an optimal solution to **LP-MSCB**, for each job $v \in V$ we let τ_v denote the smallest time t such that $\sum_{t' \leq t} x_{v,t'} \geq 1/2$, i.e. when the LP solution has completed v to an extent of at least $1/2$. Notice by minimality of τ_v we also have $\sum_{t' \geq t} x_{v,t'} = 1 - \sum_{t' < t} x_{v,t'} > 1 - 1/2 = 1/2$.

For a bundle k, we then let $\hat{f}_k := \max_{v \in V_k} \tau_v$ be the minimum time when all jobs in V_k are completed to an extent of at least $1/2$ by the LP solution. As we show below, it is not hard to see $\sum_k w_k \cdot \hat{f}_k$ is within a constant factor of the optimal LP solution.

We then employ geometric grouping of the jobs $v \in V$. That is, for each time t in a geometric sequence we form a group with all jobs v having $\tau_v \leq t$. Using properties of the LP solution and interval graphs, we show we can properly color all jobs in each such group with $2 \cdot t$ colors. Concatenating these schedules for the various groups in this geometric sequence completes the algorithm.

To optimize our final ratio, we carefully choose the geometric growth rate and also pick an initial random geometric offset, as has been done in many previous works in minimum-latency problems, e.g. [17]. We could also try a different parameter than $1/2$ as the choice of threshold for the defining values τ_v, but it turns out that $1/2$ is the optimal value for our approach. Also, readers with experience in **MSC** algorithms may wonder about another optimization. Namely, with **MSC** once one has colored a geometric group one may get a slight improvement in the approximation guarantee by optimally ordering the colors so that larger color classes are finished earlier. However,

this optimization does not work in our setting since we are concerned with the completion times of bundles and reordering color classes within a group's coloring may not affect the completion time of a bundle.

We let $q > 1$ be a constant. It turns out the optimal setting for q in our algorithm is just e, the base of the natural logarithm. We leave q as an unspecified constant for now and only set it at the point in the analysis where it is apparent that this was the best choice of constant. The precise description of our rounding technique is presented in Algorithm 1.

Algorithm 1: MSCB Scheduling

Compute an optimal solution (x, f) to (**LP-MSCB**).
Set τ_v to the smallest integer such that $\sum_{t \leq \tau_v} x_{v,t} \geq 1/2$ for each $v \in V$.
Sample $\alpha \sim [0, 1)$ uniformly.
Let $U_j = \{v \in V : \lfloor q^{j-1+\alpha} \rfloor < \tau_v \leq \lfloor q^{j+\alpha} \rfloor\}$ for $0 \leq j \leq \log_q n$.
for each U_j in increasing order of j **do**
  Schedule all jobs in U_j within the next $2 \cdot \lfloor q^{j+\alpha} \rfloor$ unused time steps. {See Claim 1}

2.4 Analysis

Recall the sets U_j described in Algorithm 1.

Claim 1. *For each j, the jobs in U_j can be scheduled without any conflicts using at most $2 \cdot q^{j+\alpha}$ time steps.*

Proof. We claim the size of the largest clique contained in U_j is at most $2 \cdot q^{j+\alpha}$. If so, then we can properly color all of U_j using at most $2 \cdot q^{j+\alpha}$ colors because interval graphs are perfect.

First, consider any $v \in U_j$ and say $v \in V_k$. Then $\tau(v) \leq \hat{f}_k \leq q^{j+\alpha}$, so

$$\sum_{t \leq q^{j+\alpha}} x_{v,t} \geq \sum_{t \leq \tau(v)} x_{v,t} \geq \frac{1}{2}.$$

Now consider any point $i \in \{1, 2, \ldots, 2 \cdot n\}$ on the interval, our goal is to show $|U_j \cap C_i| \leq 2 \cdot q^{j+\alpha}$. Letting $X_{i,j} := \sum_{v \in U_j \cap C_i} \sum_{t \leq q^{j+\alpha}} x_{v,t}$, summing the above bound over $v \in |U_j \cap C_i|$ shows $X_{i,j} \geq |U_j \cap C_i|/2$.

