RAIRO-Oper. Res. 59 (2025) 3621–3648
https://doi.org/10.1051/ro/2025140

RAIRO Operations Research
www.rairo-ro.org

AN EXACT ALGORITHM FOR THE MINIMUM SUM COLORING PROBLEM ON PARTIALLY DECOMPOSABLE GRAPHS

Fatma Tammal[1,*] and Méziane Aïder[1,2]

Abstract. Given an undirected graph G, the Minimum Sum Coloring Problem (MSCP) asks to find a legal vertex coloring of G using natural numbers that minimizes the total sum of the colors. In this paper, we propose an exact approach for MSCP using the modular decomposition tree of G, where our optimal solution is obtained at the root by the end of the process. This approach, called Modular Decomposition for Sum Coloring (MDSC), arises from the observation that decomposing graph G into disjoint subgraphs requires a careful selection of at least one solution from each subgraph, contributing to the final optimal solution. The modular decomposition technique aids us in making these selections intelligently. As a result, the branch and bound process becomes more efficient, faster, and powerful for solving MSCP on partially decomposable graphs. Numerical experiments demonstrate this improvement on several large DIMACS, COLOR 2002–2004 challenge graphs, and our generated instances, each containing between 500 and 1500 vertices.

Mathematics Subject Classification. 05C15, 05C85, 90C35.

Received November 4, 2024. Accepted October 13, 2025.

1. Introduction

The Minimum Sum Coloring Problem (MSCP) is a graph theory problem that seeks to find the minimum sum of colors assigned to vertices in a graph such that no two adjacent vertices share the same color. The minimum number of colors required to achieve this is called *the strength* of the graph. This problem finds wide applications in various fields such as scheduling [21], VLSI design [40, 42] and error-prone data [3].

Despite its seemingly simple definition, the MSCP is known as a challenging problem, particularly in its decision variant, where determining whether a graph can be colored using a given number of colors is NP-complete [28]. This complexity has spurred significant research interest, leading to the development of various heuristic and exact algorithms, aimed at tackling different aspects of the problem, as this will be discussed in the next section.

In graph optimization, defining a certain problem P often involves dividing the vertex set V or edge set E of a graph $G = (V, E)$ into subsets under specific constraints, where the complexity of the problem lies. Additionally, partitioning G into a set H of subgraphs (see Fig. 3) helps in reducing its complexity, allowing us to extract a

Keywords. Minimum sum coloring, strength of graph, proper coloration, modular decomposition, exact method.

[1] LaROMaD, Fac. Maths, USTHB, PB 32 Bab Ezzouar, 16111 Algiers, Algeria.
[2] EPROAD EA 4669, UPJV, 7 rue du Moulin Neuf, 80000 Amiens, France.
*Corresponding author: ftammal@usthb.dz

solution from each subgraph in H and deduce the solution of P from these obtained subgraph solutions. Modular decomposition of G explores this operation by finding sets of vertices that share the same neighbours outside the set. These subgraphs are known as modules, and contracting them into single vertices to obtain a reduced graph of G (see Fig. 4). Modular decomposition provides another criterion for classifying graphs into totally decomposable graphs (cographs), partially decomposable graphs, and indecomposable graphs (prime graphs) [5]. In terms of input data size, modular decomposition can improve the performance of algorithms on graphs by reducing the problem to be solved on reduced subgraphs (prime graphs, see Fig. 4).

Modular decomposition is a valuable tool for solving essential problems in algorithmic graph theory, including transitive orientation [17], recognition of several classes of graphs [36], and various combinatorial optimization problems. For a comprehensive understanding of this technique, refer to the survey in [16].

The modular decomposition theory can be seen as a special case of the theory of a partitive family [43]. Based on this theory, modular decomposition provides another view of G as a decomposition tree (see Fig. 2), denoted as $\mathrm{MD}(G)$. $\mathrm{MD}(G)$ is composed of the graph's strong modules, those that don't overlap with other modules. One of the strengths of $\mathrm{MD}(G)$ is its succinct representation of a graph's structure, which can be computed in polynomial time [37, 41]. Several algorithms for computing $\mathrm{MD}(G)$ have been proposed [38]. In this paper, we adopt the one described in [43].

In this work, we present a novel exact approach for solving MSCP on partially decomposable graphs using modular decomposition. Our contribution lies in the development of the "Modular Decomposition for Sum Coloring (MDSC)" algorithm, which efficiently exploits the structure of decomposable graphs to simplify the coloring process. By leveraging the modular decomposition tree, we strategically break down complex graphs into smaller subgraphs, allowing us to obtain partial solutions from each module. These partial solutions are then combined to achieve the optimal solution at the root of the decomposition tree. Our experimental results demonstrate that MDSC outperforms existing methods, such as CPLEX solver and column generation algorithms, particularly in terms of CPU runtime and the number of instances solved to optimality, showcasing its practical relevance and effectiveness for large-scale graph coloring problems.

The remaining part of the paper is organized as follows: first, we provide a state of the art concerning MSCP. Next, preliminary notions are given on coloring problems, and we briefly present the main concept of modular decomposition. In Section 4, we present the notion of sequences in graph coloring along with some new definitions and results that will aid us in the main study. Section 5 discusses the fundamental operations and properties in MSCP that are developed in this study to derive our final algorithm presented in Section 6. An execution example of the method is provided in Section 7. Before concluding, we present the results of our computational experiments conducted using our algorithm along with advanced discussions.

2. State of the art

MSCP has attracted significant attention since its introduction by Kubicka *et al.* [28]. This problem is known to be NP-hard in the general case.

Tables 1 and 2 provide a chronological overview of the various works in this field.

3. Preliminaries

Let us now introduce some relative concepts of graph coloring terminologies. An undirected simple graph is a couple $G = (V, E)$ where V is the set of vertices, and E is the set of the edges. All graphs considered in this paper are finite, undirected, and contain no loops or multiple edges. We set $n = |V|$ and $m = |E|$. The set of neighbours of a vertex v is denoted by $N(v)$. The degree of a vertex v, denoted by $\deg(v)$, is the number of edges incident to v. The maximum and minimum degrees of a graph are commonly denoted by Δ and δ respectively. The complement of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ where $uv \in \bar{E}$ if and only if $uv \notin E$. If G_1 and G_2 are two disjoint graphs, then their union $G_1 \cup G_2$ is the graph with vertex set $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and edge set $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$.

TABLE 1. Minimum Sum Coloring Problem in literature – part I.

Reference	Type[1]	Main results[2]
[35]	A	ϵ-approximation for interval graphs with $\epsilon < 2$
[18]	U	$s(G) \leq \left\lceil \frac{col(G)+\Delta(G)}{2} \right\rceil$ and $s(G) \leq \left\lceil \frac{\chi(G)+\Delta(G)}{2} \right\rceil^3$
[39]	E	Polynomial time algorithms for chain bipartite graphs and k-split graphs
[31]	E, A	Polynomial time algorithms for bipartite graphs with $\Delta \leq 3$ and 27/26-approx. for $\Delta = 5$
[1]	E	Polynomial time for a subclass of P_4-sparse graphs
[33]	U, L	Different lower and upper bounds using decomposition graphs into cliques, trees and paths
[6]	M	Tabu search algorithm
[12]	H, M	Hybridization of a genetic algorithm and a local search algorithm
[44]	H, M	Extraction of independent sets then using tabu search to form solutions
[2]	E, A	Polynomial time for a sub class of P_4-sparse graph and 2-approx. for general P_4-sparse graph
[4]	E, A	Prove the NP-hardness of MSCP for $(k+1)$-clawfree graphs with $k = 2, 4, 5$ and 2-approx. for unit square graphs
[13]	M, L	Lower bound by using the graph decomposition into cliques and adapted an ant colony optimization metaheuristic
[27]	M	Memetic algorithm based on tabu search with two neighborhoods
[32]	E	Evaluation of constraint programming capabilities
[26]	M, U, L	Stochastic hybrid evolutionary search algorithm for computing upper and lower bounds
[29]	U	Compute an algebraic and empirical upper bounds for $s(G)$
[24]	M	Bi-objective memetic algorithm using effective crossover with tabu search for mutation operator
[23]	E, M, U	Two formulations of MSCP are given and an iterated multi-start tabu search to get the upper bound
[20]	M	Parameter tuning for the bi-objective genetic algorithm
[15]	E, L	Two formulations of MSCP are given where the second one is solved by column generation and get tight lower bound
[45]	M, U, L	An extraction and backward expansion search approach is used to compute the upper and lower bounds
[25]	M	Bi-objective memetic algorithm is combined with the iterated variable neighborhood search

Notes. [(1)]E: Exact, H: Heuristic, M: Metaheuristic, U: Upper bound, L: Lower Bound, A: Approximation; [(2)]Upper and Lower bounds of the chromatic sum ($\Sigma(G)$) or the strength of the graph ($s(G)$); [(3)]$col(G)$: coloring number. $\chi(G)$: chromatic number. $\Delta(G)$: maximum degree of G.

A coloring of a graph $G = (V, E)$ is a mapping $c : V \to \mathbb{N}$ and a k-coloring is a coloring $c : V \to \{1, 2, \ldots, k\}$. A coloring is proper if adjacent vertices receive different colors. If there is a proper k-coloring for a graph G, then G is said to be k-colorable. A proper k-coloring may also be seen as a partition V into k independent sets $\{S_1, \cdots, S_k\}$ where each vertex in S_l is colored with color l, for $l \in \{1, \ldots, k\}$. The classical Graph Coloring Problem (GCP) on a graph G aims to find a proper $\chi(G)$-coloring, where $\chi(G)$ is the smallest number k for which G is k-colorable and is called the chromatic number of G while MSCP asks to find a proper k-coloring that minimizes the total sum of colors assigned to the vertices (we note in the rest of the paper MSCP(G)). This minimum sum coloring is called the chromatic sum of G and is denoted by $\Sigma(G)$. Hence, $\Sigma(G) = \min_{c \in \mathcal{C}} f(c)$, where $\mathcal{C}$ is the set of all proper k-colorings of G (for all possible values of k) and $f(c) = \sum_{i=1}^{n} c(v_i)$. We also observe that $f(c) = \sum_{l=1}^{k} l|S_l|$ where S_l is the set of vertices receiving color l in the coloration c. The smallest number of colors used in a minimum sum coloring of G is called the *strength* of G and is denoted by $s(G)$. Let UB_{hmt} [18], UB_a and UB_s [29] the upper bounds of $s(G)$, and let $\widehat{\mathrm{UB}} = \min\{\mathrm{UB}_a, \mathrm{UB}_s, \mathrm{UB}_{hmt}\}$. In the sequel, we take $\widehat{\mathrm{UB}}$ as an upper bound of $s(G)$.

Given a subset of vertices $M \subseteq V$, $G[M]$ is the subgraph induced by M (any edge in G between two vertices in M belongs to $G[M]$). The sets V and $\{v\}, \forall v \in V$ are trivial strong elements (trivial strong modules) of

TABLE 2. Minimum Coloring Sum Problem in literature – part II.

Reference	Type	Main results
[10]	E	MSCP has been solved on D-Wave quantum computer
[30]	L	Two new theoretical lower bounds based on sequences in the graph coloration
[14]	H, A	Performance study of the first fit algorithm on claw-free graphs
[34]	E, L	A column generation approach is proposed to solve the linear relaxation of a set partition-based formulation
[19]	H, M, U	Using hyper-heuristics to get tight upper bounds
[9]	E	A branch-and-price algorithm to solve the problem
[22]	M	A two-phase infeasible space metaheuristic approach is developed
[8]	A	A $(1.796+\epsilon)$-approximation algorithm is developed for the chordal graphs

V (of G). Let $\pi = \{m_1, \cdots, m_h\}$ be a partition of the vertices set of G, let m_i and m_j be disjoint sets in π, we say that m_i and m_j are adjacent if any vertex of m_i is adjacent to all the vertices of m_j, and we say non-adjacent if the vertices of m_i are all non-adjacent to the vertices of m_j. If, for all m_i, $i \in \{1, \ldots, h\}$, are modules of G, then π is a modular partition (or congruence partition) of G. For example, in Figure 1, the sets $\pi' = \{\{1\}, \{2\}, \ldots, \{12\}\}$ and $\pi'' = \{\{1,2,3\}, \{4,5,6\}, \{7\}, \{8,9,10,11,12\}\}$ are both partitions of the set V. However, π' represents a modular decomposition since all of its elements are trivial strong modules. In contrast, π'' does not form a modular decomposition because the set $\{1,2,3\}$ is not a module, even though the other subsets are. Additionally, we observe that in π'', the modules $\{4,5,6\}$ and $\{7\}$ are adjacent, while $\{4,5,6\}$ and $\{8,9,10,11,12\}$ are not adjacent. We can define a quotient graph $G_{/\pi}$ whose vertices are the parts (or modules) belonging to the modular partition π, two vertices v_i and v_j of $G_{/\pi}$ are adjacent if and only if the corresponding modules m_i and m_j are adjacent in G.

