Sum Coloring Interval and k-Claw Free Graphs with Application to Scheduling Dependent Jobs*

Magnús M. Halldórsson† Guy Kortsarz‡ Hadas Shachnai§

March 10, 2003

Abstract

We consider the sum coloring and sum multicoloring problems on several fundamental classes of graphs, including the classes of interval and k-claw free graphs. We give an algorithm that approximates sum coloring within a factor of 1.796, for any graph in which the maximum k-colorable subgraph problem is polynomially solvable. In particular, this improves on the previous best ratio known of 2 for interval graphs. We introduce a new measure of coloring, *robust throughput*, that indicates how 'quickly' the graph is colored, and show that our algorithm approximates this measure within a factor of 1.4575. In addition, we study the contiguous (or non-preemptive) sum multicoloring problem on k-claw free graphs. This models, for example, the scheduling of dependent jobs on multiple dedicated machines, where each job requires the exclusive use of at most k machines. Assuming that k is a fixed constant, we obtain the first constant factor approximation for the problem.

1 Introduction

We consider the problem of graph multicoloring, with the objective of minimizing the sum of highest colors. A multicoloring of a graph assigns to each vertex a collection of positive integers (colors) so that the sets of colors assigned to a given pair of adjacent vertices are disjoint. The sum multicoloring of an assignment is the sum over all vertices v of the highest color assigned to v. We need to find an assignment of colors to the vertices, such that sum

*A preliminary version of this paper appeared in *Proc. of the Fourth International Workshop on Approximation algorithms* (APPROX '01). LNCS #2129, Springer-Verlag, pp. 114–126.

†Department of Computer Science, University of Iceland, IS-107 Reykjavik, Iceland. E-mail: mmh@hi.is.

‡Department of Computer Science, Rutgers University, Camden, NJ. E-mail: guyk@crab.rutgers.edu.

§Contact author: Department of Computer Science, The Technion, Haifa 32000, Israel. E-mail: hadas@cs.technion.ac.il. Currently on leave at Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

multicoloring is minimized. This is known as the *sum multicoloring (*SMC*)* problem (see, e.g. [BHK$^+$00]). In the special case where each vertex is assigned a *single* color, our objective is to minimize the total sum of the colors assigned to the vertices, or the sum coloring of the assignment. This is known as the *sum coloring (*SC*)* (or *chromatic sum*) problem (see, e.g., [BK98, BHK99, KKK89, NSS99]).

The sum coloring and sum multicoloring problems often model scheduling jobs which are dependent because they utilize the same non-sharable resource. The exclusivity requirement can be captured by a graph of the pairwise conflicts; the vertices of the graph represent the jobs, and an edge in the graph between two vertices represents a dependency or conflict between the two corresponding jobs, which forbids scheduling these jobs at the same time. Assuming that jobs may have arbitrary integral execution times, a schedule of the jobs corresponds to a multicoloring of the graph, where each vertex is assigned as many colors as the processing time of the corresponding job. The large body of research on coloring various classes of graphs therefore translates to statements about the qualities of schedules. We elaborate on that in Section 1.1.

1.1 Definitions

Colorings Let $G = (V, E)$ be a undirected graph, possibly with vertex weights. We denote by n the number of vertices. A *coloring* of a graph is an assignment $\psi : V(G) \to \mathbb{N}$ of positive integers to the vertices such that adjacent vertices are assigned different colors. The vertices receiving the same color then form an *independent set* and are called a *color class*.

The *chromatic number* of G, denoted by $\chi(G)$, is the minimal number of colors required for coloring the vertices in G properly. The chromatic sum of ψ is the sum of the colors assigned to the vertices, or $\mathtt{SC}(G, \psi) = \sum_v \psi(v)$. The *sum coloring* problem is that of finding a coloring of a given graph G with the minimum chromatic sum $\mathtt{SC}(G)$. We also denote by $\mathtt{SC}(G, \mathsf{Alg})$ the chromatic sum of the coloring found by a given algorithm Alg.

In the maximum size k-colorable subgraph problem we seek a k-colorable subgraph of maximum cardinality, i.e. an induced subgraph whose chromatic number is at most k. The maximum size k-colorable subgraph problem is solvable, for example, on interval graphs, by a greedy algorithm [YG87], on chordal graph with constant maximum clique size (or k-trees), and on comparability graphs and their complements [F80] (see below the precise definitions of these classes). This holds also for the vertex-weighted problem, where we seek a k-colorable subgraph of maximum total weight. We denote by $\mathrm{kc}(G)$ the size of the maximum k-colorable subgraph in G.

The *throughput* of a partial coloring is the number of vertices colored. The maximum throughput possible with k colors is $\mathrm{kc}(G)$, the size of a maximum k-colorable subgraph. Given an algorithm that finds a coloring with color classes $I_1, I_2, \ldots$, we can ask, for any

given k, for the throughput of the first color classes, $Thr(\psi, k) = \sum_{i=1}^{k} |I_i|$, and compare that to the optimal throughput after k colors, $\mathrm{kc}(G)$. We say that the throughput of a coloring is robust, if it is good for each value of k simultaneously. The *robust throughput* (RT) measure of a coloring ψ compares the throughput to the optimal throughput for each color bound k, and takes the maximum. That is,

$$\mathtt{RT}(G, \psi) = \max_k \frac{\mathrm{kc}(G)}{Thr(\psi, k)}.$$

The robust throughput problem is to find a coloring that minimizes $\mathtt{RT}(G, \psi)$.

Multicolorings An instance of a multicoloring problem is a pair (G, x), where $G = (V, E)$ is a graph, and x is a vector of *color requirements* (or *lengths*) of the vertices. For a given instance, we denote by $p = \max_{v \in V} x(v)$ the maximum color requirement. A *multicoloring* of G is an assignment $\psi : V \to 2^{\mathcal{N}}$, such that each vertex $v \in V$ is assigned a set of $x(v)$ distinct colors, and adjacent vertices receive non-intersecting sets of colors.

Taking the cue from the relation to scheduling (see below), a multicoloring ψ is called *non-preemptive* if the colors assigned to each vertex are contiguous, i.e. if for any $v \in V$, $(\max_{i \in \psi(v)} i) - (\min_{i \in \psi(v)} i) + 1 = x(v)$. Otherwise, it is *preemptive*.

Denote by $f_\psi(v) = \max_{i \in \psi(v)} i$ the largest color assigned to v by a multicoloring ψ. The *sum multicoloring* (SMC) of ψ on G is

$$\mathtt{SMC}(G, \psi) = \sum_{v \in V} f_\psi(v) \ .$$

The SMC problem is to find a multicoloring ψ, such that $\mathtt{SMC}(G, \psi)$ is minimized; the non-preemptive version is npSMC. When all the color requirements are equal to 1, the problem reduces to the sum coloring (SC) problem.

Relationships of colorings and schedules As mentioned above, many scheduling problems can be represented as graph coloring problems. Traditionally, such problems arise in relation with timetabling. In general, they occur when jobs (represented by vertices in the graph) may conflict due to the fact that they need exclusive use of some resources. When the jobs are of different lengths (under some discrete measure), we obtain a multicoloring instance; when the lengths are identical, we obtain an ordinary graph coloring instance. When jobs must be run without interruption, we have a *non-preemptive* instance, and otherwise *preemptive*.

There are several measures that are considered for schedules. One is the *makespan*, or the time from the start to finish; this corresponds to the number of colors used in the (multi)coloring. Another measure is the *sum of completion times* of the jobs, or equivalently

the average completion time. This corresponds to, and is a primary motivation for, the sum (multi)coloring measure.

The robust throughput measure that we study here was previously studied in the area of combinatorial optimization (see, e.g., [HR02]). Generally, robust throughput examines the overall amount of "work" done until *any* given moment, throughout the execution of an algorithm. It may be viewed as a stronger version of the traditional throughput measure that is commonly studied in real-time scheduling (see, e.g. [BG$^+$01]).

Graph classes We define below the graph classes for which we derive our results.

Comparability graphs: An undirected graph is a comparability graph if it admits a transitive orientation, namely, it is possible to give a single direction to every edge $(a, b) \in E$ so that if (a, b) is directed from a to b and (b, c) is directed from b to c, then (a, c) (exists and) is directed from a to c. *Co-comparability graphs* are the complements of comparability graphs.

A special class of comparability graphs is the class of *permutation graphs*. A permutation graph is defined in terms of a permutation of the numbers $\{1, \ldots, n\}$. The matching diagram is obtained by writing the sequence of integers in a horizontal row and the sequence of its permuted values in another row below it, and by drawing n straight line segments between the two 1's, the two 2's, etc. Given some permutation σ, the graph $G(\sigma)$ corresponding to σ has vertices $\{1, \ldots, n\}$ and i, j are joined by an edge in $G(\sigma)$ if their lines intersect. A graph is called a permutation graph if there exists a permutation σ such that $G = G(\sigma)$. It is known that a graph G is a permutation graph if and only if G is both comparable and co-comparable (c.f. [G80].)