On the other hand, by the LP constraints we also have

$$X_{i,j} = \sum_{t \le q^{j+\alpha}} \sum_{v \in U_j \cap C_i} x_{v,t} \le \sum_{t \le q^{j+\alpha}} \sum_{v \in C_i} x_{v,t} \le \sum_{t \le q^{j+\alpha}} 1 = q^{j+\alpha}.$$

From these two bounds on $X_{i,j}$ we have $|U_j \cap C_i|/2 \le X_{i,j} \le q^{j+\alpha}$ so $|U_j \cap C_i| \le 2 \cdot q^{j+\alpha}$.

Finally, consider any clique $C \subseteq U_j$. Any clique of G is contained in a clique of the form C_i so $C \subseteq U_j \cap C_i$. Thus, we have $|C| \le |U_j \cap C_i| \le 2 \cdot q^{j+\alpha}$, that is the bound holds for all cliques $C \subseteq U_j$. □

Next, we show each bundle k finishes within time $O(\hat{f}_k)$. Fix any such bundle k and pick any $v_k \in V_k$ with $\tau_{v_k} = \hat{f}_k$. For any value α sampled by the algorithm, the completion time of bundle k is upper bounded by the completion time of all jobs in the bundle U_j that contains v_k. This is because no job of V_k will be placed in a bundle $U_{j'}$ having $j' > j$ and because we concatenated the schedules for the various buckets in increasing order of j.

Since $0 \le \alpha \le 1$ then there is some integer j_k such that $v_k \in U_{j_k - 1}$ or $k \in U_{j_k}$, depending on the value of α. The breaking point between these two events occurs at $\alpha = \log_q \hat{f}_k - (j_k - 1)$. Letting T_α be the quantity $2 \cdot q^{j+\alpha}$ for the group $j \in \{j_k - 1, j_k\}$ that v_k is assigned to for a given α, we have:

$$T_\alpha \le \begin{cases} 2 \cdot q^{j_k - 1 + \alpha} & k \in U_{j_k - 1} \\ 2 \cdot q^{j_k + \alpha} & k \in U_{j_k} \end{cases}$$

Since we concatenate the schedules for the groups U_j in increasing order of index j and each group U_j is completed by time $2 \cdot q^{j+\alpha}$ (Claim 1), then for any j each job in U_j will be completed by time $\sum_{j'=1}^{j} 2 \cdot q^{j'+\alpha} \le \frac{2 \cdot q}{q-1} \cdot q^{j+\alpha}$. Therefore we have

$$\begin{aligned}
\mathbf{E}_{\alpha \sim [0,1)}[T_\alpha] &= \int_0^1 T_\alpha \, d\alpha \\
&= \frac{2 \cdot q}{q-1} \cdot \left(\int_0^{\log_q \hat{f}_k - (j_k - 1)} q^{j_k + \alpha} \, d\alpha + \int_{\log_q \hat{f}_k - (j_k - 1)}^1 q^{j_k - 1 + \alpha} \, d\alpha \right) \\
&= \frac{2 \cdot q}{q-1} \cdot \left(\frac{q^{j_k + \alpha}}{\ln q} \Big|_0^{\log_q \hat{f}_k - (j_k - 1)} + \frac{q^{j_k - 1 + \alpha}}{\ln q} \Big|_{\log_q \hat{f}_k - (j_k - 1)}^1 \right) \\
&= \frac{2 \cdot q}{\ln q} \cdot \tau_{v_k} = \frac{2 \cdot q}{\ln q} \cdot \hat{f}_k
\end{aligned}$$

At this point, we see the optimal choice of q is $e \approx 2.717$, the base of the natural logarithm. Setting q to e yields

$$\mathbf{E}_{\alpha\sim[0,1)}[T_\alpha] = \frac{2\cdot q}{\ln q}\cdot \hat{f}_k = 2\cdot e\cdot \hat{f}_k$$

We complete the proof by bounding f_k by $O(\hat{f}_k)$. Recalling $\sum_{t\geq\tau_{v_k}} x_{v_k,t} \geq 1/2$, we see

$$f_k \geq \sum_t t\cdot x_{v_k,t} \geq \sum_{t\geq\tau_{v_k}} t\cdot x_{v_k,t} \geq \tau_v \cdot \sum_{t\geq\tau_{v_k}} x_{v_k,t} \geq \frac{\tau_{v_k}}{2} = \frac{\hat{f}_k}{2}$$