A non-trivial modular partition $\mathscr{P} = \{M_1, \cdots, M_h\}$ that only contains maximal strong modules is known as a maximal modular partition. For example, in Figure 4, the maximal modular partition is $\mathscr{P} = \{M_1, M_2, M_3, \{3\}, \{7\}\}$. We note that each graph has a unique maximal modular partition that can be obtained by several linear algorithms. In our study, we used Tedder *et al.* [43] algorithm which built MD(G) recursively by selecting a pivot, partitioning vertices by adjacency, and refining these partitions to identify strong modules. The full tree is computed in linear time of $O(n+m)$. Let MD(G) and $G_{/\mathscr{P}}$ be the corresponding modular decomposition tree and the quotient graph of $\mathscr{P}$, respectively, and let η be a node in MD(G). The degree of η, denoted by $\deg(\eta)$, is the number of its children. Based on the Modular Decomposition Theorem [17], the nodes of MD(G) are of two types, the first called degenerate node where we get two cases: if $G[M]$ is not connected then the node is labeled parallel, else, if $\bar{G}[M]$ is not connected then the node is labeled series. Otherwise, the second type is called a prime node. Consequently, the smallest prime graph is the P_4, *i.e.*, the chordless path on four vertices. It is easy to check that the degree of any prime node is at least 4. The totally decomposable graphs are P_4-free graphs, called cographs. In a modular decomposition, a graph is prime if it only contains trivial modules. To conclude this brief passage on modular decomposition terminology, the following figures (Figs. 1–4) show us a simple example of a decomposable graph G, its modular decomposition MD(G) and its quotient graph $G_{/\mathscr{P}}$.

MSCP can be formulated as an integer linear program by defining the binary variables:

$$x_u^l = \begin{cases} 1; & \text{if color } l \text{ is assigned to vertex } u, \\ 0; & \text{otherwise.} \end{cases}$$

The objective function is defined below, and the constraints are defined as follows:

– (1.b) asserts that each vertex must receive exactly one color;

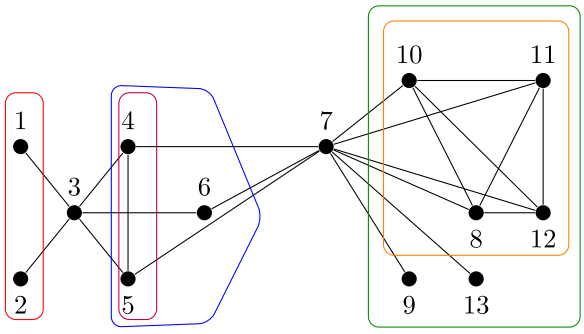

FIGURE 1. Example of a decomposable graph G.

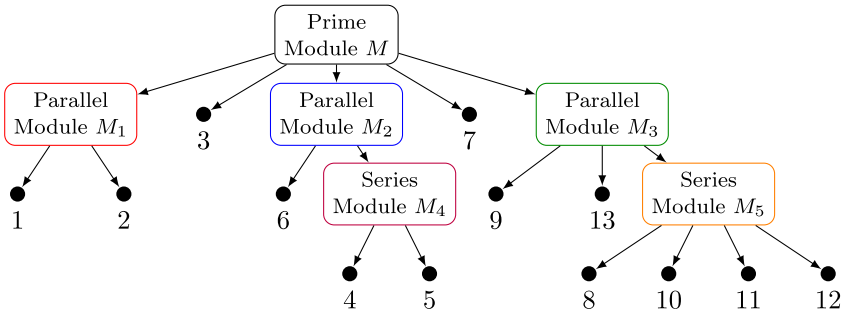

FIGURE 2. MD(G) of graph G.

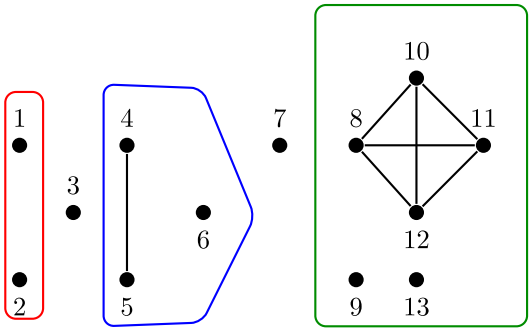

FIGURE 3. Strong module of $G_{/\mathscr{P}}$.

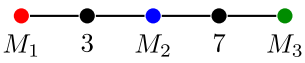

FIGURE 4. The quotient graph $G_{/\mathscr{P}}$.

– (1.c) asserts that every pair of adjacent vertices does not share the same color, this constraint is called edge-inequality.

So the formulation can be summarized as shown in Formulation 1.

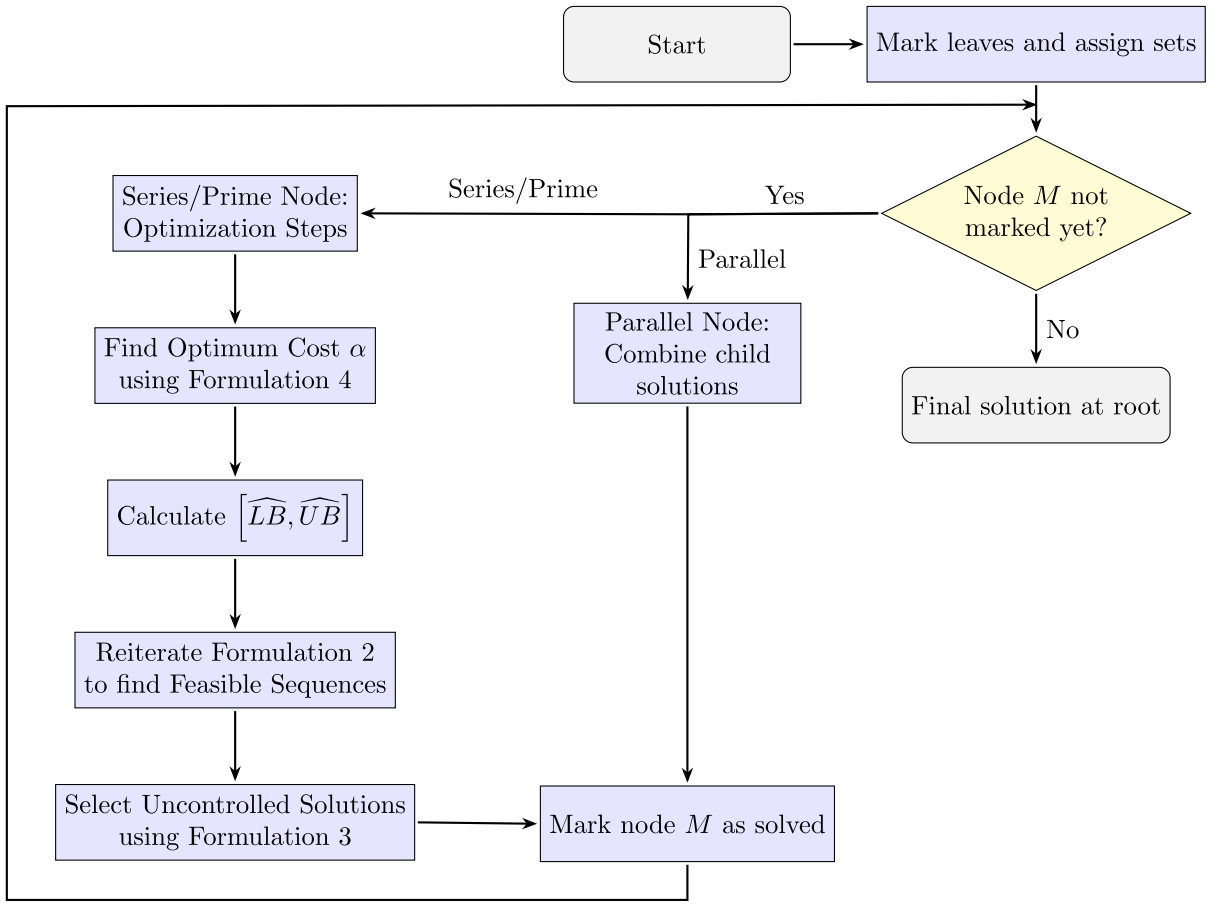

FIGURE 5. Flowchart of MDSC algorithm.

Let $D = \{x = ((x_u^l))_{u\in V}^{l=1,\ldots,\widehat{\mathrm{UB}}} |$ the constraints $(1.b)$, $(1.c)$ and $(1.d)$ are verified$\}$. In the following, we will use $x \in D$ to represent the three constraints.

We can easily assert that D contains all feasible solutions of MSCP. It will be used to simplify our formulations in the rest of the paper.

4. Modular decomposition for sum coloring algorithm

4.1. Overview

Before presenting the technical definitions and formulations, we first give an intuitive overview of the MDSC algorithm. Figure 5 summarizes its main steps.

The algorithm is designed to process the modular decomposition tree of the graph from the leaves up to the root:

(1) **Initialization at the leaves.** Each leaf node is marked, and its associated sets are assigned. These represent the simplest coloring units of the graph.

Formulation 1. Integer Linear Program of MSCP.

$$P(G)\begin{cases} \min \sum_{l=1}^{\widehat{\mathrm{UB}}} \sum_{u\in V} l x_u^l & (1.a) \\ \sum_{l=1}^{\widehat{\mathrm{UB}}} x_u^l = 1 \qquad \forall u \in V & (1.b) \\ x_u^l + x_v^l \leq 1 \qquad \forall uv \in E \; l = 1, \ldots, \widehat{\mathrm{UB}} & (1.c) \\ x_u^l \in \{0,1\}\,; \forall u \in V, \;\; l = 1, \ldots, \widehat{\mathrm{UB}} & (1.d) \end{cases}$$

(2) **Processing internal nodes.**

For *parallel nodes*, the solution is obtained by directly combining the solutions of their children.

For *series or prime nodes*, the algorithm requires optimization. It computes an optimal cost α, determines a possible range of color numbers, and then explores candidate sequences.

- Intuitively, a sequence describes how vertices are grouped into independent sets, for example, for a graph of 13 vertices, the sequence $(7, 2, 2, 1, 1)$ means "one color class of 7 vertices, two of size 2, and two singletons".
- A *feasible sequence* is one that corresponds to at least one proper coloring of the graph. Not all sequences are feasible, so the algorithm checks feasibility explicitly.
- Among feasible sequences, some correspond to the so-called *uncontrolled solutions*. These are solutions that either (i) achieve the optimal cost for the subgraph, or (ii) use no more than the strength number of the subgraph. Intuitively, uncontrolled solutions represent the "promising" partial solutions that can safely be carried upward in the decomposition tree.

(3) **Marking nodes and repeating.** Once a node is solved (either by combination or by selecting the best uncontrolled solutions), it is marked as complete. The process continues iteratively until no nodes remain.

(4) **Final solution.** The root of the decomposition tree contains the global optimal solution of the MSCP for the graph.

This high-level description aims to give the reader a clear picture of the algorithm's workflow: starting from simple cases at the leaves, handling different types of internal nodes, filtering solutions through feasibility checks, and finally combining them at the root.

The next subsections provide the precise mathematical background. Section 4.2 introduces the formal notion of feasible sequences, Section 4.3 explains fundamental operations and the role of uncontrolled solutions, and Section 4.4 details the complete algorithm.

4.2. Sequences in coloring theory

Let $G = (V, E)$ be a graph, with $|V| = n$. We observe that for any partition of V into k simple sets $\{S_1, S_2, \cdots, S_k\}$, we can associate a non-negative sequence p of at most n elements such that $p[i] = |S_i|$, for $i \in \{1, \cdots, k\}$ and $p[i] = 0$ for $i > k$. By a feasible sequence for G, we refer to a sequence that can be associated with at least one proper coloring in G. Moreover, if its corresponding independent sets are maximal, then p is said to be a maximal sequence. p is considered as an ordered sequence when its elements are arranged in a non-increasing order. We specifically consider finite-support non-negative ordered integer sequences where $|p| = \max\{i : p[i] > 0\}$.

In the sequel, we denote by Ω_n the set of all ordered sequences of G, Ω_n^k the set of all ordered sequences of V corresponding to k-partitions, and by $\Sigma p = \sum_{i=1}^{k} ip[i]$, the cost of the sequence p. Let us recall the following definitions of the notion of dominance.