Interval graphs: A graph G is an interval graph if its vertices can be mapped to intervals on the real line so that two vertices are adjacent in G if and only if their corresponding intervals intersect. It is well known that interval graphs are co-comparability graphs (c.f. [F95, FMW97]). A more general class of intersection graphs of geometric objects are *trapezoid graphs*. They generalize to *m-trapezoid graphs* which form a proper hierarchy of co-comparability graphs, with $m = 0$ corresponding to interval graphs, $m = 1$ corresponding to trapezoid graphs, and $m = \infty$ corresponding to co-comparability graphs. Refer to [F95, FMW97] for definitions of these graph classes.

$(k+1)$-claw free graphs: A graph G is *$(k+1)$-claw free* if it does not contain the star $K_{1,k+1}$ as an induced subgraph. *Claw-free* graphs are those that are 3-claw free. Examples of $k+1$-claw free graphs include

- *Line graphs*: The line graph $L(G)$ is the edge-adjacency graph of a graph $G(V, E)$. Namely, the edges of G are the vertices of $L(G)$, and two vertices of $L(G)$ are joined in $E(L(G))$ if the two corresponding edges share a vertex in G. Any line graph is a claw-free graph.

- *Proper interval graphs*: These are intersection graphs of a family of intervals, so that no interval in the family is contained in another interval. Clearly, a proper interval graph is claw free.

- *Unit circle intersection graphs*: In these graphs the vertices correspond to unit circles in the plane and two vertices are joined by an edge if the two corresponding circles intersect. It is known that unit circle intersection graphs are 6-claw free (see, e.g., [MB$^+$95]).

The graph class that is most relevant for our applications is the class of intersection graphs of hypergraphs of sets of size at most k; such hypergraphs can be represented also as directed bipartite graphs of maximum out-degree k. Formally, consider a bipartite graph $B(V_1, V_2, E)$ so that all edges cross from V_1 to V_2. Let $deg(v)$ denote the number of neighbors of v and assume that $\max_{v \in V_1} deg(v) = k$. Form a new graph called the intersection graph G of B so that its vertices are the V_1 vertices, and two vertices $u, v \in V_1$ are adjacent in G if and only if they have a mutual neighbor in B. Observe that the neighborhood in G of every vertex v can be formed as a union of at most k cliques. Thus, the graph G is $(k+1)$-claw free.

This example is of particular importance as it can represent a scenario of dedicated tasks. The collection of jobs is represented by V_1; V_2 represents the processors and there is an edge from $v \in V_1$ to $w \in V_2$ if the job v requires the use of the processor w. The above $(k+1)$-claw free example occurs if every job requires the use of at most k processors.

Partial k-trees: A graph is a k-tree if it can be constructed incrementally, so that in each iteration a new vertex is added and made adjacent to a clique of size k already in the graph. A *partial k-tree* is a subgraph of a k-tree. Trees are partial 1-trees. Outerplanar graphs, namely, planar graphs that can be embedded in the plane so that all the vertices lie on the exterior face, are partial 2-trees. Series parallel graphs (see e.g., [BF96]) are partial 2-trees as well. Another example is *chordal graph*, which does not contain a cycle of length 4 (or more) as an induced subgraph. If, in addition to being chordal, the graph has maximum clique size k, then the resulting graph is a partial k-tree.

The above notions are closely related. For example, a *split graph* is a graph that can be decomposed into a clique and an independent set with arbitrary edges between them. It is easy to see that a split graph is chordal. A large group of split graphs are also comparability graphs (see [G80]).

1.2 Our results

The current paper has two parts. In the first part (Section 2), we discuss the sum coloring problem. Our main result is an approximation algorithm with a performance ratio of 1.796 for `SC`. The algorithm runs in polynomial time on graphs for which the maximum

induced k-colorable subgraph problem is solvable in polynomial time. This class includes comparability graphs and their complements (co-comparability graphs), which in turn include interval, trapezoid, and permutation graphs. They also include partial k-trees. Of particular interest is the class of interval graphs, for which we improve the best previously known approximation ratio of 2 [NSS99] for sum coloring. Our algorithm approximates also the *chromatic number* and the *robust throughput* of a given graph, the former within factor 2.718 and the latter within factor 1.4575.

In the second part of the paper (Section 3), we discuss the sum multicoloring problem. Our main result is a $2k(2k-1)$-approximation for $(k+1)$-claw free graphs, and in particular 12-approximation for line graphs. The special case of line graph has important applications, e.g, in biprocessor scheduling and data migration (see below). The previously best ratio known for this problem was $\log n$ [BHK$^+$00].[1]

Among other applications our approximation encompasses dedicated processor scheduling, where the number of processors required by any job is at most k. We elaborate on that in Section 1.4. The result is extended in Section 3.3 to yield the same approximation ratio for the problem of minimizing the weighted sum of completion times of dedicated tasks with release times, denoted in standard scheduling notation as $P|\text{fix}_j, r_j|\sum_j w_j C_j$. However, we first discuss the unweighted case without release times, so as to make the exposition simpler.

1.3 Applications of the sum coloring problem

We list below some applications that give rise to the sum coloring problem in (co-)comparability graphs and intervals graphs.

Train scheduling: Consider a railroad crossing, in which a set of n railroads, numbered by $1, \ldots, n$, intersect, and exchange their relative locations; that is, the i-th railroad becomes the $\sigma(i)$-th railroad after the intersection, where σ is a permutation of $(1, \ldots, n)$ (see in Figure 1). The goal is to schedule the arrivals of the trains to the intersection, such that two conflicting trains cross at distinct time intervals. When our objective is to minimize the average time that it takes for the trains to cross the intersection, we get an instance of the `SC` problem on permutation graph.

VLSI design: In the wire-minimization problem [NSS99], terminals lie on a single vertical line (each terminal is represented by an interval on this line), and with unit spacings are vertical bus lanes. Pairs of terminals are to be connected via horizontal wires on each side to a vertical lane, with non-overlapping pair utilizing the same lane. With the vertical segments fixed, the wire cost corresponds to the total length of horizontal segments. Numbering the

[1] Unless specified otherwise, all the logarithms in the paper are to the base of 2.

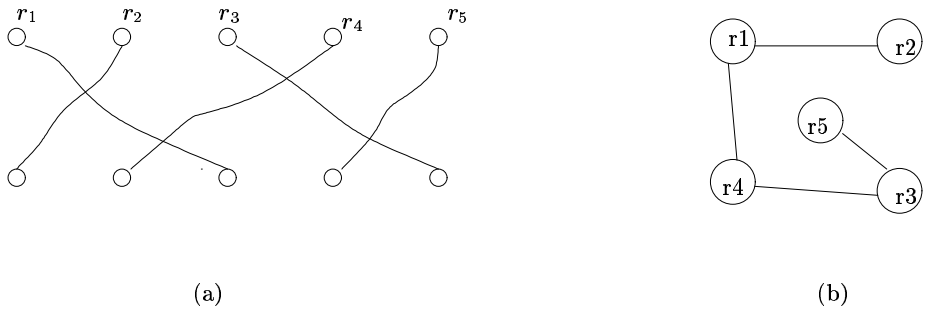


Figure 1: A train scheduling example: (a) a 5-railroad crossing; (b) The intersection graph of the railroad system.

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

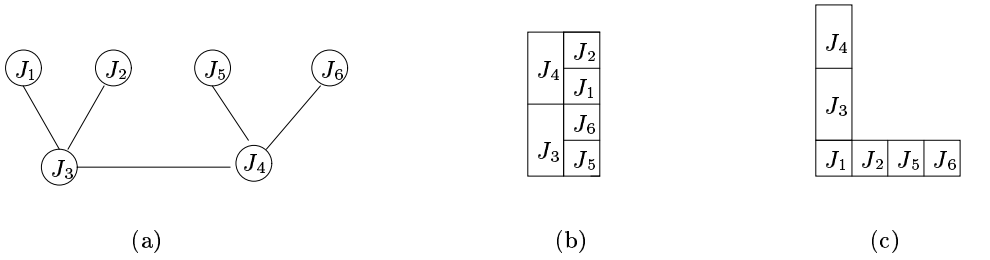
Storage allocation: Storage allocation in a warehouse involves minimizing the total distance traveled by a robot [W97]. Goods are checked in and out at known times; thus, goods that are not in the warehouse at the same time can share the same location. We represent each of the goods by an interval on the line, which gives the time interval in which it is available at the warehouse. Numbering the storage locations by their distance from the counter, the total distance corresponds to sum coloring the intervals formed by the goods.