To put this all together, by Claim 1 the completion time of a bundle k is at most T_α. In expectation over the random choice of α, this is at most $2\cdot e\cdot \hat{f}_k$. Finally, from the bound directly above we see the expected completion time of a bundle is then at most $4\cdot e\cdot f_k$. Thus, the expected total completion time of all bundles is at most $4\cdot e \leq 10.874$ times the optimum value of (**LP-MSCB**).

2.5 Extensions

Perfect Graphs

The only change to the LP is that the second collection of constraints is replaced by the following more general constraints:

$$\sum_{v\in C} x_{v,t} \leq 1 \quad \forall\, t, \forall \text{ cliques } C \text{ of } G \tag{2.1}$$

In general, there are exponentially many cliques (and even exponentially-many maximal cliques) in a perfect graph. Still, these constraints can be separated in polynomial time for perfect graphs (Theorem 67.6 in [29]) meaning the LP can still be solved optimally in polynomial time using the Ellipsoid method [15].

The rest of the proof carries through essentially without modification: the size of a maximum clique in U_j is still bounded to be at most $2\cdot q^{j+\alpha}$. That is, let $C\subseteq U_j$ be a clique. Since each $v\in U_j$ has $\tau_v \leq \lfloor q^{j+\alpha}\rfloor$ then

$$\sum_{v\in C}\sum_{t\leq q^{j+\alpha}} x_{v,t} \geq \sum_{v\in C}\frac{1}{2} = |C|/2.$$

On the other hand, by the more general clique constraints (2.1) we have

$$\sum_{t \leq q^{j+\alpha}} \sum_{v \in C} x_{v,t} \leq \sum_{t \leq q^{j+\alpha}} 1 \leq q^{j+\alpha}.$$

Since G is perfect, then by definition U_j can be colored using at most $2 \cdot q^{j+\alpha}$ colors and such a coloring can be done in polynomial time (Corollary 67.2c[29]). The rest of the analysis is unchanged, thus the full form of Theorem 1 is proven.

Derandomizing

It is simple to efficiently derandomize our approach. We simply list all break points α of the form $\log_q \hat{f}_k - (j_k - 1)$ over all bundles k and try all α between these break points. Our algorithm is deterministic once α is given and these break points are the only values of α where the behavior of the algorithm changes. Taking the best solution found over all such α is surely no worse than the expected cost of the returned solution when choosing α randomly

Chromatic Number Upper-bounded by β times the maximum clique size

As established in Corollary 1, specific graph classes exhibit a chromatic number upper-bounded by a factor of β times the maximum clique size. Recalling our prior demonstration in Claim 1, the maximum clique size within a graph U_j is bounded to at most $2 \cdot q^{j+\alpha}$. Perfect graphs necessitate this exact number of colors for coloring U_j. In contrast, in these specific graph classes, at most β times more colors are needed for U_j. Thus, a maximum of $2 \cdot \beta \cdot q^{j+\alpha}$ colors suffice. The rest of the analysis is similar to 2.4. Consequently, the approximation factor increases by a factor of β to become $\beta \cdot 10.874$.

Extensions to Other Graph Classes

For Corollary 1, the assumptions mean we can approximately separate the clique constraints $\sum_{v \in C} x_{v,t} \leq 1$ in polynomial time ultimately leading to an efficient algorithm that finds an LP solution with cost at most OPT where all constraints hold except perhaps these new clique constraints. Instead, we

would have $\sum_{v \in C} x_{v,t} \leq c$ where c is the approximation factor of computing a maximum-weight clique in G.

Using the approximate separation oracle for the maximum-weight clique problem, we can obtain a solution to the relaxed linear program (LP) with a cost that does not exceed OPT, while only approximately satisfying the clique constraints. This use of the approximate separation oracle has been extensively demonstrated in the literature. For example, refer to Lemma 30 in [13].