Definition 1 ([1])**.** Let p and q be two integer sequences. We say that p dominates q, if for all $t \geq 1$ it holds that $\sum_{i=1}^{t} p[i] \geq \sum_{i=1}^{t} q[i]$.

The following lemma is a direct result of this definition.

Lemma 1 ([1]). *Let p and q be two sequences and let $k = \max\{|p|, |q|\}$. If p dominates q and $\sum_{i=1}^{k} p[i] = \sum_{i=1}^{k} q[i]$, then it holds that $\Sigma p \leq \Sigma q$.*

As a consequence, the minimum possible value of the cost of p among all possible sequences in Ω_n^k is given by the following sequence:

$$p = \left(n-k+1, \underbrace{1, \ldots, 1}_{k-1} \right).$$

We denote such a sequence p by d_n^k.

Property 1 ([29]). d_n^k dominates all sequences in Ω_n^k.

Accordingly, a dominated sequence in Ω_n^k is a sequence that has the maximum possible cost [29], and could be defined by:

$$p = \left(\overbrace{\left\lceil \frac{n}{k} \right\rceil, \cdots, \left\lceil \frac{n}{k} \right\rceil}^{n \bmod k \text{ times}}, \overbrace{\left\lfloor \frac{n}{k} \right\rfloor, \cdots, \left\lfloor \frac{n}{k} \right\rfloor}^{k-(n \bmod k) \text{ times}} \right).$$

We denote this sequence by b_n^k.

Example 1. Let G be the graph in Figure 1 such that $|V| = 13$, and let the feasible solution $\{\{1,2,4,6,8,9,13\},\{3,7\},\{5,10\},\{11\},\{12\}\}$ has its sequence $p = (7,2,2,1,1)$ with cost equal to $\Sigma p = 26$. It is clear that the sequence $q = (5,3,2,1,1,1)$ is dominated by p as defined in Lemma 1. For p, we have $n = 13$ and $k = 5$, so $d_{13}^5 = (9,1,1,1,1)$ and $b_{13}^5 = (3,3,3,2,2)$. Hence, d_{13}^5 domintes all sequences in Ω_{13}^5 and b_{13}^5 is dominated by all sequences in it.

To optimize these observations, we introduce the polynomial procedure *CostSharpMin*. This procedure aims to identify the first appearance of a given cost sequence q within Ω_n. It provides the minimum number of colors, denoted by $\widehat{\text{LB}}$, required to attain this specified cost.

Procedure 1: Cost-SharpMin.

```
Data: q, n
Result: LB̂
begin
    Initialization: k := 2;
    while (k ≤ |q|) do
        if (Σq ≤ Σb_n^k) then
            LB̂ := k;
            Terminate;
        else
            k := k + 1;
        end
    end
end
```

Utilizing this procedure, we obtain, for a given fixed cost α, a range of color numbers $k \in [\widehat{\text{LB}}, \widehat{\text{UB}}]$, where all solutions with this cost use k colors. Moreover, we lack knowledge about which sequences of them are feasible. So, we will employ the following formulations to determine this information.

We associate the integer decision variable y_i, which represents the cardinal of the independent set i with $i \in \{1, \cdots, k\}$. Formulation 2 allows us to find a sequence whose cost equals α using k colors if it exists. The model is defined as follow:

Formulation 2. Program to find a sequence with cost $= \alpha$ using k colors.

$$S(G,\alpha,k)\begin{cases} \min \sum_{i=1}^{k} y_i & (2.a)\\ \sum_{i=1}^{k} y_i = n & (2.b)\\ y_i - y_{i+1} \geq 0 \;\; i = 1,\ldots,k-1 & (2.c)\\ \sum_{i=1}^{k} i y_i = \alpha & (2.d)\\ y_i \geq 1 \;\; i = 1,\ldots,k. & (2.e)\end{cases}$$

- **Objective** (2.a) is optional, as it only guides the search trajectory. Its value is fixed by constraint (2.b), making it redundant for optimization but useful for initialization;
- **Constraint** (2.b) enforces that the solution must use exactly n vertices in total;
- **Constraint** (2.c) ensures that the sequence follows a strictly decreasing (inverse) order;
- **Constraint** (2.d) fixes the total cost of the sequence to a predefined value α;
- **Constraint** (2.e) guarantees that every subset in the sequence is non-empty (*i.e.*, contains at least one vertex).

After finding such a sequence using this formulation, we check its feasibility using Formulation 3 and store it in the set denoted by $\mathbb{F}$. The model is defined as follows:

Formulation 3. Program to assert the feasibility of a given sequence q.

$$F(G,\alpha,k,q)\begin{cases} \min \sum_{l=1}^{k}\sum_{u\in V} l x_u^l & (3.a)\\ \sum_{u\in V} x_u^l - \sum_{u\in V} x_u^{l+1} \geq 0 \qquad l = 1,\ldots,k-1 & (3.b)\\ \sum_{u\in V} x_u^l = q[l] \qquad l = 1,\ldots,k & (3.c)\\ \sum_{l=1}^{k}\sum_{u\in V} l x_u^l = \alpha & (3.d)\\ x = \left(\left(x_u^l\right)\right)_{u\in V}^{l=1,\ldots,\widehat{\mathrm{UB}}} \in D. & (3.e)\end{cases}$$

- **Objective** (3.a) is optional, serving only as a search trajectory guide. Its value is determined by constraint (3.d), rendering it redundant for optimization but potentially useful for initialization;
- **Constraint** (3.b) ensures that the size of the independent sets follows a strictly decreasing (inverse) order;
- **Constraint** (3.c) enforces that the solution must has exactly $q[l]$ vertices in each independent set l of the solution;
- **Constraint** (3.d) fixes the total cost of the solution to a predefined value α;
- **Constraint** (3.e) ensures the solution lies within D, the feasible set of MSCP on G.

The reiteration of Formulation 2 gives us all possible sequences in Ω_n^k such that $k \in [\widehat{\mathrm{LB}}, \widehat{\mathrm{UB}}]$, so the worst-case scenario, every sequence within Ω_n^k, might need feasibility verification by Formulation 3. However, once we identify feasible solutions and confirm their uniqueness in $\mathbb{F}$, we can discard the remaining alternative solutions.

4.3. Fundamental operations and proprieties in MSCP

In this paper, we construct an optimal solution using the modular decomposition tree. To achieve this, we introduce a new critical set called "uncontrolled solutions". We traverse our decomposition tree with this set, moving from the leaves to the root, thereby ensuring the optimality of the final solution.

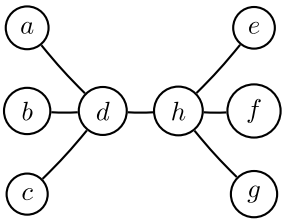

FIGURE 6. Partially decomposable graph G.

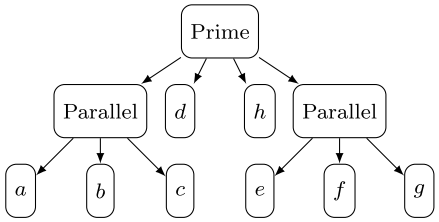

FIGURE 7. Modular decomposition of G.

As previously defined, a feasible sequence can correspond to many solutions. Consequently, the analysis of the problem with the solutions set is expected to be more challenging compared to the study of the feasible sequences set.

In the following, consider $G = (V, E)$ as a decomposable graph with its modular decomposition tree denoted as MD(G). This tree is formed by a set of strong modules $\mathscr{P} = \{M_1, \cdots, M_h\}$. Let X^i represent a proper coloring of module M_i, structured as: $X^i = \{S^i_1, \cdots, S^i_{|X^i|}\}$, where $|X^i|$ signifies the number of colors used in X^i.

Definition 2. An uncontrolled solution X^i of MSCP($G[M_i]$) is either an optimal-cost solution for $G[M_i]$ or a solution with $|X^i| \leq s(G[M_i])$; otherwise is a controlled solution.

Example 2. In Figure 6, an optimal solution of G is: $\{\{a, b, c, e, f, g\}; \{d\}; \{h\}\}$ with optimal cost equal to 11 and $s(G) = 3$. So all unconctrolled solutions must have cost equal to 11 and/or using $s(G)$ colors or less. In this case, the uncontrolled solutions of G are: $\{\{d, e, f, g\}; \{a, b, c, h\}\}$ and $\{\{a, b, c, e, f, g\}; \{d\}; \{h\}\}$.

Definition 3. Let $X^* = \left\{S^*_1, S^*_2, \cdots, S^*_{|X^*|}\right\}$ be an optimal solution of MSCP(G). Let $\mathscr{P}$ its maximal modular partition.

Let X^i an uncontrolled solution of MSCP($G\,[M_i]$), $\forall\ M_i \in \mathscr{P}$. We say that the uncontrolled solution X^i is a partial solution of X^* if it is properly contained in X^*, in other words:

$$\text{if for all } \{u, v\} \in S^i_l, \forall l \in \left\{1, \cdots, |X^i|\right\} \text{ then } \exists\ l^* \in \{1, \cdots, |X^*|\} : \{u, v\} \in S^*_{l^*}$$

and we denote: $X^i \subset^p X^*$.

Remark 1. By definition, every partial solution is an uncontrolled solution. The reverse is not true.

Example 3. Let $G = G_1 \vee G_2 \vee G_3$, where G_1, G_2, and G_3 are three copies of the graph showed in Figure 6, whose modular decomposition is presented in Figure 7. Figure 8 represents its MD(G). For each node that is a strong module $M_i, i \in \{1, \cdots, 10\}$, we have a partial solution from $G[M_i], i \in \{1, \cdots, 10\}$ that contributes to

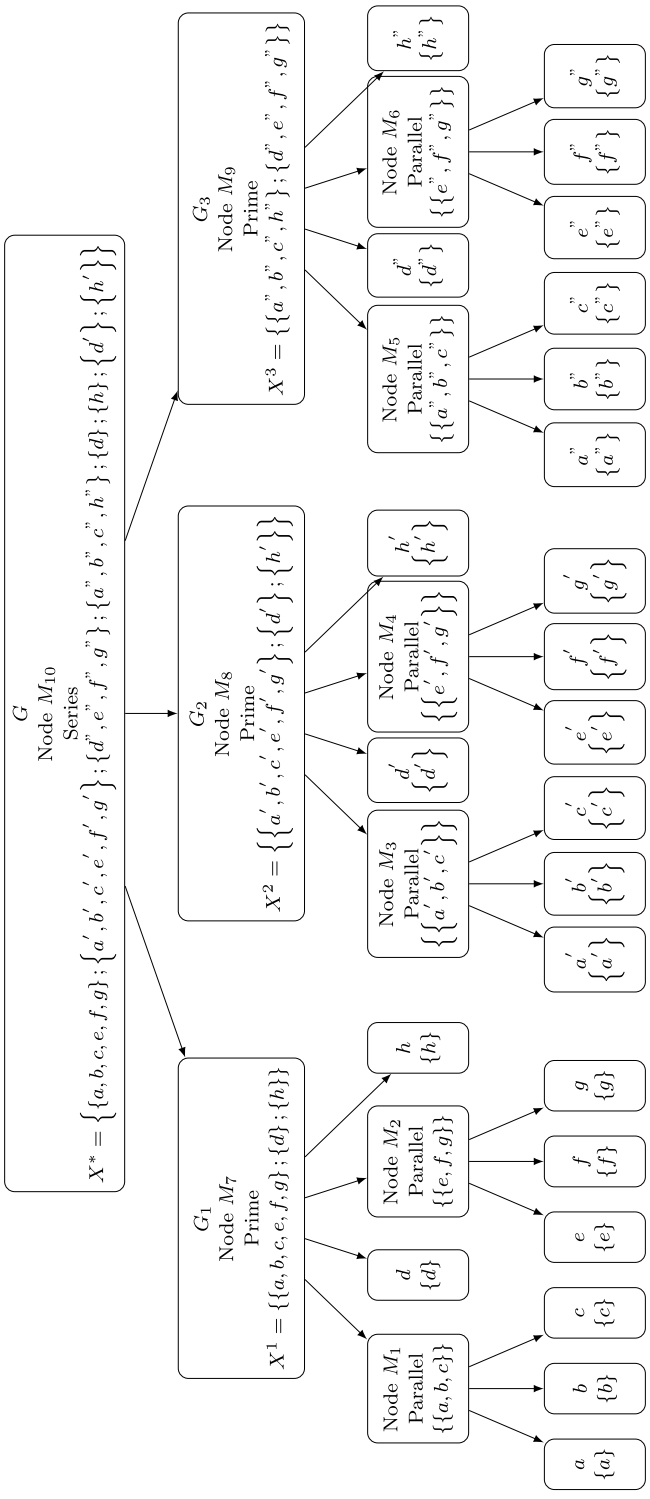

FIGURE 8. An optimal coloration of G.

construct our solution X^* in the root. Each subgraph $G_i, \forall i = 1, \cdots, 3$ has its own optimal solution, as shown in Example 2. In particular,we observe that the optimal solution of subgrap G_3 is an uncontrolled solution but not a partial solution of X^*. Meanwhile, the other uncontrolled solution of G_3 is a partial solution of X^*. Therefore, the global solution X^* must include at least one uncontrolled solution of each subgraph, whether it is the optimal one or not.