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

1.4 Applications of sum multicoloring

Instances of sum multicoloring on $(k+1)$-free graphs are derived mainly from applications that involve *resource constrained scheduling*. Consider the following scenario that exemplifies the problem.

Example. Suppose that a set of jobs $J_1, \ldots, J_6$ is initiated in a distributed computing environment. Each of the jobs is available to run on a different processor, however, the jobs share accesses to a set of files $f_1, \ldots, f_5$, stored on the network file system. In particular, J_1, J_2, J_5 and J_6 require read/write operations in the files f_1, f_2, f_3 and f_4 respectively. J_3



(a) (b) (c)

Figure 2: Resource constrained scheduling (Example 1): (a) the conflict graph; (b) A schedule that minimizes overall completion time; (c) a schedule that minimizes sum of completion times

needs to copy data from f_1, f_2 to f_5; J_4 requires simultaneous reading of the contents of f_3, f_4 into the file f_5. The execution time of J_1, J_2, J_5, J_6 is one unit, while J_3, J_4 require 2 time units each. Note that each of the jobs requires an exclusive access to the corresponding file/s throughout its execution. While we can schedule the jobs such that the latest completion time is 4, the only schedule that minimizes average completion time (or, equivalently the sum of completion times) is the one that schedules the unit-length jobs first, using a total of 5 time units. This is illustrated in Figure 2.

Generally, in resource-constrained scheduling we are given a collection of n *jobs* of integral lengths and a collection of *resources*. We assume that each job requires an exclusive access to particular subset of the resources to execute. The resource-constrained scheduling problem can then be modeled as a multicoloring problem on the conflict graph. We address here the case where each task uses up to k resources. Hence, the conflict graph is an intersection graph of a collection of sets of size at most k, and is thus $k+1$-claw free.

A natural example of a limited resource is processors in a multi-processor system. In the biprocessor task scheduling problem, we are given a set of jobs and a set of processors, and each job requires the exclusive use of two dedicated processors. We are interested in finding a schedule which minimizes the sum of completion times of the jobs. In scheduling terms, this problem is denoted by $P|\text{fix}_j|\sum C_j$, with $|\text{fix}_j| = 2$. In the special case of two resources per task, such as the biprocessor task scheduling problem, the conflict graph is a line graph.

Another application of sum multicoloring of $(k+1)$-free graphs is scheduling data migration over a network, also known as the *file transfer problem* (see, e.g., [CG+85, K03]). Suppose that the network is fully connected, and we have a set of files $f_1, \ldots, f_M$; each file f_i needs to migrate from a source s_i to a destination t_i. During migration, f_i requires the exclusive access to s_i and t_i, for a prespecified time interval; thus, f_i and f_j are in conflict if $\{s_i, t_i\} \cap \{s_j, t_j\} \neq \emptyset$. Our goal is to find a migration schedule that minimizes the

sum of completion times, where the sum is taken over all files. This translates to the sum multicoloring problem on the conflict graph of the files, which is a line graph.

1.5 Related work

The sum coloring problem was introduced in [K89], and the sum multicoloring problem in [BHK$^+$00]. The paper studies two variants of `SMC`. In the *non-preemptive* `SMC` (`npSMC`), we require that the set of colors assigned to each vertex is *contiguous*, while in the *preemptive* `SMC` (`pSMC`) we allow any proper coloring of G. Table 1 summarizes the known results for these problems in various classes of graphs. New bounds given in this paper are shown in boldface. The last two columns give known upper bounds for preemptive and non-preemptive `SMC`. Entries marked with $\cdot$ follow by inference, either by using containment of graph classes (comparability and interval graphs are perfect), or by `SC` being a special case of `SMC`. When omitted, [BBH$^+$98] is the references for `SC` and [BHK$^+$00] for `SMC`.

	SC		**SMC**	
	u.b.	*l.b.*	`pSMC`	`npSMC`
General graphs	$\cdot$	$n^{1-\epsilon}$	$n/\log^2 n$	$n/\log n$
Perfect graphs	4	$\cdot$	16	$O(\log n)$
Comparability	**1.796**	$\cdot$	**7.184**	$\cdot$
Interval graphs	**1.796** (2 [NSS99])	$c > 1$ [G01]	$\cdot$	$\cdot$
Bipartite graphs	27/26 [GJ$^+$02]	$c > 1$ [BK98]	1.5	2.8
Line graphs	2	NPC	2	**12**
Partial k-trees	1 [J97]		PTAS [HK99]	FPAS [HK99]
Planar graphs	$\cdot$	NPC [HK99]	PTAS [HK99]	PTAS [HK99]
Trees	1 [K89]		PTAS [HK$^+$99]	1 [HK$^+$99]
$k+1$-claw free	$k+1$		$k+1$	$\mathbf{4k^2 - 2k}$

Table 1: Known results for sum (multi-)coloring problems

Resource-constrained scheduling has been recently investigated in the literature (see, e.g., [BKR96, K96]). Kubale [K96] studied the complexity of scheduling biprocessor tasks. He also investigated special classes of graphs, and showed that `npSMC` of line graphs of trees is NP-hard in the weak sense. Afrati *et al.* [AB$^+$00] gave a polynomial time approximation scheme for the problem that we consider, minimizing the sum of completion times of dedicated tasks. However, their method applies only to the case where the total number of processors is a fixed constant. Later their results were generalized in [FJP01] to handle weighted completion times and release times.

Very recently, our results for `SMC` on $k+1$-claw free graphs were improved in [K03], using methods due to [CP$^+$96]. The approximation ratio for line graphs (bi-processor tasks) was

improved to 10, and for general k the bound was improved to $O(k)$. It is worth noting that the [K03] paper uses exponentially large linear programs, which are solved with the ellipsoid method. Our algorithm has the advantage of being a much faster combinatorial algorithm. (In fact, we use a simple greedy algorithm). Moreover, it can be applied also in an online setting (see Section 3.3).

Coffman *et al.* [CG$^+$85] studied non-preemptive multicoloring of line graphs so as to minimize the overall number of colors (or *makespan*), which arises in the data migration problem. They showed that a class of greedy algorithms yields a 2-approximation and gave a $(2+\epsilon)$-approximation for a version with more general resource constraints. A comprehensive survey of other known results for scheduling dedicated tasks so as to minimize the overall completion time is given in [BC$^+$00].

Few analyses have been done on the robust throughput measure. Hassin and Rubinstein [HR02] gave an optimal bound of $\sqrt{2}$ for maximum (partial) weighted matching, improving on the obvious bound of 2 for a greedy selection.

2 Approximating Sum Coloring of Interval and Comparability Graphs

We consider algorithms for sum coloring on natural classes of graphs, and proceed to discuss the less studied measure of robust throughput. In Sections 2.1-2.3 we give a parameterized algorithm, ACS, and analyze its performance in terms of the sum coloring, robust throughput, and chromatic number measures. In Section 2.4 we analyze the robust throughput of two earlier algorithms, originally given for sum coloring.

2.1 An Algorithm Based on Finding k-Colorable Subgraphs

A common coloring strategy that often works well in practice is to iteratively color large independent sets. If we can find maximum independent sets, this is the MaxIS algorithm of [BBH$^+$98] which gives a 4-approximation for sum coloring. To improve on this idea, we may look towards generalizations. Viewing independent sets as 1-colorable subgraphs, we are easily led to the generalization of iteratively coloring maximum k-colorable subgraphs.

The algorithm schema ACS (*Assign Color Sets*) colors subgraphs in rounds; in round i it finds and colors a c_i-colorable subgraph, where $c_0, c_1, \ldots$ is a geometrically increasing sequence. The rounds correspond to the iterations of the main loop of ACS.

ACS assumes as a subroutine an algorithm kIS(G,k) for finding a maximum k-colorable subgraph, where k can be as large as the chromatic number of G. Such an algorithm can be run recursively to obtain a k-coloring of the subgraph that it finds. Namely, if $H_k =$ kIS(G, k), we can obtain $H_i =$ kIS(H_{i+1}, i), for $i = k-1, k-2, \ldots, 1$, with the color classes being $V(H_i) - V(H_{i-1})$, for $i = 1, \ldots, k$ (here, H_0 denotes the empty graph).

```
ACS(G, q, α)
 i ← 0; ℓ_i ← 0
 while (G ≠ ∅) do
   c_i = ⌊q^{i+α}⌋
   G_i ← kIS(G, c_i)
   Color G_i with colors ℓ_i + 1, ℓ_i + 2, ..., ℓ_i + c_i
   G ← G − G_i
   ℓ_{i+1} ← ℓ_i + c_i
   i ← i + 1;
 end
end
```

Figure 3: The coloring algorithm ACS

Given parameters $q \geq 1$ and $\alpha \in [0, 1)$, we denote by $c_i = \lfloor q^{i+\alpha} \rfloor$ the number of colors used in round i; $\ell_i = \sum_{j=0}^{i-1} c_j$ is the number of colors used by ACS *before* round i.