The approximate relationship between maximum cliques and the chromatic number of graphs satisfying the assumptions of Corollary 1 allow us to conclude U_j can be colored with at most $c' \cdot q^{j+\alpha}$ colors where c' is also a constant. Carrying this term through the rest of the analysis shows the algorithm is an $O(1)$-approximation.

Release Time

Many real-world job scheduling scenarios involve jobs that have dependencies or are not available for processing until a specific time. This section addresses how our algorithm incorporates release times for jobs.

Let r_v be the release time of job $v \in V$. With this new property, we add a new constraint to our **LP-MSCB**. This constraint makes sure that no job can be processed before its release time.

$$x_{v,t} \quad = \quad 0 \quad \forall\, v \in V, 0 \leq t < r$$

In Algorithm 1, each job v was assigned to U_j if $\lfloor q^{j-1+\alpha} \rfloor < \tau_v \leq \lfloor q^{j+\alpha} \rfloor$, This approach could potentially schedule a job before its release. This can occur when a job v has $e^{j-1+\alpha} < r_v$ and $\tau_v \leq e^{j+\alpha}$. The worst case scenario happens when $r_v = e^{j+\alpha}$ and the algorithm schedules it at time step $2 \cdot e^{j-1+\alpha} + 1$ (remembering the maximum clique size is bounded by $2 \cdot e^{j+\alpha}$). Then, we have a problem when

$$2 \cdot e^{j-1+\alpha} + 1 < e^{j+\alpha}$$

Solving this inequality reveals that only when $j \leq 1.331$ can a job potentially be scheduled before its release time. Since j must be an integer, the only problematic case is $j = 1$.

Therefore, we propose a simple modification: we simply ignore $j = 1$ and schedule all jobs from $j \geq 2$. This exclusion is negligible. If the algorithm could schedule a set of bundles with small j values, then ignoring only $j = 1$ will not cause the approximation factor to exceed 10.874. In other cases, the impact on the factor will be negligible.

Chapter 3

MSCB with Task Concurrencies

Recall in **MSCB-TC**, the bundles form a *partition* $V_1, \ldots, V_p$ of P and for each bundle k we have a bound d_k on the number of jobs in V_k that can be scheduled at any single time. This models settings where clients can only deliver/retrieve a bounded number of their jobs at any single time. Also, recall that we assume G is a chordal graph.

The new algorithm starts with (**LP-MSCB**) except the cliques C_i used to define the constraints are the polynomially-many maximal cliques of G [14] (which can be enumerated in polynomial time) and two additional classes of constraints are added. First, for any bundle $1 \leq k \leq p$ and any time t we add the constraints

$$\sum_{v \in V_k} x_{v,t} \leq d_k.$$

That is, at any given time a maximum of d_k jobs for bundle k can be processed. We call these *concurrency constraints*. Next, For any bundle $1 \leq k \leq p$ we add the constraints

$$f_k \geq \lceil |V_k|/d_k \rceil$$

which enforces the trivial lower bound that $\lceil |V_k|/d_k \rceil$ time steps are required to finish bundle k even if we processed d_k of its jobs per step. Note, without this bound the LP could cheat in the following way: if $d_k = 1$ and $V_k = \{v_1, \ldots, v_m\}$ we could set $x_{v_i,t} = 1/m$ for all $1 \leq i \leq m$ and $1 \leq t \leq m$ which would permit us to set $f_k = (m+1)/2$ whereas an integer solution would clearly require $f_k \geq m$.

For the rest of this section, by "schedule" we mean a proper coloring of G with the additional constraint that for any bundle k and any time t we have at most d_k jobs in V_k being colored with t.

We need to make some minor modifications to the algorithm. First, we now define $\hat{f}_k := \max\{\lceil |V_k|/d_k \rceil, \max_{v \in V_k} \tau_v\}$. Next, we change U_j to be

$$U_j = \{k : \lfloor q^{j-1+\alpha} \rfloor < \hat{f}_k \leq \lfloor q^{j+\alpha} \rfloor\}.$$

Finally, when we color U_j, we will ensure that the new concurrency constraints are satisfied with this coloring. The following structural result enables us to do this while limiting the loss in the final approximation guarantee. Here, we are letting $\chi(G)$ denote the chromatic number of G.