Definition 4. Let X^1 and X^2 be two solutions. The addition of X^1 and X^2, denoted by $X^1 + X^2$, is the solution that results from the union of independent sets in X^1 and X^2 having the same color.
Notice that: $\left|X^1 + X^2\right| = \max\left\{\left|X^1\right|, \left|X^2\right|\right\}$.

Remark 2. $\Sigma(X^1 + X^2) = \Sigma X^1 + \Sigma X^2$.

Now, all these definitions and observations are used to create solutions from the modular decomposition tree's leaves to reach the root. The process to get this solution is defined in the following results:

Lemma 2. *Let X^1, X^2 and X^3 be three solutions such that $|X^1| < |X^2|$. If $\Sigma X^1 < \Sigma X^2$ then $\Sigma(X^1 + X^3) < \Sigma(X^2 + X^3)$.*

Proof. We have $\Sigma X^1 < \Sigma X^2$ and by adding $\Sigma X^3 \geq 0$ to each term we get:

$$\Sigma X^1 + \Sigma X^3 < \Sigma X^2 + \Sigma X^3 \Leftrightarrow \Sigma(X^1 + X^3) < \Sigma(X^2 + X^3); \text{ by Remark 2.}$$

□

Lemma 3. *Let G_1 and G_2 be two disjoint graphs, and let $G = G_1 \cup G_2$. Then, every optimal solution X^* of G can be expressed as $X^* = X^1 + X^2$, where $X^1 \subset^p X^*$ from G_1 and $X^2 \subset^p X^*$ from G_2.*

Proof. Let $X^* = \{S_1^*, S_2^*, \cdots, S_k^*\}$ be an optimal solution of G. Let $S_{i_1}^*, S_{i_2}^*, \cdots, S_{i_t}^*$ be the sets in the partition of X^* having non-empty intersection with $V(G_1)$. We claim that $i_1 = 1, i_2 = 2, \cdots, i_t = t$ respectively. Otherwise, there is some $i \geq 1$ and $j \geq 1$ such that $S_i^* \cap V(G_1) = \emptyset$ and $S_{i+j}^* \cap V(G_1) \neq \emptyset$ where $i + j \leq k$. Since no vertex of G_1 is adjacent to a vertex of G_2 then vertices in $S_{i+j}^* \cap V(G_1)$ can migrate to S_i^*, obtaining a solution with sequence that strictly dominates that of X^*, a contradiction. The same happens to G_2, so X^* can be expressed as $X^* = X^1 + X^2$, where X^1 and X^2 are solutions of G_1 and G_2 respectively. So X^1 and X^2 are partial solutions. □

For each subgraph $G[M_i]$, there are several uncontrolled solutions whose number is represented by γ_i, and the jth-uncontrolled solution X_j^i has $|X_j^i|$ independent sets.

Definition 5. Let $X_j^i = \{S_1^i, S_2^i, \cdots, S_{|X_j^i|}^i\}$, be the j-th uncontrolled solution of MSCP($G[M_i]$); $i \in \{1, \cdots, h\}$ and $j \in \{1, \cdots, \gamma_i\}$. We put $\mathbb{F}^i = \{X_1^i, X_2^i, \cdots, X_{\gamma_i}^i\}$ be *the uncontrolled solutions set* with different sequences of MSCP on the subgraph $G[M_i]$.

Example 4. In Figure 6, if we take the subgraph $G[M_7]$, we find that $\gamma_7 = 2$ such that $X_1^7 = \left\{\underbrace{\{a, b, c, e, f, g\}}_{S_1^7}, \underbrace{\{d\}}_{S_2^7}, \underbrace{\{h\}}_{S_3^7}\right\}$ and $X_2^7 = \left\{\underbrace{\{a, b, c, h\}}_{S_1^7}, \underbrace{\{d, e, f, g\}}_{S_2^7}\right\}$.

In the next step, we manage to create another set that combines different uncontrolled solutions from several children of any node in MD(G).

Definition 6. Let $\mathscr{P}_i = \left\{m_1^i, \cdots, m_{\mathscr{H}}^i\right\}, \forall i \in \{1, \cdots, h\}$ be the modular decomposition of the subgraph $G[M_i]$. A combination of independent sets $\Im^c(G[M_i])$ formed by combining one selected uncontrolled solutions from each child $t, t \in \{1, \cdots, \mathscr{H}\}$ such that:

$$\Im^c(G[M_i]) = \left\{ \underbrace{S_1^1, S_2^1, \cdots, S_{|X_{j_1}^1|}^1}_{X_{j_1}^1}, \underbrace{S_1^2, S_2^2, \cdots, S_{|X_{j_2}^2|}^2}_{X_{j_2}^2}, \cdots, \underbrace{S_1^{\mathscr{H}}, S_2^{\mathscr{H}}, \cdots, S_{|X_{j_{\mathscr{H}}}^{\mathscr{H}}|}^{\mathscr{H}}}_{X_{j_{\mathscr{H}}}^{\mathscr{H}}} \right\}$$

where $c \in \{1, \cdots, \prod_{t=1}^{\mathscr{H}} \gamma_t\}$ which represents the cth-possible combination of all the children's uncontrolled solutions of M_i, which means that $X_{j_t}^t \in Z^t, \forall j_t \in \{1, \cdots, \gamma_t\}, \forall t \in \{1, \cdots, \mathscr{H}\}$.

We put: $\Im(G[M_i]) = \{\Im^1(G[M_i]), \cdots, \Im^{\prod_{t=1}^{\mathscr{H}} \gamma_t}(G[M_i])\}$.

The last set $\Im(G[M_i])$ contains all the possible combinations that can be held to obtain the optimal solution on the induced graph $G[M_i]$. By applying the same process node by node until we reach the root, we can obtain all the possible combinations of MD(G).

Definition 7. The modular solutions graph $G[M]_{/\Im^c} = (\Im^c, E_{\Im^c})$ whose vertices represent the parts (or stables) belonging to $\Im^c(G[M])$. Two vertices of $G[M]_{/\Im^c}$ are adjacent if and only if the corresponding solutions are solutions of adjacent modules in $G[M]_{/\mathscr{P}}$, or they are solutions of the same module.

For a given combination $\Im^c$, if $M_i, \forall i \in \{1, \cdots, h\}$ is a parallel node (*we are in the trivial case*) then $G[M_i]_{/\Im^c}$ is a disconnected graph. So, the optimal solution of $G\left[M_i\right]_{/\Im^c}$ is trivial, it is expressed by addition operations such that: $X_c^* = X_{j_1}^1 + X_{j_2}^2 + \cdots + X_{j_{\mathscr{H}}}^{\mathscr{H}}$, but if $X_{j_1}^1, X_{j_2}^2, \cdots, X_{j_{\mathscr{H}}}^{\mathscr{H}}$ are only optimal solutions of the M_i's children then we get the optimal solution of $G[M_i]$ directly. Otherwise, M_i is a series node or a prime node where $G\left[M_i\right]_{/\Im^c}$, is a complete graph or a prime graph respectively. For this *two cases*, we associate to $G[M_i]_{/\Im^c}$ a vector of vertex weight representing the cardinals of stables in this combination:

$$\begin{aligned} W_c &= \left(\left|S_1^1\right|, \cdots, \left|S_{|X_{j_1}^1|}^1\right|, \left|S_1^2\right|, \cdots, \left|S_{|X_{j_2}^2|}^2\right|, \cdots, \left|S_1^{\mathscr{H}}\right|, \cdots, \left|S_{|X_{j_{\mathscr{H}}}^{\mathscr{H}}|}^{\mathscr{H}}\right|\right) \\ &= \left(w_1, w_2, \cdots, w_{|\Im^c|}\right) \end{aligned}$$

Formulation 4 is a sum coloring integer program on a weighted graph $G[M_i]_{/\Im^c} = (\Im^c, E_{\Im^c}, W_c)$. We put initially $\alpha = +\infty$, and we can explain the model as follows:

Formulation 4. Program of minimum sum coloring problem on weighted graph.

$$W\left(G, c, W_c, \alpha, \widehat{\mathrm{UB}}\right) = \begin{cases} \alpha^c = \min \displaystyle\sum_{l=1}^{\widehat{\mathrm{UB}}} \sum_{u \in \Im^c} l w_u x_u^l & (4.a) \\ \displaystyle\sum_{u \in \Im^c} w_u x_u^l - \sum_{u \in \Im^c} w_u x_u^{l+1} \geq 0 \quad l = 1, \ldots, \widehat{\mathrm{UB}} - 1 & (4.b) \\ \displaystyle\sum_{l=1}^{\widehat{\mathrm{UB}}} \sum_{u \in \Im^c} l w_u x_u^l \leq \alpha & (4.c) \\ x = \left(\left(x_u^l\right)\right)_{u \in \Im^c}^{l=1,\ldots,\widehat{\mathrm{UB}}} \in D & (4.d) \end{cases}$$

- **Objective** (4.a) is the weighted cost of MSCP on $G[M]_{/\Im^c}$;
- **Constraint** (4.b) ensures that the solution follows a strictly decreasing (inverse) order;
- **Constraint** (4.c) fixes the total cost of the solution to be at most equal to α;
- **Constraint** (4.d) ensures the solution lies within D, the feasible set of MSCP on $G[M]_{/\Im^c}$.

With regard to our work, α is the optimum cost of node M_i where:

$$\alpha = \min\left\{\alpha^c,\ \forall c \in \left\{1, \ldots, \prod_{t=1}^{\mathscr{H}} \gamma_t\right\}\right\}$$

We can see that α will be updated after each execution of Formulation 4.

Based on these results, it is obvious that the optimal solution of MSCP X^* on the subgraph $G[M]$ is expressed by partial solutions of its strong modules. A detailed illustration of this last part is provided in the Running example section (Sect. 5).

4.4. Final algorithm

The algorithm is detailed as follows in Algorithm 2. It can be summarized at the flowchart in Figure 5.

Algorithm 2: MDSC Algorithm.

Data: G and MD(G)
Result: $X^*(G)$
Begin
  Let mark each leaf f of MD(G) and let be its solution formed by its singleton independent set;
  For each not marked node M of MD(G) that has all its children marked do:
    if *M is a* parallel *node (The trivial case)* **then**
      – The optimal solution of $G[M]$ is obtained by the addition operation of its children's optimal solutions;
      – Mark the node M;
    end
    if *M is a* series *node or a* prime *node* **then**
      – Find the optimum cost α of $G[M]$ and $s(G[M])$ the corresponding minimum number of colors, using Formulation 4;
      – Calculate $[\widehat{\mathrm{LB}}, \widehat{\mathrm{UB}}]$ using *Cost-SharpMin* procedure;
      – Reiterate the Formulation 2 to find the set of all sequences with the cost at least α using k colors ($k \in [\widehat{\mathrm{LB}}, \widehat{\mathrm{UB}}]$);
      – Select the uncontrolled solutions of $G[M]$ using Formulation 3 and form the set $\mathbb{F}$ which gives us the feasible sequences;
      – Mark the node M;
    end
  An optimal solution of G is the optimal solution found in the root of MD(G);
end

In the MDSC algorithm, Formulation 4 is first used to compute the optimal coloring cost α and an upper bound $\widehat{\mathrm{UB}}$ on the number of colors needed. Next, the CostSharpMin procedure is applied to determine a lower bound $\widehat{\mathrm{LB}}$ on the number of colors required to achieve cost α. Using these bounds, Formulation 2 is then employed iteratively to generate all candidate sequences whose coloring cost is exactly α and whose number of colors lies between $\widehat{\mathrm{LB}}$ and $\widehat{\mathrm{UB}}$. After each solution is obtained, a Dantzig cut [7] is added to exclude the current solution and force the discovery of new optimal sequences if they exist for the same actual number of colors. Once this set of candidate sequences is identified, Formulation 3 is used to filter out infeasible sequences, retaining only those that correspond to valid colorings of the weighted quotient graph. These actions ensure that the branch-and-bound process explores only the most promising combinations of uncontrolled solutions at series and prime nodes.