In the course of the analysis of the algorithm, we shall determine the optimal choice of the parameter q for the optimization measures at hand. The parameter α will be chosen uniformly at random from the interval $[0, 1)$. This can be simulated deterministically within any desired precision by trying all sufficiently closely spaced values in the range $[0, 1)$. The MaxIS algorithm of [BBH+98] corresponds to the choice of $q = 1$ and $\alpha = 0$, i.e., in each step a single independent set is extracted from the graph.

2.2 The Chromatic Sum of Algorithm ACS

In our analysis, we relate the quality of the algorithm's solutions to the optimal solutions via two intermediate functions. These functions fill the role of normalizing the color values given by the two solutions, as if the colors assigned in each round contribute equally to the objective function.

The following definitions are used throughout this subsection. We denote by ψ_α the coloring produced by ACS, for a given fixed α. Let g_v denote the round in which ACS colors v. Formally, $g : V \to \mathbb{N}$ is defined by $g(v) = g_v = i$ such that $\ell_i < \psi_\alpha(v) \leq \ell_{i+1}$. Let $\phi(i)$ denote the average of the colors used by ACS in round i, or

$$\phi(i) = \ell_i + (c_i + 1)/2. \tag{1}$$

Let $\psi_{\mathsf{OPT}}(G)$ denote the optimal sum coloring of G. Let $h : V \to \mathbb{N}$ be defined by $h(v) = h_v = i$ such that $c_{i-1} < \psi_{\mathsf{OPT}}(v) \leq c_i$. For convenience, let $c_{-1} = 0$. Note that

the quantities and functions g_v, h_v, c_i, ℓ_i and $\phi(i)$ are all functions of α and q, even if not explicitly marked so.

We note that the average of the colors of the vertices colored in round i is at most $\phi(i)$. This is because the color classes are non-increasingly ordered by the number of vertices assigned that color. This is coded in the following lemma.

Lemma 2.1 $\mathtt{SC}(G, \psi_\alpha) = \sum_v \psi_\alpha(v) \leq \sum_v \phi(g_v)$

Intuitively, we expect ACS to have colored a vertex v by round h_v, since v is within the first c_{h_v} color classes of the optimal solution, but not the first c_{h_v-1} classes. This is substantiated, on average, in the following claim.

Lemma 2.2 $\sum_v \phi(g_v) \leq \sum_v \phi(h_v)$

Proof: By the definition of ACS, it will have colored by round i at least as many vertices as those that have an h-value at most i. That is, for $i = 0, 1, \ldots,$

$$|\{v : g_v \leq i\}| \geq |\{v : h_v \leq i\}|. \tag{2}$$

Name the vertices $v_1, v_2, \ldots, v_n$ in non-decreasing order by g-values, and $u_1, u_2, \ldots, u_n$ in non-decreasing order by h-values. Then, by (2), $g(v_j) \leq h(u_j)$, for $j = 1, 2, \ldots, n$, and since ϕ is monotone non-decreasing, $\phi(g_{v_j}) \leq \phi(h_{u_j})$. The lemma now follows. □

In the next lemma we bound $\phi(h_v)$ by a closed form expression.

Lemma 2.3 *For $q \geq 1$ and $\alpha \geq 0$, let $f(q, k, \alpha) = q^{k+\alpha}[\frac{1}{q-1} + \frac{1}{2}]$. Then, for any $v \in V$,*

$$\phi(h_v) \leq f(q, h_v, \alpha) + 1/2 - q^\alpha/(q-1).$$

Proof: By definition,

$$\begin{aligned}
\phi(k) = \ell_k + (c_k + 1)/2 &= \sum_{i=0}^{k-1} \lfloor q^{i+\alpha} \rfloor + \lfloor q^{k+\alpha} \rfloor/2 + 1/2 \\
&\leq \sum_{i=0}^{k-1} q^{i+\alpha} + q^{k+\alpha}/2 + 1/2 \\
&= q^\alpha [\frac{q^k - 1}{q-1} + \frac{q^k}{2}] + 1/2 \\
&= f(q, k, \alpha) + 1/2 - q^\alpha/(q-1).
\end{aligned}$$

□

We can now deduce a bound on the worst-case performance ratio of a deterministic form of ACS.

Theorem 2.1 *The performance ratio of* ACS *for sum coloring is at most* $q(1/(q-1)+1/2)$, *when* $\alpha = 1$.

Proof: Observe that

$$\mathtt{SC}(G) = \sum_v \psi_{\mathtt{OPT}}(v) \geq \sum_v c_{h_v-1} + 1 \geq \sum_v q^{h_v-1+\alpha}. \tag{3}$$

When $\alpha \geq \log_q(q-1)/2$, Lemma 2.3 gives that $\phi(h_v) \leq f(q, h_v, \alpha)$. It then follows from Lemmas 2.1 and 2.2 that

$$\mathtt{SC}(G, \psi_\alpha) \leq \sum_v f(q, h_v, \alpha) = \sum_v q^{h_v+\alpha} \left[\frac{1}{q-1} + \frac{1}{2}\right]. \tag{4}$$

Combining (3) and (4), we find that the performance ratio is bounded by

$$\frac{\mathtt{SC}(G, \psi_\alpha)}{\mathtt{SC}(G)} \leq q \left[\frac{1}{q-1} + \frac{1}{2}\right].$$

When $q = 1 + \sqrt{2}$, this is at most $3/2 + \sqrt{2} \approx 2.91$. □

To obtain a better performance ratio, we analyze the expected performance of ACS when the parameter α is chosen uniformly at random from $[0, 1)$.

Lemma 2.4 *For any* $q \in (1, e^2]$,

$$E[\phi(h_v)] \leq \frac{q+1}{2\ln q} \psi_{\mathtt{OPT}}(v),$$

for any $v \in V$, *where the expectation is over the random choices of* α.

Proof: By the definition of h_v, the color $\psi_{\mathtt{OPT}}(v)$ of vertex v in the optimal solution satisfies

$$c_{h_v-1} = \lfloor q^{h_v-1+\alpha} \rfloor < \psi_{\mathtt{OPT}}(v) \leq c_{h_v} = \lfloor q^{h_v+\alpha} \rfloor. \tag{5}$$

Let us write $\psi_{\mathtt{OPT}}(v) = q^x$, i.e. $x = \log_q \psi_{\mathtt{OPT}}(v)$. Since $\psi_{\mathtt{OPT}}(v)$ is integral, we have in fact that

$$q^{h_v-1+\alpha} < q^x = \psi_{\mathtt{OPT}}(v) \leq q^{h_v+\alpha}.$$

Let $y_v = h_v + \alpha - x$ and note that y_v is in the range $[0, 1)$. We may write

$$y_v = (\alpha - x) \bmod 1.$$

The values $\psi_{\mathsf{OPT}}(v)$ and x are fixed and independent of α. Thus, when α is chosen uniformly at random from $[0,1)$, y_v is also uniformly distributed in $[0,1)$. The random variable q^{y_v} then has expected value

$$E[q^{y_v}] = \int_0^1 q^t dt = \frac{q-1}{\ln q}.$$

Hence,

$$E[q^{h_v+\alpha}] = E[q^{h_v+\alpha-x}] \cdot q^x = \frac{q-1}{\ln q}\psi_{\mathsf{OPT}}(v). \tag{6}$$

By Lemma 2.3,

$$E[\phi(h_v)] \leq E[f(q,h_v,\alpha)] + 1/2 - E[q^\alpha]/(q-1) = E[f(q,h_v,\alpha)] + (1/2 - 1/\ln q).$$

Thus, when $q \leq e^2$, then $\ln q \leq 2$, and we obtain that

$$E[\phi(h_v)] \leq E[f(q,h_v,\alpha)] \leq \psi_{\mathsf{OPT}}(v)\frac{q-1}{\ln q}\left[\frac{1}{q-1} + \frac{1}{2}\right] = \psi_{\mathsf{OPT}}(v)\frac{1+(q-1)/2}{\ln q}.$$

□

Theorem 2.2 *Let q be the solution of $\ln x = (x+1)/x$, or approximately 3.591. Then, the expected performance ratio of* ACS *for sum coloring is at most $q/2 \leq 1.796$.*

Proof: By Lemma 2.4, we get for this value of q that

$$E[\phi(h_v)] \leq \psi_{\mathsf{OPT}}(v) \cdot \frac{q+1}{2\ln q} = \psi_{\mathsf{OPT}}(v) \cdot q/2 \leq 1.796\,\psi_{\mathsf{OPT}}(v).$$

Combining Lemmas 2.1 and 2.2, and using the linearity of expectation, we get that

$$E[\mathtt{SC}(G,\mathsf{ACS}] \leq \sum_v E[\phi(h_v)] \leq \sum_v 1.796\,\psi_{\mathsf{OPT}}(v) \leq 1.796\,\mathtt{SC}(G).$$

□

Remarks Note that in our analysis we have only utilized the following obvious property of algorithm ACS. After the i-th round, ACS has colored as many vertices as contained in the first c_i colors of the optimal solution. We can argue that our analysis is tight for such restricted solutions. This would suggest that for general graphs, ACS would not perform better than shown here.