Lemma 2. *Let $G = (V, E)$ be a chordal graph whose vertices are partitioned as $V_1, \ldots, V_p$. Further, for each $1 \leq k \leq p$ let $d_k \geq 1$ be an integer. In polynomial time, we can compute a proper coloring of G using at most $\chi(G) + \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil - 1$ colors such that for each $1 \leq k \leq p$, no color appears more than d_k times among nodes in V_k.*

Proof. Recall that a graph is a chordal graph if and only if it has a perfect elimination ordering, i.e. an ordering $v_1, v_2, \ldots, v_n$ such that for each $1 \leq j \leq n$, the *left-neighborhood* $N_\ell(v_j) = \{i < j : v_i v_j \in E\}$ of each node is a clique and that this ordering can be computed in linear time [14].

To compute the coloring we need, process the nodes v_i in this order. When coloring v_i, we simply avoid using a color already assigned to a node in $N_\ell(v_i)$ or already assigned to d_k nodes in the same part V_k as v_i. This can be done if we allow $\chi(G) + \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil - 1$ colors. □

We briefly remark that Lemma 2 is tight even for interval graphs where $d_k = 1$ for all V_k. Consider the case where V_1 consists of m jobs whose corresponding intervals are $[1, 2], [3, 4], [5, 6], \ldots, [2m - 1, 2m]$ and $V_2, \ldots, V_p$ each consists of a single job whose corresponding interval is $[1, 2m]$. Figure 3.1 depicts this interval graph. The chromatic number is exactly p but no two jobs can receive the same color since the only non-intersecting pairs of intervals have their

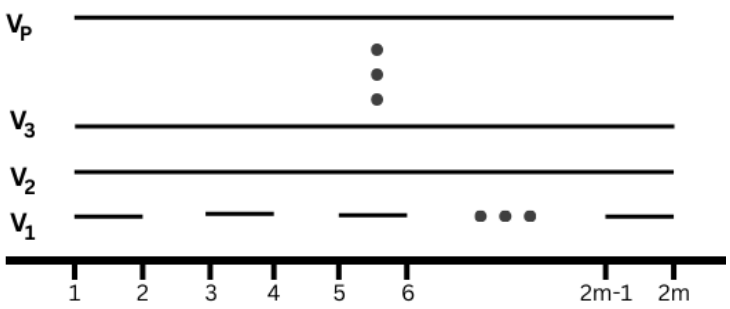

Figure 3.1: Tightness of Lemma 2 for Interval Graphs.

corresponding jobs in the same bundle V_1. Therefore, $|V_1| + |V_2| + \ldots + |V_p| = p + m - 1$ colors are required.

Towards coloring U_j, we define $\mathcal{V}_j = \{k : V_k \cap U_j\} \neq \emptyset$ to be all bundles having some job in U_j and then we let $S_j = \max_{k \in \mathcal{V}_j} \lceil |V_k \cap U_j|/d_k \rceil$. Since $\lceil |V_k \cap U_j|/d_k \rceil$ is a lower bound on the time required to finish all jobs $V_k \cap U_j$ due to the task concurrency constraint for bundle k, we have that S_j is another lower bound for the time needed to complete all jobs in the set U_j. The new LP constraints help assert this lower bound as well.

Lemma 3. *For each group j, $S_j \leq q^{j+\alpha}$.*

Proof. This is demonstrated by leveraging the additional constraint incorporated into our LP. For every $k \in \mathcal{V}_j$, we know that $\frac{|V_k|}{d_k} \leq \hat{f}_k$. Furthermore, based on the new definition of U_j it is clear that $\hat{f}_k \leq q^{j+\alpha}$. Consequently, $\frac{|V_k|}{d_k}$ is less than equal to $q^{j+\alpha}$, implying $|V_k \cap U_j| \leq q^{j+\alpha} \cdot d_k$. Therefore, it follows that $S_j \leq q^{j+\alpha}$.