Theorem 1. *The algorithm MDSC gives an optimal solution for the problem MSCP with a finite number of iterations.*

Proof. The number of operations is finite on each node in MD(G). Hence, MDSC is a finite algorithm. Due to the uniqueness of MD(G) where the children's modules form a partitive family of maximal strong modules of G and based on previous lemmas, we can say that the MDSC gives an optimal solution of MSCP on the graph G where all the cases are treated (trivial and general cases). □

5. Running example

In this section we applied our algorithm on the MD(G) described in Figure 8:

- Let mark each leaf of MD(G) and let its solution be formed by its singleton independent set;
- For each not marked node M of MD(G) that has all its children are marked do:
- In parallel nodes:
  node M_1: has a unique combination $c \in \{1\}, \Im^1(M_1) = \{\{a\},\{b\},\{c\}\}$, $G[M_1]_{/\Im^1}$ is an independent set so the unique optimal solution of node M_1 is $\{a,b,c\}$ with a cost: 3
  the same treatment in node M_2, node M_3, node M_4, node M_5 and node M_6.
- In prime nodes:
  node M_7: all its children have a unique solution, so the parent has a unique combination: $c \in \{1\}, \Im^1(M_7) = \{\{a,b,c\};\{e,f,g\};\{d\};\{h\}\}$, $G[M_7]_{/\Im^1}$ is a prime graph, we apply the steps in the prime case:
  (1) The optimal solution of $G[M_7]$ is: $\{\{a,b,c,e,f,g\};\{d\};\{h\}\}$, $q = (6,1,1)$, $\alpha = 11$, $s(G[M_7]) = 3$.
  (2) Apply Cost-SharpMin procedure and get the interval: $[\widehat{\mathrm{LB}}, \widehat{\mathrm{UB}}] = [2,3]$.
  (3) Apply the Formulation 2 for $k = 2$ and $\alpha = 11$:

$$S(G[M_7],11,2) = \begin{cases} \min y_1 + y_2 \\ \quad y_1 + y_2 = 8 \\ \quad y_1 - y_2 \geq 0 \\ \quad y_1 + 2y_2 \geq 11 \\ \quad y_1 \ , \ y_2 \geq 1. \end{cases}$$

  The solution is: $(4,4)$.
  Apply the Formulation 2 for $k = 3$ and $\alpha = 11$:

$$S(G[M_7],11,3) = \begin{cases} \min y_1 + y_2 + y_3 \\ \quad y_1 + y_2 + y_3 = 8 \\ \quad y_1 - y_2 \geq 0 \\ \quad y_2 - y_3 \geq 0 \\ \quad y_1 + 2y_2 + 3y_3 \geq 11 \\ \quad y_1 \ , \ y_2 \ , \ y_3 \geq 1. \end{cases}$$

  The solutions are: $(6,1,1),(5,2,1),(4,3,1),(4,2,2),(3,3,2)$.
  The list of the obtained sequences is $\{(4,4),(6,1,1),(5,2,1),(4,3,1),(4,2,2),(3,3,2)\}$.
  (4) Apply the Formulation 3 gives us the list of feasible sequences: $Z^7 := \{(4,4),(6,1,1)\}$.
  (5) Mark the node M_7.
- The same treatment with node M_8 and node M_9. We get two uncontrolled solutions from each prime node in MD(G), so the number of combinations in the parent node is 8.
- In series nodes:
  The root: $G[M]_{/\Im^c}$, is a complete graph for all $c \in \{1,\cdots,8\}$.
  Series nodes are treated like prime nodes, so, we can summarize the 8 combinations that have different sequences into 4 cases as follows:

$$c_1 = \{\{a,b,c,h\},\{e,f,d,g\},\{a',b',c',h'\},\{e',f',d',g'\},\{a'',b'',c'',h''\},\{e'',f'',d'',g''\}\}.$$

TABLE 3. Categorized groups.

Set	Group	n	d	$\#R$	$\lvert R\rvert$
1	1	500	$]0.41, 0.45[$	$\{5, 6\}$	20
	2	1000	$]0.45, 0.50[$	$\{5, 6, 7\}$	
	3	1500	$]0.32, 0.41[$	$\{7, 10, 12, 13\}$	
	4	1000	$]0.25, 0.30[$	19	
2	1	500	$]0.18, 0.22[$	14	40
	2		$]0.24, 0.28[$	10	60
	3		$]0.22, 0.35[$	$\{7, 8\}$	80
	4		$]0.30, 0.40[$	$\{5, 6\}$	100

The cost of optimal solution of $G[M]_{/\Im^{c_1}}$: 84, with sequence $(4, 4, 4, 4, 4, 4)$.

$$c_2 = \{\{a, b, c, h\}, \{e, f, d, g\}, \{a', b', c', h'\}, \{e', f', d', g'\}, \{a'', b'', c'', e'', f'', g''\}, \{d''\}, \{h''\}\}.$$

The cost of optimal solution of $G[M]_{/\Im^{c_2}}$: 75, with sequence (6,4,4,4,4,1,1).

$$c_3 = \{\{b, c, a, e, f, g\}, \{d\}, \{h\}, \{b', c', a', e', f', g'\}, \{d'\}, \{h'\}, \{b'', c'', a'', e'', f'', g''\}, \{d''\}, \{h''\}\}.$$

The cost of optimal solution of $G[M]_{/\Im^{c_3}}$: 75, with sequence $(6, 6, 6, 1, 1, 1, 1, 1, 1)$.

$$c_4 = \{\{b, c, a, e, f, g\}, \{d\}, \{h\}, \{b', c', a', e', f', g'\}, \{d'\}, \{h'\}, \{d'', e'', f'', g''\}, \{b'', c'', a'', h''\}\}.$$

The cost of optimal solution of $G[M]_{/\Im^{c_4}}$: 72, with sequence $(6, 6, 4, 4, 1, 1, 1, 1)$.

– So we can see that the fourth combination gives us an optimal solution of G.

6. Experimental tests

6.1. Technical tools setup

The proposed MDSC algorithm is coded in C++ and compiled using the Microsoft® C++ compiler under the Windows® operating system. We utilize the CPLEX® solver to resolve the used linear programs. It is executed on an Intel® i7-10700KF processor running at 2×4.5 GHz, with 16 GB of RAM. For the modular decomposition software, we use the program described at this site: https://github.com/mogproject/modular-decomposition. It provides the modular decomposition tree with fast CPU runtime using the algorithm described in [43].

6.2. Dataset

We carried out experiments on four sets of instances. The first two consist of 133 instances from the DIMACS challenge and the COLOR 2002–2004 challenge, of which only 67, whose graphs are partially decomposable, are reported (graphs that are not decomposable are not reported). These instances were collected from http://dimacs.rutgers.edu/Challenges/ and http://mat.gsia.cmu.edu/COLOR04/ respectively. The last two are 40 instances for each set, randomly generated to have partially decomposable graphs. Each set is divided into 4 groups of 10 instances. They are classified according to the number of vertices (n), the density of the graph (d), the number of prime nodes ($\#R$) in the MD(G) and its maximum size ($|R|$). The differences between the groups are summarized in Table 3.

The first set is used to show the influence of the number of vertices, the density, and the number of prime nodes on the performance of the algorithm. The second set demonstrates the influence of prime nodes' size on the algorithm performance. More details will be provided in the next paragraph. As we will see in the experimental tests, neither the number nor the size of series nodes influences the algorithm performance, because they form complete graphs, which are quickly treated by the algorithm.

6.3. Experimental results

Since our method is an exact approach, its main purpose is to provide optimal solutions together with optimality certificates. For this reason, the most relevant benchmark is against other exact algorithms or general-purpose solvers (*e.g.*, CPLEX). While heuristic and metaheuristic methods exist for related problems, they do not provide guarantees of optimality and therefore serve a different purpose. Consequently, our comparisons focus on exact methods, which constitute the scientifically fair and natural baseline for evaluating our contribution. To the best of our knowledge, no recent branch-and-cut or SAT-based exact algorithms have been proposed for the MSCP. Therefore, our algorithm will be compared to the CPLEX solver and the method proposed by Delle Donne *et al.* [9], which employs a column generation approach tailored for resolving large instances.

The next four tables summarize all the detailed experimental results to show the efficiency of MDSC algorithm for the number of solved instances and the CPU runtime. For each instance, we report the number of vertices n, the number of edges m, the number of parallel nodes $\#P$, the number of the series nodes $\#S$, the number of the prime nodes $\#R$ and its maximum size $|R|$, the exact chromatic sum $\Sigma(G)$, the CPU runtime of MDSC algorithm, the CPU runtime of CPLEX solver, the exit gap of the CPLEX method computed as $100(ub-lb)/lb$, where ub and lb are the upper and lower bounds computed by CPLEX for the corresponding formulation within the time limit (0 for the instances solved to proven optimality) and for the column "CG", we report the CPU runtime of Delle Donne *et al.* [9] algorithm and its exit gap. A 2 h runtime limit is fixed in this paper. If the experiment was stopped by this criterion, then the CPU time is marked by "$*$", but if the mark is "$**$", then this means that we encountered an out-of-memory error before reaching the timeout limit for the CPLEX algorithm. If CPLEX or Delle Donne *et al.* [9] algorithm finds an upper bound but not a lower bound with the time limit then the gap is marked by "$-$" because it cannot be computed. Finally, if even the upper bound is not found by the time limit then the CPU and the gap are marked by "/". The best performances are highlighted in bold for CPU time when the solution is found; otherwise, the best GAP value is highlighted if available.

6.4. Discussion of the results

Our experiments were conducted in three phases. In the first phase, we tested our algorithm on well-established benchmarks, including DIMACS and COLOR 2002–2004. For each instance, we computed MD(G) and found that only 67 instances were partially decomposable. As shown in Tables 4 and 5, 42 instances were successfully solved, while the remaining 25 could not be resolved within the allotted time. These latter instances lack a practical modular decomposition because the prime nodes dominate most of the graph's vertices, leading to huge nodes that significantly hinder the resolution process.

In the second phase, we solved the first set of generated graphs. It became clear that the algorithm is not significantly affected by the graph's density but is somewhat influenced by its size and the number of prime nodes.

In the final phase, we addressed the second set of generated graphs. Here, we observed a decline in the algorithm's performance as the size of the prime nodes increased.

When comparing our algorithm to the CPLEX solver and Delle Donne *et al.* [9] method, it is clear that our approach significantly outperforms them in both CPU runtime and the number of instances solved to optimality. Specifically, our algorithm finds the optimal solution for 122 instances, compared to 45 instances by the CPLEX solver and 23 by the latter method. Although, for DIMACS instances, there is generally no substantial difference in CPU runtime between the CPLEX solver and our MDSC algorithm, MDSC notably outperforms CPLEX on the "inithx" instances. On the other hand, Delle Donne *et al.* [9] algorithm exhibited poor CPU runtime performance overall, except for the r.250.1c instance, where it found the optimal solution quickly, unlike the other two algorithms. The remaining generated instances in Tables 6 and 7 showcase the superior performance of the MDSC algorithm in terms of both CPU runtime and the number of instances solved to optimality. Additionally, several instances hit the memory limit when using the CPLEX solver, a limitation never encountered with the MDSC algorithm across all tested instances.

TABLE 4. DIMACS and COLORS benchmark graphs results – Part I.