For the classes of graphs we are most concerned here, especially interval graphs, this is a non-optimally weak property. A tighter analysis would undoubtedly uncover better performance of the algorithm, but just as it has proved hard to give a good analysis of the MaxIS algorithm of [BBH$^+$98] for interval graphs, such an analysis would require some ideas that are not currently in play.

Derandomization We mentioned that ACS can be derandomized by trying sufficiently close values for α. To substantiate this, we need to examine the smoothness of the performance function.

Suppose that $\overline{a} \in [0,1)$ is the value for α that minimizes the performance ratio of ACS for sum coloring and let $\hat{a} = \overline{a} + \epsilon$, for some $\epsilon > 0$. Recalling that h_v and ϕ depend on the value of α, let us denote them as h_v^α and ϕ^α.

Observe that c_i and h_v are monotone with respect to the parameter α; thus, $c_i^{\hat{a}} \geq c_i^{\overline{a}}$ and $h_v^{\hat{a}} \leq h_v^{\overline{a}}$. Following the analysis of Lemma 2.3, we obtain that

$$\begin{aligned}
\phi^{\hat{a}}(h_v^{\hat{a}}) &= \sum_{i=1}^{h_v^{\hat{a}}-1} \lfloor q^{i+\overline{a}+\epsilon} \rfloor + \frac{\lfloor q^{h_v^{\hat{a}}+\overline{a}+\epsilon} \rfloor}{2} + \frac{1}{2} \\
&\leq q^\epsilon \Big(\sum_{i=1}^{h_v^{\overline{a}}-1} \lfloor q^{i+\overline{a}} \rfloor + \frac{\lfloor q^{h_v^{\overline{a}}+\overline{a}} \rfloor}{2} + \frac{1}{2} \Big) \\
&= q^\epsilon \phi^{\overline{a}}(h_v^{\overline{a}}), \qquad (7)
\end{aligned}$$

even if the possible increase in g_v is not considered. Set $\delta = q^\epsilon - 1$, in which case $\epsilon = \log_q(1+\delta)$. Then, from (7)

$$\begin{aligned}
\sum_v \phi^{\hat{a}}(h_v^{\hat{a}}) &\leq (1+\delta) \sum_v \phi^{\overline{a}}(h_v^{\overline{a}}) \\
&\leq (1+\delta) E[\sum_v \phi(h_v)] \\
&\leq (1+\delta) 1.796\, \mathtt{SC}(G).
\end{aligned}$$

Thus, it suffices to deterministically examine $1/\log_q(1+\delta) \approx (\log q)/\delta$ distinct values for α to get within $1+\delta$ factor of the randomized guarantee.

Implications Since the analysis gave expected values for each vertex separately, the performance ratio in Theorem 2.2 holds also for the vertex-weighted variant problem. Intuitively, we can view each weighted vertex as a compatible collection of unweighted vertices.

Corollary 2.3 ACS *approximates* $\mathtt{SC}(G)$ *within a factor of 1.796, even in the weighted case.*

This improves on the previously best ratio known of 2 by Nicoloso et al. [NSS99], stated for (unweighted) sum coloring interval graphs. This also yields a corollary for the *preemptive* sum multicoloring problem. In [BHK+00], a 4ρ-ratio approximation is obtained for graphs where SC can be approximated within a factor of ρ. We thus obtain an improvement, for graphs for which maximum size k-colorable subgraph is polynomial solvable, from the previous 16-factor.

Corollary 2.4 pSMC *is approximable within* $4 \cdot 1.796 \approx 7.184$ *on interval and comparability graphs and their complements.*

2.3 The Robust Throughput of Algorithm ACS

We proceed to analyze the throughput behavior of ACS.

Theorem 2.5 *For any graph* G, $\mathtt{RT}(G, \psi_\alpha) \leq 1.45751$. *This holds also in the weighted case.*

Proof: The offset parameter α is of no help here, and shall be set to 0. Further, for simplicity, we look only at the case when $q = 2$, which we have experimentally found to give best results.

Let k be any natural number. Recall that ψ_α is the coloring output by ACS. We compare the number of vertices found by ACS in the first k color classes, given by $Thr(\psi_\alpha, k)$, to the number of vertices in the first k colors of an optimal solution, given by $\mathrm{kc}(G)$. To simplify the notation in the calculations below, let $A(k) = Thr(\psi_\alpha, k)$ and $\mathcal{O}(k) = \mathrm{kc}(G)$.

Let $m = \lfloor \log k \rfloor$ denote the number of rounds. Recall that $c_i = 2^i$ and $\ell_i = 2^i - 1$. The set of vertices found by ACS then consists of the $A(\ell_m)$ vertices found in the first $m-1$ rounds, and at least a $(k - \ell_m)/k$ fraction of the vertices colored in the m-th round. The analysis uses the following simple observation. Since in each round i, $0 \leq i < m$, ACS finds a maximum c_i-colorable subgraph, clearly, the number of vertices colored in round i is at least a c_i/k-fraction of the vertices in $\mathcal{O}(k)$ that remained uncolored after round $i-1$.

$$A(\ell_{i+1}) - A(\ell_i) \geq \frac{c_i}{k}(\mathcal{O}(k) - A(\ell_i)).$$

Rewriting, we get

$$\mathcal{O}(k) - A(\ell_{i+1}) \leq (1 - \frac{c_i}{k})(\mathcal{O}(k) - A(\ell_i)).$$

By induction, we have that

$$\mathcal{O}(k) - A(\ell_m) \leq \mathcal{O}(k) \prod_{i=0}^{m-1} (1 - \frac{2^i}{k}). \tag{8}$$

In the last round, we count the number of vertices covered by ACS with the final $k - \ell_m$ colors. The fraction still not colored after that round amounts to

$$\mathcal{O}(k) - A(k) \leq (1 - \frac{(k - (2^m - 1))}{k})(\mathcal{O}(k) - A(\ell_m),$$

and from (8), we get that

$$\mathcal{O}(k) - A(k) \leq \mathcal{O}(k)\frac{2^m - 1}{k} \prod_{i=0}^{m-1} (1 - \frac{2^i}{k}). \tag{9}$$

By computational analysis, we find that the r.h.s. of (9) is maximized when $k/2^m \approx 1.3625$, converging to about 0.3139 $\mathcal{O}(k)$. This implies a performance ratio $\mathcal{O}(k)/A(k) \leq 1/(1 - 0.3139) \approx 1.4575$ for the throughput of the coloring, for a given value k. Since this bound is independent of k, we have derived a bound for $\mathtt{RT}(G, \psi_\alpha)$, the robust throughput of ACS. □

On graphs for which maximum size k-colorable subgraph is not polynomially solvable but approximable within a ρ-factor, we easily obtain a 1.4575ρ ratio for robust throughput.

Finally, we can argue a bound on the number of colors used.

Theorem 2.6 *The expected number of colors used by* ACS *is at most* $\frac{q}{\ln q}\chi(G)$. *When* $q = e \approx 2.718$, *this is at most* $e \cdot \chi(G)$.

Proof: Let m be the final round of ACS, i.e. the minimal integer satisfying $c_m = \lfloor q^{m+\alpha} \rfloor \geq \chi(G)$. Since ACS finished in round m and not in round $m-1$,

$$\lfloor q^{m+\alpha-1} \rfloor < \chi(G) \leq \lfloor q^{m+\alpha} \rfloor,$$

which, since $\chi(G)$ is integral, implies that

$$q^{m+\alpha-1} < \chi(G) \leq q^{m+\alpha}.$$

Let $x = \log_q \chi(G)$ and $o = m + \alpha - x$. Thus, $\chi(G) = q^x = q^{m+\alpha-o}$, for $o \in [0, 1)$.

The total number of colors used by ACS is at most

$$\sum_{i=0}^{m} q^{i+\alpha} \leq \frac{q^{m+\alpha+1}}{q-1} = \frac{q^{x+o+1}}{q-1} = \chi(G) \cdot \frac{q^{1+o}}{q-1}.$$

Following the proof of Lemma 2.4, the expected number of colors used by ACS is at most

$$E[q^o] \cdot \frac{q}{q-1}\chi(G) = \frac{q}{\ln q}\chi(G).$$

Indeed, since $E[q^o] \leq q$, it also follows that in the worst case ACS uses fewer than $q^2/(q-1) \cdot \chi(G)$ colors. □

We have seen that the same algorithm approximates all three measures: $\chi(G)$, $\mathtt{SC}(G)$, and robust throughput. The parameters used to obtain optimum values were not the same; however, with judicious choices of the parameters, one can obtain colorings that approximate simultaneously (in the expected sense) all the measures considered, as suggested in the following result.