□

As with the **MSCB** approximation, the maximum clique size in U_j is at most $2 \cdot q^{j+\alpha}$. Further, we have just shown $|V_k \cap U_j| \leq q^{j+\alpha} \cdot d_k$ for any $k \in \mathcal{V}_j$. So Lemma 2 means there is a proper coloring of U_j using at most $3 \cdot q^{j+\alpha}$ colors such that no bundle in $\mathcal{V}_k$ has more than d_k jobs colored with the same color.

The rest of the analysis is similar to the analysis of the algorithm for **MSCB** except the approximation ratio has changed since we used $3 \cdot q^{\alpha+j}$

colors instead of $2 \cdot q^{\alpha+j}$ colors to color each U_j. Thus, it is a $6 \cdot e \leq 16.31$-approximation.

Corollary 2 essentially follows by how Corollary 1 followed from Theorem 1 but using a more general form of Lemma 2. Namely, if there is an ordering of the nodes $v_1, v_2, \ldots, v_n$ such that the left-neighborhood $N_\ell(v_i) = \{v_j : v_i v_j \in E, j < i\}$ of any node v_i can be covered with $R = O(1)$ cliques then we can find a proper coloring of G using at most $R \cdot \chi(G) + \max_k \lfloor |V_k|/d_k \rfloor$ colors by picking the lowest available color not appearing in the left-neighborhood of v_i that is also not used d_k times in the part V_k with v_i.

3.1 Extensions

Chromatic Number Upper-bounded by β times the maximum clique size

In Section 2.5, we demonstrated the applicability of our **MSCB** algorithm to graphs where the chromatic number is bounded by the maximum clique size times a constant β. Here, we extend this analysis to explore the **MSCB-TC** algorithm's performance for this specific graph class.

Corollary 2 essentially follows by how Corollary 1 followed from Theorem 1 but using a more general form of Lemma 2. Recall from Section 3 that the **MSCB-TC** algorithm requires a perfect elimination ordering for vertex coloring. We employ the same strategy for these graphs, but with a new ordering criterion. Specifically, if an ordering $v_1, v_2, \ldots, v_n$ exists such that for any vertex v_i, its left neighborhood $N_\ell(v_i) = \{v_j : v_i v_j \in E, j < i\}$ can be covered by at most β cliques, then the **MSCB-TC** algorithm is applicable. Similar to chordal graphs, when coloring vertex v_i we avoid colors assigned to vertices in $N_\ell(v_i)$ or to d_k vertices already in the same part V_k as v_i. This is achievable if we allow a maximum of $\beta \cdot \chi(G) + \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil - 1$ colors.

The analysis from Section 3 is directly applicable here. We can establish that $\beta \cdot \chi(G)$ is upper bounded by $\beta \cdot 2 \cdot q^{j+\alpha}$ and $\left\lceil \frac{|V_k|}{d_k} \right\rceil$ is upper bounded by $q^{j+\alpha}$. Consequently, coloring any set U_j requires at most $(2 \cdot \beta + 1) \cdot q^{j+\alpha}$ colors.

This translates to a constant-factor approximation ratio of $(4 \cdot \beta + 2) \cdot e$.

Release Time with Task Concurrency Limit

Building on the concept of release times introduced in Section 2.5, this section explores how it interacts with the task concurrency limit.

We incorporate the same release time constraint from Section 2.5 into the analysis here. However, one key difference exists: bundles here are partitions. This means all jobs within a bundle are assigned to U_j if $\lfloor q^{j-1+\alpha} \rfloor < \hat{f}_k \leq \lfloor q^{j+\alpha} \rfloor$.

As in the previous section, the potential issue arises when a job v in V_k has a release time r_v greater than $e^{j-1+\alpha}$ and its $\hat{f}_k \leq e^{j+\alpha}$. The remaining analysis for this scenario remains identical to Section 2.5.

3.2 MSCB-TC in Perfect Graphs - A Barrier

Lemma 2 fails to hold in perfect graphs even within any constant factor. That is, it may require $\Theta(\sqrt{n}) \cdot \max\{\chi(G), \max_k |V_k|/d_k\}$ colors to even if $d_k = 1$ for all k. Consider the following simple example on $n = N^2$ nodes for some integer N. The graph $G_N = (V, E)$ is partitioned into sets $V_1, \ldots, V_N$ and each V_k has exactly N nodes. We have an edge between any pair of nodes in different parts, but each part is an independent set.