Instance	n	m	$\#R$	$\#P$	$\#S$	$\|R\|$	$\Sigma(G)$	**MDSC**	**CPLEX**		**CG**	
									CPU	Gap	CPU	Gap
1-FullIns_3	30	100	1	0	1	29	54	**0**	**0**	0	**0**	0
1-FullIns_4	93	593	1	0	1	92	166	**3**	**3**	0	12	0
1-FullIns_5	282	3247	1	0	1	281	499	250	**246**	0	7200	–
2-FullIns_3	52	201	1	0	1	51	93	**0**	2	0	**0**	0
2-FullIns_4	212	1621	1	0	1	211	363	33	**18**	0	7200	–
2-FullIns_5	852	12 201	1	0	1	851	–	*	7200	**5.76**	7200	–
3-FullIns_3	80	346	1	0	1	79	145	**1**	**1**	0	7	0
3-FullIns_4	405	3524	1	0	1	404	683	**210**	214	0	7200	–
3-FullIns_5	2030	33 751	1	0	1	2029	–	*	**	–	/	/
4-FullIns_3	114	541	1	0	1	113	205	2	**1**	0	45	0
4-FullIns_4	690	6650	1	0	1	689	1138	**1549**	1595	0	7200	–
4-FullIns_5	4146	77 305	1	0	1	4145	–	*	**	–	/	/
5-FullIns_3	154	792	1	0	1	153	280	4	**2**	0	7200	–
5-FullIns_4	1085	11 395	1	0	1	1084	–	*	7200	**2.43**	/	/
Abb313GPIA	1557	65 390	1	59	0	1489	–	*	**	–	/	/
Anna	138	493	1	7	13	102	276	**2**	6	0	4217	0
David	87	406	1	1	9	64	237	**1**	4	0	20	0
DSJR500.1	500	3555	1	0	17	480	–	*	7200	**1.13**	/	/
DSJR500.1c	500	121 275	1	22	0	477	–	*	**	–	7200	–
DSJR500.5	500	58 862	1	0	3	497	–	*	**	–	/	/
fpsol2.i.1	496	11 654	7	20	44	119	3403	**66**	523	0	7200	–
fpsol2.i.2	451	8691	8	34	55	136	1668	**48**	515	0	7200	–
fpsol2.i.3	425	8688	8	34	54	136	1636	**48**	473	0	7200	–
games120	120	638	1	0	1	119	443	**1**	2	0	443	0
Homer	561	1629	1	35	47	328	1150	**20**	135	0	7200	–
Huck	74	301	1	5	13	41	243	**0**	1	0	243	0
inithx.i.1	864	18 707	16	62	99	157	3676	**181**	3117	0	7200	–
inithx.i.2	645	13 979	20	69	109	101	2050	**22**	1932	0	7200	–
inithx.i.3	621	13 969	19	72	105	100	1986	**21**	1880	0	7200	–
Jean	80	254	1	4	7	52	217	**0**	1	0	217	0
miles250	128	387	3	3	13	88	325	**0**	1	0	325	0
miles500	128	1170	1	0	8	114	705	**9**	11	0	705	0
miles750	128	2113	1	0	5	122	1173	**40**	57	0	1173	0
miles1000	128	3216	1	0	3	123	1666	**50**	71	0	1666	0
miles1500	128	5198	1	0	15	102	3354	**85**	95	0	3354	0
mug88_1	88	146	1	0	13	75	178	**0**	**0**	0	178	0
mug88_25	88	146	1	0	16	72	178	**0**	**0**	0	178	0
mug100_1	100	166	1	0	16	84	202	**0**	**0**	0	202	0
mug100_25	100	166	1	0	17	83	202	1	**0**	0	202	0
mulsol.i.1	197	3925	3	10	26	51	1957	**3**	27	0	7200	–
mulsol.i.2	188	3885	4	16	27	59	1191	**3**	34	0	1191	0
mulsol.i.3	184	3916	4	17	28	59	1187	**3**	35	0	1187	0
mulsol.i.4	185	3946	4	17	29	59	1189	**3**	35	0	1189	0
mulsol.i.5	186	3973	4	17	29	59	1160	**3**	36	0	1160	0

TABLE 5. DIMACS and COLORS benchmark graphs results – Part II.

Instance	n	m	$\#R$	$\#P$	$\#S$	$\lvert R\rvert$	$\Sigma(G)$	MDSC	CPLEX		CG	
									CPU	Gap	CPU	Gap
r.125.1	125	209	8	2	14	42	257	1	**0**	0	257	0
r.125.1c	125	7501	4	16	2	54	2184	**10**	323	0	2184	0
r.125.5	125	3838	1	0	4	121	–	*	7200	3.37	7200	**0.21**
r.250.1	250	867	1	0	19	228	704	**2**	8	0	7200	–
r.250.1c	250	30 227	1	21	1	227	5789	*	7200	7.5	**150**	0
r.250.5	250	14 849	1	0	3	247	–	*	7200	**7.91**	7200	9.64
r.1000.1	1000	14 378	1	0	21	979	–	*	7200	**20.04**	/	/
r.1000.1c	1000	485 090	1	24	0	974	–	*	/	/	7200	–
r.1000.5	1000	238 367	1	0	5	995	–	*	/	/	7200	–
school.1	385	19 095	1	2	8	370	–	*	**	–	7200	–
school.sch.1	352	14 612	1	2	8	339	–	*	**	–	7200	–
wap01a	2368	110 871	1	0	288	1571	–	*	**	–	/	/
wap02a	2464	111 742	1	0	305	1569	–	*	**	–	/	/
wap03a	4730	286 722	1	0	410	3713	–	*	**	–	/	/
wap04a	5231	294 902	1	0	556	3730	–	*	**	–	/	/
wap05a	905	43 081	1	0	62	748	–	*	**	–	/	/
wap06a	947	43 571	1	0	86	730	–	*	**	–	/	/
wap07a	1809	103 368	1	0	107	1611	–	*	**	–	/	/
wap08a	1870	1 041 176	1	0	133	1621	–	*	**	–	/	/
will199GPIA	701	7065	1	6	0	695	–	*	**	–	/	/
zeroin.i.1	211	4100	1	11	9	80	1822	**18**	29	0	92	0
zeroin.i.2	211	3541	2	14	15	95	1004	**16**	32	0	7200	–
zeroin.i.3	206	3540	1	14	16	95	998	**16**	31	0	7200	–

Another way to confirm our judgement is to use the performance profile plot as showed in Figure 9. We included Dolan–Moré performance profiles [11], which provide a robust way to compare solvers across a heterogeneous set of instances. For each instance i and solver s, we denote by $t_{i,s}$ the CPU time and define the performance ratio

$$r_{i,s} = \frac{t_{i,s}}{\min\limits_{s' \in \mathcal{S}} t_{i,s'}},$$

i.e., the ratio of the runtime of solver s to the best runtime achieved on instance i. The performance profile of solver s is then

$$\rho_s(\tau) = \frac{1}{|\mathcal{I}|} \left|\{i \in \mathcal{I} : r_{i,s} \leq \tau\}\right|,$$

which represents the fraction of instances solved within a factor τ of the best solver.

Special cases were handled as follows: timeouts ("*") were replaced by $t_{i,s} = 7200\,\text{s}$, out-of-memory errors ("**") by $t_{i,s} = 10\,800\,\text{s}$, instances unsolved by all solvers by $t_{i,s} = 72\,000\,\text{s}$, and out-of-memory errors on such unsolved instances by $t_{i,s} = 100\,000\,\text{s}$. This penalty scheme ensures consistent ratios without biasing solvable cases. Since such substitutions may create very large ratios, we plotted the τ-axis on a logarithmic scale to improve readability.

The results (Fig. 9) show that our method (MDSC) consistently outperforms competitors: it solves about 80% of the instances within a factor close to the best solver, while CPLEX and Delle Donne *et al.* [9] method require larger ratios before reaching similar coverage. This demonstrates both the efficiency and robustness of MDSC Algorithm.

TABLE 6. Generated graphs benchmark results – first group.

Instance	n	m	$\#R$	$\#P$	$\#S$	$\lvert R\rvert$	$\Sigma(G)$	MDSC	CPLEX		CG	
									CPU	Gap	CPU	Gap
G1	500	54 943	6	15	23	20	11 772	**1**	5154	0	7200	–
G2	500	54 908	6	13	24	20	13 815	**2**	5188	0	7200	–
G3	500	51 755	6	19	18	20	6050	**1**	7200	0.92	7200	–
G4	500	51 826	6	20	17	20	6158	**1**	**	/	7200	–
G5	500	51 819	6	19	20	20	6356	**1**	**	/	7200	–
G6	500	56 431	6	10	28	20	15 765	**1**	7200	0.08	7200	–
G7	500	53 389	6	16	22	20	10 037	**2**	**	0.3	7200	–
G8	500	50 240	6	22	14	20	4093	**1**	7200	–	7200	–
G9	500	53 366	6	16	22	20	9933	**2**	7200	0.27	7200	–
G10	500	53 365	6	16	23	20	9922	**1**	5062	0	7200	–
G11	1000	259 328	5	41	39	20	25 302	**2**	**	/	7200	–
G12	1000	271 315	5	38	40	20	27 371	**4**	**	/	7200	–
G13	1000	218 490	5	36	40	20	22 082	**2**	**	/	7200	–
G14	1000	266 731	7	36	40	20	26 193	**3**	**	/	7200	–
G15	1000	246 295	7	34	42	20	25 179	**3**	**	/	7200	–
G16	1000	238 972	5	38	42	20	24 321	**3**	**	/	7200	–
G17	1000	225 542	5	38	42	20	18 328	**2**	**	/	7200	–
G18	1000	271 336	5	38	42	19	27 439	**3**	**	/	7200	–
G19	1000	266 335	5	42	36	20	19 831	**2**	**	/	7200	–
G20	1000	266 572	6	39	41	20	23 419	**2**	**	/	7200	–
G21	1500	380 041	10	52	65	20	30 816	**3**	**	/	7200	–
G22	1500	420 124	7	59	59	20	35 620	**3**	**	/	7200	–
G23	1500	433 796	7	57	60	20	44 052	**5**	**	/	7200	–
G24	1500	399 669	7	57	60	20	34 396	**4**	**	/	7200	–
G25	1500	433 796	7	58	60	20	43 146	**5**	**	/	7200	–
G26	1500	382 746	12	47	65	20	36 590	**4**	**	/	7200	–
G27	1500	371 024	12	49	63	20	27 922	**4**	**	/	7200	–
G28	1500	454 299	7	59	58	20	45 418	**6**	**	/	7200	–
G29	1500	379 258	7	55	60	20	33 239	**4**	**	/	7200	–
G30	1500	380 520	13	42	68	20	39 846	**4**	**	/	7200	–
G31	1000	142 906	19	30	43	20	22 549	**5**	**	/	7200	–
G32	1000	143 062	19	29	42	20	20 753	**4**	**	/	7200	–
G33	1000	137 075	19	37	29	20	10 180	**4**	**	/	7200	–
G34	1000	143 836	19	26	49	20	22 259	**6**	**	/	7200	–
G35	1000	134 713	19	33	37	20	16 961	**4**	**	/	7200	–
G36	1000	147 196	19	27	51	20	30 650	**4**	**	/	7200	–
G37	1000	141 022	19	31	41	20	16 639	**4**	**	/	7200	–
G38	1000	136 749	19	32	42	20	19 280	**4**	**	/	7200	–
G39	1000	137 302	19	29	44	20	20 231	**4**	**	/	7200	–
G40	1000	142 994	19	33	35	20	16 674	**4**	**	/	7200	–

Limitation and futur works. To better understand the factors affecting the performance of MDSC Algorithm, we define two structural ratios derived from the modular decomposition tree MD(G).

Reduction ratio: RR which captures the gain from reducing the original graph to a smaller reduced instance with $|R|$ vertices:

$$\mathrm{RR} = \frac{n - |R|}{n}.$$

TABLE 7. Generated graphs benchmark results – second group.

Instance	n	m	$\#R$	$\#P$	$\#S$	$\lvert R\rvert$	$\Sigma(G)$	**MDSC**	**CPLEX**		**CG**	
									CPU	Gap	CPU	Gap
G41	500	23 183	14	3	14	40	5666	**84**	7200	34.69	7200	–
G42	500	27 009	14	1	8	40	5368	**232**	7200	37.12	7200	–
G43	500	27 089	14	0	7	40	5491	**223**	7200	35.03	7200	–
G44	500	26 967	14	0	6	40	5329	**142**	7200	36.09	7200	–
G45	500	27 217	14	0	10	40	5598	**223**	7200	36.24	7200	–
G46	500	23 213	14	1	10	40	5609	**187**	7200	35.66	7200	–
G47	500	27 417	14	0	9	40	5968	**233**	7200	37.21	7200	–
G48	500	23 264	14	1	11	40	5765	**347**	7200	35.02	7200	–
G49	500	27 051	14	0	10	40	5449	**237**	7200	36.93	7200	–
G50	500	22 039	14	1	12	40	5368	**105**	7200	35.17	7200	–
G51	500	30 168	10	5	1	60	2978	**28**	**	/	7200	–
G52	500	31 020	10	5	1	60	3792	**170**	7200	47.95	7200	–
G53	500	31 229	10	4	1	60	3763	**210**	**	/	7200	–
G54	500	31 390	10	5	1	60	3912	**231**	7200	51.36	7200	–
G55	500	31 120	10	8	1	60	4032	**239**	**	/	7200	–
G56	500	31 446	10	6	1	60	4017	**288**	7200	49.41	7200	–
G57	500	31 227	10	5	1	60	3982	**252**	**	46.91	7200	–
G58	500	31 243	10	4	1	60	4014	**134**	7200	47.58	7200	–
G59	500	34 782	10	5	0	60	4173	**144**	**	/	7200	–
G60	500	31 279	10	7	1	60	4072	**141**	**	/	7200	–
G61	500	40 536	7	7	0	80	2839	**57**	**	/	7200	–
G62	500	30 953	8	3	1	80	3424	**376**	**	/	7200	–
G63	500	41 755	7	6	0	80	3623	**196**	**	/	7200	–
G64	500	30 822	8	6	1	80	3389	**183**	**	/	7200	–
G65	500	30 688	8	5	1	80	3298	**1158**	**	/	7200	–
G66	500	42 140	7	3	0	80	3662	**257**	**	/	7200	–
G67	500	30 830	8	2	1	80	3461	**988**	**	/	7200	–
G68	500	30 650	8	6	1	80	3287	**399**	**	/	7200	–
G69	500	41 938	7	4	0	80	3615	**382**	**	/	7200	–
G70	500	29 128	8	6	1	80	2678	**57**	**	/	7200	–
G71	500	49 051	6	3	0	100	3177	**870**	**	/	7200	–
G72	500	49 313	6	4	0	100	3257	**600**	**	/	7200	–
G73	500	38 892	6	4	0	100	3098	**411**	**	/	7200	–
G74	500	48 478	6	4	0	100	2992	**416**	**	/	7200	–
G75	500	38 642	5	5	2	100	3046	**784**	**	/	7200	–
G76	500	48 546	6	2	0	100	2899	**1608**	**	/	7200	–
G77	500	48 391	6	4	0	100	3311	**596**	**	/	7200	–
G78	500	48 912	6	4	0	100	3142	**322**	**	/	7200	–
G79	500	38 592	5	4	2	100	2981	**265**	**	/	7200	–
G80	500	39 048	5	5	2	100	3146	**229**	**	/	7200	–