Corollary 2.7 ACS, *with* $q = 2$, simultaneously *approximates robust throughput within an expected factor of 1.4575,* $\mathtt{SC}(G)$ *within a factor of 2.164, and* $\chi(G)$ *within a factor of 2.88.*

2.4 The Robust Throughput of Two Earlier Algorithms

2.4.1 Algorithm MaxIS

The MaxIS coloring algorithm is the iterative greedy method that colors in each round a maximum independent set, among the yet-to-be colored vertices. It was first considered for sum coloring in [BBH$^+$98], where it was shown to give a 4-approximation. This was shown to be the tight bound for this algorithm in [BHK99]. The ratio of 4 holds for graphs where the *maximum independent set (IS)* problem is polynomially solvable; this includes e.g. all perfect graphs. When IS is α-approximable, MaxIS yields a 4α-approximation for sum coloring.

For robust throughput, we obtain the following result.

Theorem 2.8 *The performance ratio of MaxIS for robust throughput is* $e/(e-1) \approx 1.58$, *and that is tight.*

Proof: Recall that a *k-colorable subgraph* in a graph G is a vertex subset $S \subset V$ that can be partitioned into k independent sets.

Let $\mathcal{O}(k)$ denote the cardinality of a maximum k-colorable subgraph H. Each independent set colored by MaxIS contains at least $1/k$ fraction of vertices that remain in H. Let $I(i)$ denote the number of vertices in the first i color classes found by MaxIS, where $I(0) = 0$. Then, the number of vertices remaining after iteration i, $i \geq 1$, is at most a $(k-1)/k$ fraction of the number of vertices before this iteration, that is,

$$\mathcal{O}(k) - I(i) \leq (1 - \frac{1}{k})(\mathcal{O}(k) - I(i-1)) \tag{10}$$

Applying (10) recursively, we get that

$$\mathcal{O}(k) - I(i) \leq (1 - \frac{1}{k})^i(\mathcal{O}(k) - I(0)) = (1 - \frac{1}{k})^i\mathcal{O}(k).$$

Taking $i = k$, we have

$$\mathcal{O}(k) - I(k) \leq (1 - \frac{1}{k})^k\mathcal{O}(k) \leq e^{-1}\mathcal{O}(k),$$

or

$$I(k) \geq \frac{e-1}{e}\mathcal{O}(k).$$

To show that the bound is tight, suppose that the optimal solution OPT consists of k equi-sized color classes $I_1, \ldots, I_k$, for some $k > 1$, and suppose that $|I_\ell| = k^s$, $s > k$, for $1 \leq \ell \leq k$. In the following we describe the structure of G that would allow MaxIS to select in each step an independent set which contains $1/k$ of the remaining vertices in I_ℓ, for all $1 \leq \ell \leq k$.

We partition the vertices in I_ℓ to $(s+1)$ subsets $A_{\ell,1}, \ldots, A_{\ell,s+1}$. The size of $A_{\ell,j}$ is given by

$$|A_{\ell,j}| = \frac{k^s(k-1)^{j-1}}{k^j}, \tag{11}$$

for $1 \leq j \leq s$, and $|A_{\ell,s+1}| = (k-1)^s$. The vertices in $A_{\ell,j}$ are connected by edges to all the vertices in $A_{m,h}$, $m \neq \ell$ and $h \neq j$; thus, the vertices $B_j = \cup_{\ell=1}^{k} A_{\ell,j}$ form an independent set. Also, from (11) we have that $|B_j| = \frac{k^s(k-1)^{j-1}}{k^{j-1}}$. Note that the vertices in $A_{1,s+1}$ can be connected in such a way that in iteration ℓ, MaxIS can select all the vertices in B_j plus a vertex in $A_{1,s+1}$; therefore, the resulting IS is slightly larger than the size of each of the remaining sets $I_1, \ldots, I_k$. This would make MaxIS select in each iteration $1/k$ of the remaining vertices in I_ℓ, $1 \leq \ell \leq k$.

Consider the end of iteration k. Then $\mathcal{O}(k) = |V| = k^{s+1}$, while MaxIS still needs to color

$$\frac{k^{s+1}(k-1)^k}{k^k} = \mathcal{O}(k)(\frac{k-1}{k})^k$$

vertices. For sufficiently large k, $((k-1)/k)^k$ is close to e^{-1}. Hence, we get that $I(k) \to \frac{e-1}{e}|V| = \frac{e-1}{e}\mathcal{O}(k)$. □

2.4.2 The algorithm of Nicoloso *et al.* for interval graphs

An algorithm of Nicoloso *et al.* [NSS99] for sum coloring interval graphs starts by computing $G_1, G_2, \ldots, G_{\chi(G)}$, where G_i is a maximum i-colorable subgraph. They show that when G is an interval graph and the G_i's are computed by a left-to-right greedy algorithm, then (a) G_i contains G_{i-1}, and (b) the difference set $G_i - G_{i-1}$ is 2-colorable. Thus, the algorithm colors G by coloring G_1 with the color 1, and coloring $G_i - G_{i-1}$ with $2i-2$ and $2i-1$, for $i > 1$.

Nicoloso *et al.* showed that the performance of the algorithm for `SC`, and simultaneously for chromatic number, was 2, and that it was tight. From this description of the algorithm, it is easy to derive bounds on its performance for the robust throughput. Since, for each k, a maximum k-colorable subgraph is fully colored with the first $2k-1$ colors, a ratio of 2 follows for robust throughput.

Observation 2.9 *The algorithm of [NSS99] attains a performance ratio of 2 for the robust throughput of interval graphs (as well as sum coloring and chromatic number).*

It should be fairly clear that these bounds on the performance of the algorithm cannot be improved. They also crucially depend on special properties of interval graphs. Also, it is not clear if these bounds hold when the algorithm is used on weighted graphs.

3 The npSMC problem on $(k+1)$-claw free graphs

In this section we give an approximation algorithm for **npSMC** on $k+1$-claw free graphs. As noted earlier, one application of this result for constant k is an $O(1)$-ratio approximation for the dedicated task scheduling problem $P|\text{fix}_j|\sum C_j$ with $|\text{fix}_j| = k$. Note that we place no constraints on the number of machines in the dedicated task scheduling problem, which can be arbitrarily fixed or unbounded. The constraint is only that the number of machines required by a given job is at most k.

3.1 Algorithm SG

Let G be a given $k+1$-claw free graph, and let β be a parameter dependent on k to be determined later. Informally, our strategy is the following. We allocate $x'(v) = (\beta+1)x(v)$ colors to each vertex v, or $\beta+1$ times more than required. We constrain this allocation so that the last $x(v)$ colors be contiguous, as they will form the actual set of colors assigned to v. We give higher priority to vertices with small $x(v)$ over vertices with larger $x(v)$. The allocation is performed one color at a time to a maximal independent set of vertices that have higher priority than others, either because they are shorter jobs, or because they have become *active*. Observe that the independent set is maximal only in the graph induced by the collection of not yet fully colored vertices. Active vertices are those v that have received at least $\beta x(v)+1$ colors (but fewer than $(\beta+1)x(v)$), and thus must receive a contiguous set until fully allocated.

We assume that all color requirements are different, since ties can be broken in a fixed but arbitrary way. Thus, the selected independent set I satisfies two constraints: (i) I contains all currently active vertices (ii) if a not fully colored vertex v does not participate in the independent set it either has an active neighbor, or it has a neighbor in the independent set with strictly smaller color requirement.

The logic for allocating an additional $(\beta+1)x(v)$ colors is to build a buffer so that a long job does not accidentally become active and delay many short jobs for a long time. This way, all the neighbors of v have fair chance to be colored to completion before v becomes active.

SG(G)

1. $j \leftarrow 1$.
2. **while** G is not empty **do**
   (a) $I_j \leftarrow$ all currently active vertices
   (b) **while** there exists a vertex with no neighbor in I_j **do**
      i. Let v be a vertex of minimum $x(v)$ among those that have no neighbors in I_j (and not in I_j themselves)

ii. $I_j \leftarrow I_j \cup \{v\}$

(c) Assign color j to the vertices in set I_j, decrease by 1 their color requirements in G.

(d) Update active vertices, and delete fully colored vertices from G.

(e) $j \leftarrow j + 1$.

3. The last $x(v)$ colors allocated to v form the coloring of v in the `npSMC` solution.

See Figure 4 for an illustration of the algorithm.