It is easy to see such graphs are perfect. More generally, a graph that is partitioned into b nonempty independent sets and has all possible edges between these parts has chromatic number b and maximum clique size b (pick one node from each part). Since any induced subgraph of our graph G_N is of this form, then G_N is also perfect.

But any coloring satisfying task concurrency limits of $d_k = 1$ for all parts must in fact use n colors. Two nodes in different parts cannot receive the same color because they are connected by an edge and two nodes in the same part cannot receive the same color because the task concurrency limit is 1. Yet, $\chi(G) = N = \sqrt{n}$ and the maximum size of a part is also $N = \sqrt{n}$.

Still, this is the worst case. The following variation of Lemma 2 leads to

an $O(\sqrt{n})$-approximation for **MSCB-TC** in perfect graphs.

Lemma 4. *Let $G = (V, E)$ be a graph whose vertices are partitioned as $V_1, \ldots, V_p$. Further, for each $1 \leq k \leq p$ let $d_k \geq 1$ be an integer. There is a proper coloring of G using at most $\sqrt{n} \cdot \max\left\{\chi(G), \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil\right\}$ colors such that for each $1 \leq k \leq p$, no color appears more than d_k times among nodes in V_k. Such a coloring can be computed in polynomial time if G can be optimally colored in polynomial time.*

Proof. If $\max_k \lceil |V_k|/d_k \rceil \geq \sqrt{n}$, then the trivial n-coloring (i.e. all nodes get different colors) suffices. Otherwise, consider a proper coloring $\sigma : V \to \{1, 2, \ldots, \chi(G)\}$ of G. Order the nodes $v_1^k, v_2^k, \ldots, v_{|V_k|}^k$ arbitrarily in each part V_k.

Recolor a vertex v_i^k with the pair $(\chi(v_i^k), \lfloor i/d_k \rfloor)$. Clearly, this is a proper coloring since the first components of the new colors of nodes will differ on any edge of G. Further, at most d_k nodes in V_k will have the same second part of the pair describing their color. The number of colors used is $\chi(G) \cdot \max_k \lceil |V_k|/d_k \rceil \leq \chi(G) \cdot \sqrt{n}$, as required.

Finally, this can be done in polynomial time if we can compute a coloring of G with $\chi(G)$ colors in polynomial time. □

Using this in place of Lemma 2 yields an $O(\sqrt{n})$-approximation for **MSCB-TC** in perfect graphs. This proves Theorem 3.

Chapter 4

Conclusion

We have given the first constant-factor approximations for **MSCB** in a large variety of graph classes including perfect graphs and unit-disc graphs. Our techniques extend to give the first constant-factor approximations for **MSCB-TC** in certain graphs like chordal graphs, interval graphs, and unit-disc graphs.

It would be interesting to see what other graph classes admit constant-factor approximations for **MSCB** and, perhaps, also for **MSCB-TC**. Perhaps it is possible to design a constant-factor approximation for **MSCB-TC** in perfect graphs. Another interesting direction would be to get a purely combinatorial constant-factor approximation for **MSCB** in certain graph classes, i.e. one that avoids solving a linear program. Such algorithms exist for **MSC** in many cases, e.g. [2], [17]. One barrier is that it seems hard to approximate the maximum number of bundles that can be completed in a given number of time steps even in simple graph classes like interval graphs (Lemma 1). Perhaps a bicriteria approximation could be designed to circumvent this hardness, it would immediately lead to an $O(1)$-approximation through standard minimum latency arguments.

References

[1] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang, "On scheduling coflows - (extended abstract)," in *Proceedings of 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2017, pp. 13–24.

[2] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, "On chromatic sums and distributed resource allocation," *Information and Computation*, vol. 140, no. 2, pp. 183–202, 1998.

[3] A. Bar-Noy and G. Kortsarz, "The minimum color-sum of bipartite graphs," *Journal of Algorithms*, vol. 28, no. 2, pp. 339–365, 1998.

[4] B. K. Bhattacharya and D. Kaller, "An $O(m+n\log n)$ algorithm for the maximum-clique problem in circular-arc graphs," *Journal of Algorithms*, vol. 25, no. 2, pp. 336–358, 1997.