Challenge ratio: CR which quantifies the cumulative difficulty posed by prime nodes, where $\mathbb{R}$ is the set of all prime nodes and $|\omega|$ is the size of each prime node in $\mathbb{R}$:

$$\mathrm{CR} = \frac{\Sigma_{\omega\in\mathbb{R}}|\omega|}{n}.$$

As shown in Figure 10, Figures 11 and 12 RR and CR exhibit an *inverse correlation*: when one is high, the other tends to be low. Instances with high reduction and low challenge ratios (*e.g.*, fpsol, inithx, mulsol, zeroin)

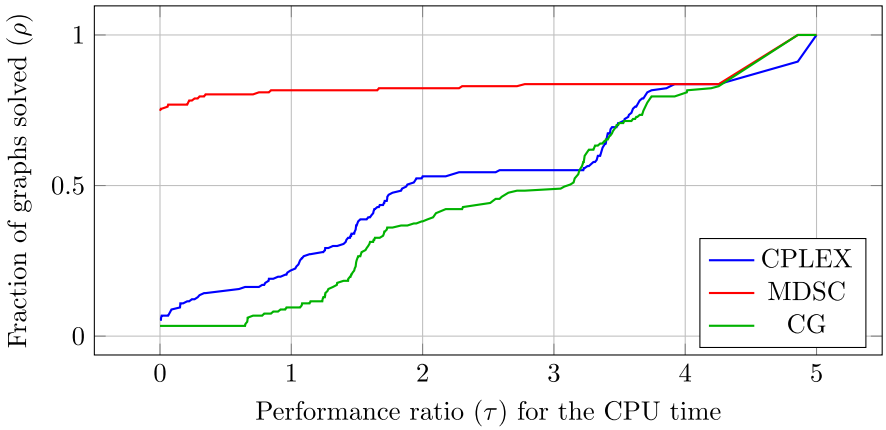

FIGURE 9. Performance profile plot.

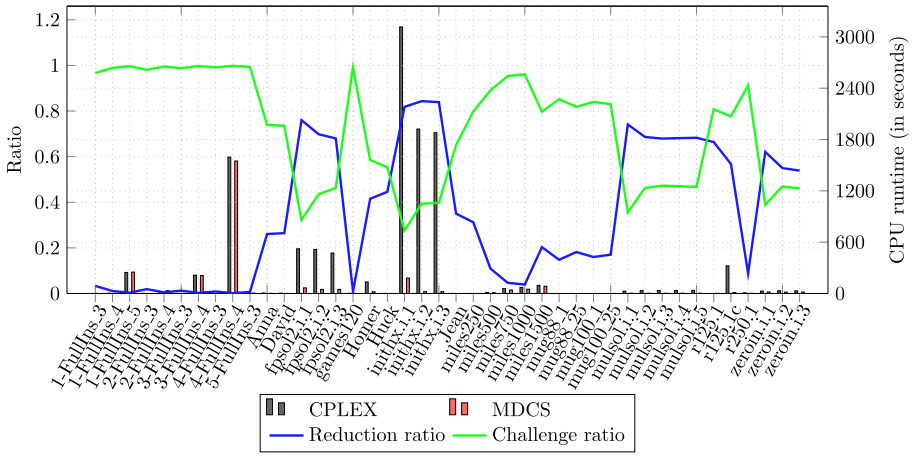

FIGURE 10. Relation between runtime and ratios for DIMACS and COLOR 2002–2004 benchmarks.

exhibit very low CPU runtime. Conversely, instances with large challenge ratios, especially those where large prime nodes dominate, are significantly more computationally demanding – sometimes reaching up to 1600 s. This confirms that while graph size and number of prime nodes matter, the size of prime nodes is the most critical factor affecting scalability.

Figure 12 further highlights this limitation by analyzing difficult instances from the DIMACS and COLOR 2002–2004 benchmarks. These graphs exhibit extremely low reduction ratios and high challenge ratios, confirming the infeasibility of MDSC under such structural conditions. In practice, MDSC remains efficient when the size of prime nodes do not exceed 100 vertices and generate a manageable number of uncontrolled solutions.

In Figure 13, we observe a particular case where both the RR and the CR are high. This reflects instances where modular decomposition effectively reduces the overall graph size, yet leaves a significant number of prime nodes whose total vertex count remains large. Despite the high reduction, the complexity introduced by these prime components results in elevated runtimes – sometimes comparable to or exceeding those of instances with low reduction and high challenge.

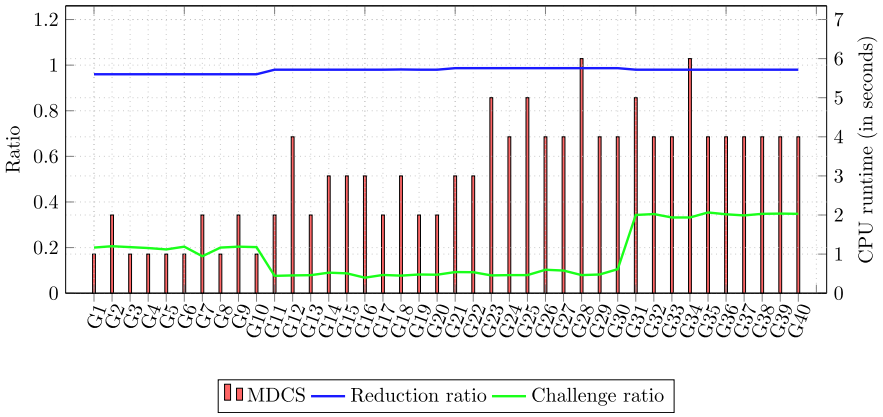

FIGURE 11. Relation between runtime and ratios for first group instances.

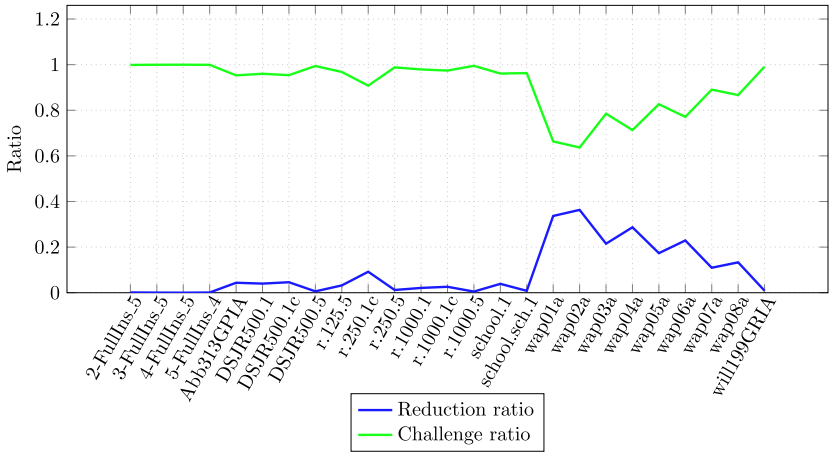

FIGURE 12. Relation between ratios for DIMACS and COLOR 2002–2004 benchmarks that are not solved by MDSC.

We also observe a critical performance shift as the RR decreases slightly while the CR remains high. Although the algorithm performs acceptably when RR = 0.92 or 0.88, a small drop to RR = 0.84 leads to a substantial increase in runtime. This effect becomes more pronounced at RR = 0.80, where runtimes exceed 1000 s for several instances. In such cases, the Challenge Ratio (CR) is close to 1, meaning that almost the entire graph remains inside a prime node, which explains why exact approaches struggle.

These results suggest that even small decreases in the reduction ratio, when combined with a consistently high challenge ratio, can lead to drastic growth in runtime – highlighting the sensitivity of MDSC to the unresolved complexity that remains after decomposition.

The instances marked by "*" (timeout) or "**" (memory exhaustion) correspond predominantly to graphs with extremely large prime nodes, for which the modular decomposition provides little simplification. Our MDSC Algorithm, by design, never encountered memory errors (contrary to CPLEX), but it could still reach the runtime limit when the prime nodes were too large to be solved within two hours. This confirms that the success or failure of MDSC is strongly tied to the structural features of the decomposition rather than the graph

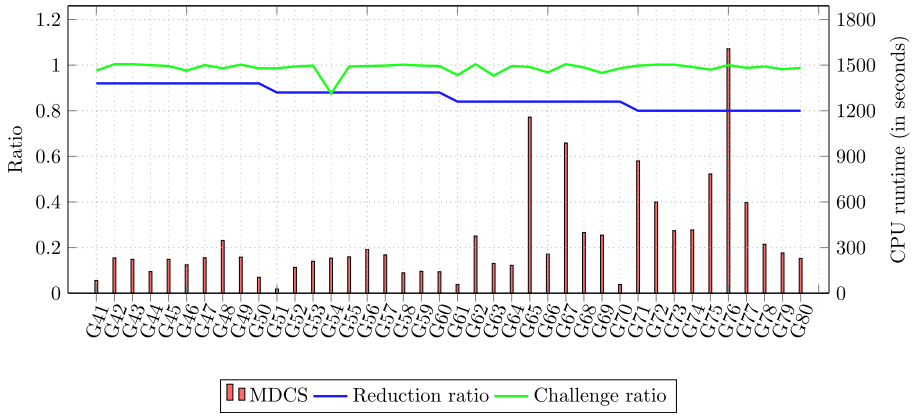

FIGURE 13. Relation between runtime and ratios for second group instances.

size or density alone. Based on these observations, we conclude that developing hybrid strategies focused on prime nodes offers a promising avenue for future investigation.

Also, unlike series or parallel nodes, prime nodes cannot be further decomposed into simpler subproblems. As a result, instances with very large prime components may lead to a substantial increase in computational effort, since the solver must explore them almost as a monolithic structure. One possible way to mitigate this limitation is through preprocessing techniques aimed at reducing prime components before optimization. Examples include:

- Graph reduction rules (removing dominated vertices, folding, etc.);
- Symmetry breaking techniques to avoid redundant exploration;
- Heuristic decompositions that approximate prime nodes with smaller tractable substructures;
- Hybrid approaches where MILP or SAT-based preprocessing simplifies the prime node before applying our exact method.

Exploring these preprocessing strategies constitutes an interesting direction for future work, as it may significantly improve scalability on graphs dominated by prime nodes.

Complexity considerations. From a theoretical standpoint, the proposed MDSC Algorithm is exponential in the worst case, since it is an exact approach for an NP-hard problem. However, its practical performance is largely governed by the decomposition structure of the input graph. In particular, the Reduction Ratio (RR) captures how much the modular decomposition decreases the effective problem size, while the Challenge Ratio (CR) highlights cases where large prime nodes remain. Instances with low RR and high CR correspond to the hardest cases, where the algorithm approaches its theoretical worst-case behavior, which explains the timeouts reported in our experiments. Conversely, for instances with favorable decomposition (high RR, low CR), the algorithm scales much better in practice than its theoretical bound would suggest.