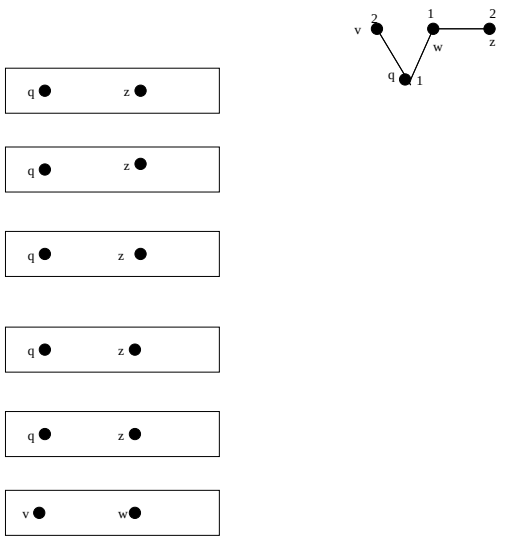


Figure 4: The first 6 independent sets selected by the algorithm. The graph is 3-claw free. Thus, $\beta = 4$. The algorithm allocates $(\beta + 1)x(q) = 5$ colors to q, and 10 to z. At round 5, q becomes fully colored and so v takes its place in I_6. Since z is not active yet, w which has a lower color requirement is intersected into the sixth independent set.

3.2 Analysis of SG

We use in the analysis the following notation. Throughout this section, fix some optimum solution, OPT, for the `npSMC` instance. A round consists of one iteration of the outer **while** loop of SG. Namely, a round is the process of choosing the next I_j. The first round is 1. Observe that the vertices included in I_j (namely, in the independent set of round j) have j as one of their assigned colors.

A vertex is *smaller* (*larger*) than its neighbor if its color-requirement is. Let $N(v)$ be the set of neighbors of v. The set of smaller (larger) neighbors of v is denoted $N_s(v)$ ($N_l(v)$). Let

O_v (respectively, O_v^s and O_v^l) denote the collection of neighbors of v (respectively, smaller and larger neighbors of v) colored before v in OPT. Similarly, let A_v, A_v^l and A_v^s be the vertices in $N(v)$, respectively, colored before v by algorithm SG, colored before v by SG and belonging to $N_l(v)$, and colored before v by SG and belonging to $N_s(v)$.

A not fully colored vertex is either selected, or delayed in a given round. It is *selected* if placed in the current independent set, in which case it is either *active* or *paying*. A vertex is active if it was given at least $\beta x(v) + 1$ colors. Thus, this vertex needs less than $x(v)$ additional colors (to get to $(\beta+1)x(v)$.) Active vertices are always inserted into the next independent set until they are fully colored. Thus, the vertices of I_j that are not active are paying vertices.

The vertex can either have a *good delay* in a round, if it has a smaller selected neighbor, or a *bad delay*, if it is delayed by a larger active neighbor.

We summarize this in the following fact. Let I be the current chosen independent set and I_a the set of active vertices, necessarily contained in I.

Fact 3.1 *In any given round, exactly one of the following holds for a given vertex v:*

1. Good delay: $I \cap N_s(v) \neq \emptyset$
2. Bad delay: $I_a \cap N_l(v) \neq \emptyset$, *and* $I \cap N_s(v) = \emptyset$
3. Selected: v *is paying or active.*

Let $d_g(v)$ ($d_b(v)$) denote the total good (bad) delay of v under SG. Fact 3.1 implies that the final color of v is given by $f_{SG}(v) = (\beta+1)x(v) + d_g(v) + d_b(v)$. We proceed to bound separately the good and bad delays. Define

$$Q(G) = \sum_v \sum_{w \in N_s(v)} x(w) = \sum_{vw \in E(G)} \min(x(v), x(w)).$$

The quantity $Q(G)$ provides an effective lower bound on the preemptive multicolor sum, and thus also on the contiguous one. Let $\mathcal{S}(G) = \sum_{v \in V} x(v)$.

Lemma 3.1 $Q(G) \leq k \cdot (\mathtt{pSMC}(G) - \mathcal{S}(G))$.

Proof: Recall that O_v are the neighbors of v colored before v in OPT. Define the *residual demand* of O_v as follows. The initial residual demand of O_v is $D_v^0 = \sum_{u \in O_v} x(u)$. Let D_v^i be the residual demand of O_v vertices after color i of OPT. Namely, let $x_i(u)$ denote the number of colors assigned to u be the first i color-classes in OPT (colors 1 to i), then $D_v^i = \sum_{u \in O_v} (x(u) - x_i(u))$.

We note that $D_v^{i+1} \geq D_v^i - k$, as for every $1 \leq j \leq i$ at most k of the neighbors of v belong to j color-class in OPT. This follows because the graph is $(k+1)$claw free. Since

the optimum does not start coloring v before the residual demand of O_v is 0, the minimum color assigned to v by OPT is at least D_v^0/k, and $\texttt{pSMC}(G) \geq \mathcal{S}(G) + \sum_v D_v^0/k$. Now, as every edge $e = (u, v)$ either contributes $x(u)$ to D_v^0 or $x(v)$ to D_u^0 we have: $\sum_v D_v^0 \geq Q(G)$. The required lemma follows. □

Lemma 3.2 *For any graph G, $\sum_v d_g(v) \leq (\beta + 1) \cdot Q(G)$.*

Proof: A smaller neighbor u of v is selected for at most $(\beta + 1) \cdot x(u)$ time units, and can delay v by at most that much. Thus, $d_g(v) \leq (\beta + 1) \sum_{w \in N_s(v)} x(w)$. □

Claim 3.2 *Consider v and round j so that $v \notin I_j$. Then at most $k-1$ of the vertices of A_v^l can be paying (namely selected but not active) at that round.*

Proof: By definition, if the round is good then $I_j \cap A_v^s \neq \emptyset$. Thus, $|I_j \cap A_v^l| \leq k-1$ because the graph is $(k+1)$claw free. In particular, at most $k-1$ of the A_v^l vertices can be paying.

In addition, if the delay is bad, by definition I_j must contain an *active* vertex u that belongs to A_v^l. It follows from the fact that the graph is $(k+1)$-claw free that at most $k-1$ A_v^l vertices can be paying. □

Lemma 3.3 *For any vertex v, $d_b(v) \leq \dfrac{k-1}{\beta - k + 1} \cdot d_g(v)$.*

Proof: The idea in the proof is to find a large collection of "events" all of which must occur in the $d_b(v) + d_g(v)$ rounds in which v is delayed.

These events are defined as follows. Observe that by definition, each round that has a bad delay for v has some $u \in A_v^l$ active at that round. Before u became active, there must have been $\beta \cdot x(u)$ rounds in which u was paying; this is because u became active. Thus the collection of events are all the times vertices of A_v^l were paying. Note that every such event must occur in the $d_g(v) + d_b(v)$ rounds that v is delayed. Indeed, by the definition of A_v^l, v must be delayed as long at least one vertex in A_v^l still need to pay.

Let $S_v = \sum_{u \in A_v} x(v)$, $S_v^l = \sum_{u \in A_v^l} x(v)$, and $S_v^s = \sum_{u \in A_v^s} x(v)$. The total number of times that some vertex in A_v^l was paying is exactly $\beta \cdot S_v^l$. By Claim 3.2 the total delay of v, $d_g(v) + d_b(v)$ is at least $\beta \cdot S_v^l/(k-1)$; this gives the total pay required by A_v^l vertices divided by the number of vertices simultaneously paying.

By the definition of A_v^l, $d_b(v)$ is at most S_v^l. In summary we have that $d_g(v) + d_b(v) \geq \beta d_b(v)/(k-1)$, and the lemma follows. □

Theorem 3.3 SG *approximates* $\texttt{npSMC}(G)$ *on $k+1$-claw free graphs by a factor of $2k(2k-1)$. In particular, it achieves a factor 12 on line graphs and proper interval graphs.*

Proof: Let $\beta = 2(k-1)$. Let $SG(G)$ denote the multicolor sum of our algorithm on G. Combining Lemmas 3.2, 3.3 and 3.1, we have that

$$\begin{aligned} SG(G) &\leq (\beta+1)\mathcal{S}(G) + (\beta+1)(1+\frac{k-1}{\beta-k+1})Q(G) \\ &= (2k-1)\mathcal{S}(G) + 2(2k-1)Q(G) \\ &\leq 2k(2k-1)\texttt{pSMC}(G) - (2k-1)^2\mathcal{S}(G). \end{aligned}$$

□

As mentioned before, among the various classes of graphs this result holds for are line graphs, proper interval graphs and the intersection graphs of families of unit circles.

Note that in the proof of Theorem 3.3 the npSMC optima is bounded by a constant times the pSMC optima. Therefore, the proof of Theorem 3.3 implies the following relation of the optimum preemptive and non-preemptive solutions, that was not known before.

Corollary 3.4 *For a $k+1$-claw free graph G, k a constant,* $\texttt{npSMC}(G) = O(\texttt{pSMC}(G))$.