[5] A. Borodin, I. Ivan, Y. Ye, and B. Zimny, "On sum coloring and sum multi-coloring for restricted families of graphs," *Theoretical Computer Science*, vol. 418, no. 2, pp. 183–202, 2012.

[6] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Mathematics*, vol. 86, no. 1, pp. 165–177, 1990. DOI: https://doi.org/10.1016/0012-365X(90)90358-O.

[7] I. DeHaan and Z. Friggstad, "Approximate minimum sum colorings and maximum k-colorable subgraphs of chordal graphs," in *Algorithms and Data Structures Symposium (WADS)*, 2023, pp. 326–339.

[8] I. DeHaan and Z. Friggstad, "Approximate minimum sum colorings and maximum k-colorable subgraphs of chordal graphs," *ASpringer, Cham*, vol. 14079, 2023.

[9] U. Feige, "Approximating maximum clique by removing subgraphs," *SIAM Journal on Discrete Mathematics*, vol. 18, no. 2, pp. 219–225, 2004.

[10] U. Feige and J. Kilian, "Zero knowledge and the chromatic number," *Journal of Computer and System Sciences*, vol. 57, no. 2, pp. 187–199, 1998.

[11] T. Fukunaga, "Integrality gap of time-indexed linear programming relaxation for coflow scheduling," in *In Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques(APPROX)*, vol. 245, 2022, 36:1–36:13.

[12] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai, "Improved bounds for sum multicoloring and scheduling dependent jobs with min-sum criteria," in *Approximation and Online Algorithms*, 2005, pp. 68–82.

[13] A. Ganesh, B. Maggs, and D. Panigrahi, "Universal algorithms for clustering problems," *ACM Transactions on Algorithms*, vol. 19, no. 15, pp. 1–46, 2023.

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

[15] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Springer Berlin, Heidelberg, 1993.

[16] M. M. Halldórsson and G. Kortsarz, "Algorithms for chromatic sums, multicoloring, and scheduling dependent jobs," in *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Metholologies and Traditional Applications*, Chapman and Hall/CRC, 2018, pp. 671–684.

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

[18] M. Halldórsson and G. Kortsarz, "Tools for multicoloring with applications to planar graphs and partial k-trees," *Journal of Algorithms*, vol. 42, no. 2, pp. 334–366, 2002.

[19] M. Halldórsson, G. Kortsarz, and M. Sviridenko, "Sum edge coloring of multigraphs via configuration lp," *ACM Trans. Algorithms*, vol. 7, no. 2, pp. 1–21, 2011.

[20] P. E. Haxell, "A condition for matchability in hypergraphs," *Graphs and Combinatorics*, vol. 11, pp. 245–248, 1995.

[21] S. Im, B. Moseley, K. Pruhs, and M. Purohit, "Matroid Coflow Scheduling," in *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, vol. 132, 2019, 145:1–145:13.

[22] K. Jansen, "The optimum cost chromatic partition problem," *in Proceedings of Third Italian Conference on Algorithms and Complexity (CIAC '97)*, vol. 40, pp. 25–36, 1997.

[23] E. Kubicka, "The chromatic sum and efficient tree algorithms," Ph.D. dissertation, Western Michigan University, 1989.

[24] M. Malafiejski, K. Giaro, R. Janczewski, and M. Kubale, "Sum coloring of bipartite graphs with bounded degree," *Algorithmica*, vol. 40, pp. 235–244, 2004.

[25] D. Marx, "A short proof of the np-completeness of minimum sum interval coloring," *Operations Research Letters*, vol. 33, no. 5, p. 382, 2005.

[26] J. Oxley, *Matroid Theory*. Oxford University Press, 2006.

[27] M. J. P. Peeters, "On coloring j-unit sphere graphs," Tilburg University, School of Economics and Management, Research Memorandum FEW 512, 1991.

[28] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015, pp. 294–303.

[29] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[30] M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of coflows in datacenters," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1674–1687, 2018.

[31] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," in *Proc. of ACM Symposium on Theory of Computing (STOC 2006)*, 2006, pp. 681–690.