Proof: Note that since SG finds a non-preemptive coloring, $\texttt{npSMC}(G) \leq \mathsf{SG}(G)$, while by (12), $\mathsf{SG}(G) \leq 2k(2k-1)\texttt{pSMC}(G)$. □

We observe that the number of colors used by our algorithm is within a constant factor of optimal. Although the bound obtained is inferior to the best algorithms designed for that purpose, it is interesting that we get a simultaneous approximation of both the chromatic number and sum multicoloring measures.

Theorem 3.5 SG *approximates the (multi-)chromatic number (i.e. makespan) $\chi(G,x)$ of $k+1$-claw free graphs by a factor of $2k(2k-1)$. In particular, it achieves a factor 12 for line graphs.*

Proof: Let v be a vertex so that $x(v)+\sum_{u\in N_s(v)} x(u)$ is maximum. In any legal coloring, at each round only at most k of the neighbors of v can be colored. Whenever v is selected, no neighbor of v can be paying. Thus we have that $\chi(G,x) \geq x(v)+\sum_{u\in N_s(v)} x(u)/k$. In turn, previous analysis gives that for every u, $d_b(u) \leq d_g(u)$, and $d_g(u) \leq (2k-1)\sum_{w\in N_s(u)} x(w)$. Thus, $f_{SG}(u) \leq x(u) + 2(2k-1)\sum_{w\in N_s(u)} x(w)$. The lemma follows from the choice of v. □

3.3 Extensions

Vertex weights We now show that our result can be extended to handle vertex weights. Jobs may have different priorities, which are reflected in the nonnegative *weight* $w(v)$ attached to each vertex v. The objective function becomes the sum of *weighted* final colors,

$\sum_v w(v)f(v)$. Straightforward modifications of the approach of Section 3.1 yield the same bound; the unweighted case was the one presented in detail just for the simplicity of exposition. Following [BHK+00], we change the priority order to follow the ratio of color requirement to the weight: v is preferred over u iff $x(v)/w(v) \leq x(u)/w(u)$. Update the definition $N_s(v)$ accordingly. Redefine $Q(G)$ as

$$Q(G) = \sum_{v \in V} w(v) \sum_{w \in N_s(v)} x(w) = \sum_{uv \in E} \min(w(v)x(u), w(u)x(v)).$$

Lemma 3.3 now follows unchanged, Lemma 3.2 holds with $d_g(v)$ modified to $w(v)d_g(v)$, and Lemma 3.1 was argued in [BHK+00]. Then we obtain the same ratio of $2k(2k-1)$ by Theorem 3.3.

Release times We now elaborate on the job scheduling application. Job scheduling is typically done dynamically, i.e., jobs arrive to the system at different times. The *release time* of a job, v, is a lower bound on the first color that can be assigned to v. An easy consideration shows that this has no detrimental effects on the performance of our algorithm, given its iterative nature. Our algorithm is online, in that it considers only jobs available at the given time, where "colors" are equivalent to "time steps".

A seemingly more difficult optimization measure (radically different than the sum of finishing times) is the the objective function of the flow time. In this measure our goal is to minimize the sum of completion time less the sum of release time. This seemingly difficult problem is open, even for sum colorings. Only the exact algorithms for trees and partial k-trees [HK99] are known to apply to this case.

Acknowledgments

We thank two anonymous referees for their helpful comments on the paper.

References

[AB+00] F. Afrati, E. Bampis, A. Fishkin, K. Jansen, and C. Kenyon. Scheduling to minimize the average completion time of dedicated tasks. In *Proc. of 20th Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science Vol. 1974, pages 454–464, 2000.

[BC+00] E. Bampis, M. Caramia, J. Fiala, A. Fishkin, A. Iovanella. Scheduling of Independent dedicated multiprocessor jobs. In *Proc. of 13th Annual International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science Vol. 2518, pages 391–402. 2002.

[BG+01] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the Throughput of Multiple Machines in Real-Time Scheduling. *SIAM Journal on Computing*, **31**:331–352, 2001.

[BK98] A. Bar-Noy and G. Kortsarz. The minimum color-sum of bipartite graphs. *Journal of Algorithms*, **28**:339–365, 1998.

[BBH+98] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation,* **140**:183–202, 1998.

[BF96] H.L. Bodlaender and B. de Fluiter. Parallel Algorithms for Series Parallel Graphs. *4th European Symposium on Algorithms*, Lecture Notes in Computer Science Vol. 1136, pages 277-289, 1996.

[B-98] P. Brucker. Scheduling Algorithms. 2nd ed., Springer, Heidelberg, 1998.

[BH94] D. Bullock and C. Hendrickson. Roadway traffic control software. *IEEE Transactions on Control Systems Technology*, 2:255–264, 1994.

[BHK99] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz. Tight Bound for the Sum of a Greedy Coloring. *Information Processing Letters* **71**, 135–140, 1999.

[BHK+00] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, H. Shachnai, and R. Salman. Sum Multicoloring of Graphs. *Journal of Algorithms*, **37**(2):422–450, November 2000.

[BKR96] P. Brucker and A. Krämer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research,* **90**:214–226, 1996.

[CP+96] S Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein and J. Wein. Improved Scheduling Algorithms for Minsum Criteria. *Proceedings 23rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 875-886, 1996.

[CG+85] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson and A. S. LaPaugh. Scheduling File Transfers. *SIAM J. Comput.* **14**:744–780, 1985.

[FMW97] S. Felsner, R. Müller and L. Wernisch. Trapezoid Graphs and Generalizations, Geometry and Algorithms. *Discrete Applied Mathematics*, **74**:13–32, 1997.

[F95] C. Flotow. On Powers of m-Trapezoid Graphs. *Discrete Applied Mathematics*, **63**:187–192, 1995.

[F80] A. Frank. On Chain and Antichain Families of a Partially Ordered Set. *J. Combinatorial Theory*, Series B, **29**:176–184, 1980.

[FJP01] A. V. Fishkin and K. Jansen and L. Porkolab. On minimizing average weighted completion time of multiprocessor tasks with release dates. *Proceedings 28th International Colloquium on Automata, Languages and Programming (ICALP)*, 2001, pages 875-886.

[GJ+02] K. Giaro, R. Janczewski, M. Kubale and M. Malafiejski. A 27/26-approximation algorithm for the chromatic sum coloring of bipartite graphs. *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, 2002, pages 135–145.

[G80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[G01] M. Gonen. Coloring Problems on Interval Graphs and Trees. M.Sc. Thesis, School of Computer Science, The Open Univ., Tel-Aviv, 2001.

[HK99] M. M. Halldórsson and G. Kortsarz. Multicoloring Planar Graphs and Partial k-trees. In *Proceedings of the Second International Workshop on Approximation algorithms* (APPROX '99). Lecture Notes in Computer Science Vol. 1671, Springer-Verlag, August 1999.

[HK+99] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multi-Coloring Trees. In *Proceedings of the Fifth International Computing and Combinatorics Conference (COCOON)*, Tokyo, Japan, Lecture Notes in Computer Science Vol. 1627, Springer-Verlag, July 1999.

[HR02] R. Hassin and S. Rubinstein. Robust Matchings and Maximum Clustering. *SIAM Journal of Discrete Mathematics*, **15**:530-537.

[J97] K. Jansen. The Optimum Cost Chromatic Partition Problem. *Proc. of the Third Italian Conference on Algorithms and Complexity (CIAC '97).* Lecture Notes in Computer Science Vol. 1203, pages 25–36, 1997.

[K03] Y. A. Kim. Data Migration to Minimize the Average Completion Time, *Proc. 14th Symposium on Discrete Algorithms (SODA)*, Jan 2003.

[K89] E. Kubicka. The Chromatic Sum of a Graph. PhD thesis, Western Michigan University, 1989.

[K96] M. Kubale. Preemptive versus non preemptive scheduling of biprocessor tasks on dedicated processors. *European Journal of Operational Research* **94**:242–251, 1996.

[KKK89] E. Kubicka, G. Kubicki, and D. Kountanis. Approximation Algorithms for the Chromatic Sum. In *Proceedings of the First Great Lakes Computer Science Conference*, Lecture Notes in Computer Science Vol. 1203, pages 15–21, Springer-Verlag, July 1989.

[MB+95] Marathe, M. V., Breu, H., Hunt III, H. B., Ravi, S. S., and Rosenkrantz, D. J. Simple heuristics for unit disk graphs. *Networks*, **25**:59–68, 1995.

[NSS99] S. Nicoloso, M. Sarrafzadeh and X. Song. On the Sum Coloring Problem on Interval Graphs. *Algorithmica*, **23**:109–126,1999.

[W97] G. Woeginger. Private communication, 1997.

[YG87] M. Yannakakis and F. Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Inform. Proc. Letters*, **24**:133–137, 1987.