7. Conclusion

In this paper, we proposed an exact algorithm, MDSC, for solving the Minimum Sum Coloring Problem by leveraging modular decomposition. The core idea is to recursively decompose the input graph into modules and solve smaller subproblems – allowing efficient resolution of large and partially decomposable graphs. Experimental results demonstrate that MDSC outperforms general-purpose solvers such as CPLEX and specialized methods like that of Delle Donne *et al.* [9] in terms of CPU runtime and number of optimally solved instances.

To better understand performance behavior, we introduced two structural indicators: the reduction ratio and the challenge ratio. These allowed us to interpret how decomposition quality and prime node complexity

influence runtime. We observed that MDSC performs exceptionally well when the reduction ratio is high and the challenge ratio is low. However, performance degrades significantly when large or numerous prime nodes dominate the decomposition – even when the overall reduction remains high. This highlights a key limitation: MDSC is sensitive to unresolved structural complexity concentrated in prime subgraphs.

From a practical standpoint, MSCP arises in real-world applications such as register allocation, task scheduling, frequency assignment, and VLSI layout optimization – where resource minimization is crucial. MDSC could serve as a subroutine within broader heuristics or be used to optimize specific graph components in these domains.

As future work, we plan to explore hybrid approaches that combine MDSC with local search or matheuristics. For example, exact resolution could be applied selectively to structured subgraphs (*e.g.*, series or parallel modules), while heuristic techniques manage more complex prime regions. Additionally, the modular decomposition framework could be extended to other combinatorial optimization problems, leveraging its versatility in handling structured input.

Acknowledgments

The authors express their gratitude to Professor Diego Delle Donne for his assistance in providing the software from his work, which was instrumental in conducting the comparative analysis, and thank the anonymous referees for their helpful comments and suggestions which contributed to the improvement of the contents of this paper.

Funding

Authors work was supported by the Direction Générale de la Recherche Scientifique et du Développement Technologique (DGRSDT).

Conflicts of interest

We declare that this work does not present any potential conflict of interest.

Data availability statement

There are three links to the different datasets used in this work, as follows:

DIMACS Challenge: http://dimacs.rutgers.edu/Challenges/;
COLOR-04 Challenge: http://mat.gsia.cmu.edu/COLOR04/;
Decomposable Generated Graphs: https://mega.nz/folder/JNdzELJC#Fagum6GRMilESijh6V7RBg.

References

[1] F. Bonomo and M. Valencia-Pabon, Minimum sum coloring of p4-sparse graphs. *Electron. Notes Discrete Math.* **35** (2009) 293–298.

[2] F. Bonomo and M. Valencia-Pabon, On the minimum sum coloring of p4-sparse graphs. *Graphs Comb.* **30** (2012) 303–314.

[3] F. Bonomo, G. Duran, A. Napoli and M. Valencia-Pabon, A one-to-one correspondence between potential solutions ofthe cluster deletion problem and the minimum sum coloring problem, and its application to p4-sparse graphs. *Inf. Process. Lett.* **115** (2015) 600–603.

[4] A. Borodin, I. Ivan, Y. Ye and B. Zimny, On sum coloring and sum multi-coloring for restricted families of graphs. *Theor. Comput. Sci.* **418** (2012) 1–13.

[5] Y. Boudabbous and P. Illel, Indecomposability graph and critical vertices of an indecomposable graph. *Discrete Math.* **309** (2009) 2839–2846.

[6] H. Bouziri and M. Jouini, A tabu search approach for the sum coloring problem. *Electron. Notes Discrete Math.* **36** (2010) 915–922.

[7] B.G. Dantzig, Linear Programming and Extensions. Princeton University Press (1963).

[8] I. DeHaan and Z. Friggstad, Approximate minimum sum colorings and maximum k-colorable subgraphs of chordal graphs, in Algorithms and Data Structures. 18th International Symposium, WADS 2023. Vol. 14079 of *Lecture Notes in Computer Science*, edited by P. Morin and S. Suri. Springer, New York, NY (2023) 627–639.

[9] D. Delle Donne, F. Furini, E. Malaguti and R.W. Calvo, A branch-and-price algorithm for the minimum sum coloring problem. *Discrete Appl. Math.* **303** (2021) 39–56.

[10] M.J. Dinneen, A. Mahasinghe and K. Liu, Finding the chromatic sums of graphs using a d-wave quantum computer. *J. Supercomput.* **75** (2019) 4811–4828.

[11] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002) 201–213.

[12] S.M. Douiri and S. Elbernoussi, New algorithm for the sum coloring problem. *Int. J. Contemp. Math. Sci.* **6** (2011) 453–463.

[13] S.M. Douiri and S. Elbernoussi, A new ant colony optimization algorithm for the lower bound of sum coloring problem. *J. Math. Modell. Algorithms* **11** (2012) 181–192.

[14] L. Epstein and A. Levin, On the performance guarantee of first fit for sum coloring. *J. Comput. Syst. Sci.* **99** (2019) 91–105.

[15] F. Furini, E. Malaguti, S. Martin and I.-C. Ternier, ILP models and column generation for the minimum sum coloring problem. *Electron. Notes Discrete Math.* **64** (2018) 215–224.

[16] M. Habib and C. Paul, A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.* **4** (2010) 41–59.

[17] M. Habib, R.M. McConnell, C. Paul and L. Viennot, Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.* **234** (2000) 59–84.

[18] H. Hajiabolhassan, M.L. Mehrabadiy and R. Tusserkani, Note minimal coloring and strength of graphs. *Discrete Math.* **215** (2000) 265–270.

[19] O. Harrabi and J. Chaouachi, Towards effective resolution approaches for solving the sum coloring problem. *J. Exper. Theor. Artif. Intell.* **32** (2020) 31–57.

[20] O. Harrabi and J. Chaouachi Siala, An effective parameter tuning for a bi-objective genetic algorithm to solve the sum coloring problem, in Soft Computing for Problem Solving. Vol. 816 of *Advances in Intelligent Systems and Computing*, edited by J.C. Bansal, K.N. Das, A. Nagar, K. Deep and A.K. Ojha. Springer, New York, NY (2017) 107–120.

[21] O. Harrabi and J. Chaouachi Siala, A new sum coloring-based integer linear programming for solving the university course timetabling problem, in 2020 International Multi-Conference on: "Organization of Knowledge and Advanced Technologies" (OCTA) (2020) 1–2.

[22] O. Harrabi and J. Chaouachi Siala, A novel unfeasible space exploring matheuristic proposal to solve the sum coloring problem, in Advances in Computational Collective Intelligence. 14th International Conference, ICCCI 2022. Vol. 1653 of *Communications in Computer and Information Science*, edited by C. Bădiă, J. Treur, D. Benslimane, B. Hnatkowska and M. Krótkiewicz. Springer, New York, NY (2022) 627–639.

[23] O. Harrabi, J. Chaouachi Siala and H. Bouziri, On solving exactly and approximately the sum coloring of graphs, in Proceedings. 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications. Hammamet, Tunisia. IEEE Computer Society, CPS (2017) 1043–1048.

[24] O. Harrabi, E. Fatnassi, H. Bouziri and J. Chaouachi Siala, A bi-objective memetic algorithm proposal for solving the minimum sum coloring problem, in GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM SIGEVO, Association for Computing Machinery, New York, NY (2017) 27–28.

[25] O. Harrabi, J. Chaouachi Siala and H. Bouziri, On integrating an iterated variable neighborhood search within a bi-objective genetic algorithm: sum coloring of graphs case application. *Electron. Notes Discrete Math.* **66** (2018) 55–62.

[26] Y. Jin and J.-K. Hao, Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Inf. Sci.* **352, 353** (2016) 15–34.

[27] Y. Jin, J.-K. Hao and J.-P. Hamiez, A memetic algorithm for the minimum sum coloring problem. *Comput. Oper. Res.* **43** (2014) 318–327.

[28] E. Kubicka and A.J. Schwenk, An introduction to chromatic sums, in Proceedings of the 17th AC Mannual computer science conference. USA, New York, NY (1989) 39–45.

[29] C. Lecat, C. Lucet and C.M. Li, Sum coloring: new upper bounds for the chromatic strength. *Discrete Appl. Math.* **233** (2017) 71–82.

[30] W. Lin, M. Xiao, Y. Zhou and Z. Guo, Computing lower bounds for minimum sum coloring and optimum cost chromatic partition. *Comput. Oper. Res.* **109** (2019) 263–272.

[31] M. Malafiejsk, K. Giaro, R. Janczewski and M. Kubale, Sum coloring of bipartite graphs with bounded degree. *Algorithmica* **40** (2004) 235–244.

[32] M. Minot, S.N. Ndiaye and C. Solnon, Évaluation d'approches complètes pour le problème de somme coloration, in Douzièmes Journées Francophones de Programmation par Contraintes, JFPC, edited by P. Vismara and E. Hebrard. SupAgro, LIRMM, Montpellier, Montepelier, France (2016) 53–62.

[33] A. Moukrim, K. Sghiouer, Y. Li and C. Lucet, Lower bounds for the minimal sum coloring problem. *Electron. Notes Discrete Math.* **36** (2010) 663–670.

[34] M. Mrad, O. Harrabi, J. Chaouachi Siala and A. Gharbi, A column generation-based lower bound for the minimum sum coloring problem. *IEEE Access* **8** (2020) 57891–57904.

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

[36] S.D. Nikolopoulos, L. Palios and C. Papadopoulos, A fully dynamic algorithm for the recognition of p4-sparse graphs. *Theor. Comput. Sci.* **439** (2012) 41–57.

[37] C. Paul, *Aspects Algorithmiques de la décomposition modulaire*. Habilitation à diriger des recherches, CNRS – LIRMM – Montpellier 2 University, Montpellier, France (2006). Available at https://theses.hal.science/tel-00112390/document.

[38] M. Rao, *Décompositions de graphes et algorithmes efficaces*. PhD thesis, Paul Verlaine University, Metz, France (2006). Available at https://hal.univ-lorraine.fr/tel-01752468.

[39] M.R. Salavatipour, On sum coloring of graphs. *Discrete Appl. Math.* **127** (2003) 477–488.

[40] A. Sen, A. Deng and S. Guha, On a graph partition problem with application to vlsi layout. *Inf. Process. Lett.* **43** (1992) 87–94.

[41] J. Spinard, P4-trees and substitution decomposition. *Discrete Appl. Math.* **39** (1992) 263–291.

[42] T. Szkaliczki, Routing with minimum wire length in the dogleg-free manhattan model is *np*-complete. *SIAM J. Comput.* **29** (1999) 274–287.

[43] M. Tedder, D. Corneil, M. Habib and C. Paul, Simpler linear-time modular decomposition via recursive factorizing permutations, in Automata, Languages and Programming, edited by L. Aceto, I. Damgård, L.A. Goldberg, M.M. Halldórsson, A. Ingólfsdóttir and I. Walukiewicz. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 634–645.

[44] Q. Wu and J.-K. Hao, An effective heuristic algorithm for sum coloring of graphs. *Comput. Oper. Res.* **39** (2012) 1593–1600.

[45] Q. Wu, Q. Zhou, Y. Jin and J.-K. Hao, Minimum sum coloring for large graphs with extraction andbackward expansion search. *Appl. Soft Comput.* **62** (2018) 1056–1065.

Appendix A. Summary of notation

Table A.1. Summary of mathematical symbols and notation used throughout the paper.

Symbol	Definition
$G = (V, E)$	An undirected simple graph with vertex set V and edge set E.
$n = \|V\|$	Number of vertices in the graph.
$m = \|E\|$	Number of edges in the graph.
$N(v)$	The neighborhood of vertex v.
$\deg(v)$	Degree of vertex v.
Δ	Maximum degree in graph G.
δ	Minimum degree in graph G.
$\bar{G}$	Complement of the graph G.
$G[M]$	Induced subgraph of G by the vertex subset $M \subseteq V$.
G/p	Quotient graph obtained from G by contracting each module in p into a vertex.
$c : V \to \{1, 2, \ldots, k\}$	A proper coloring function assigning colors to vertices.
S_l	Set of vertices assigned color l under coloring c; *i.e.*, an independent set.
$f(c)$	Cost of a coloring c, defined as $\sum_{i=1}^{n} c(v_i) = \sum_{l=1}^{\gamma} l \cdot \|S_l\|$.
$\Sigma(G)$	The minimum sum of a proper coloring of G.
$s(G)$	The minimum number of colors in a coloring that achieves $\Sigma(G)$.
X^*	The optimal solution (coloring) returned by the MDSC algorithm.
ΣX	Cost of a coloring solution X.
X_j^i	The j^{th} uncontrolled solution equal to $\left\{S_1^i, S_2^i, \cdots, S_{\|X_j^i\|}^i\right\}$.
$\mathbb{F}$	The set of uncontrolled solutions with different sequences.
γ_i	The number of uncontrolled solutions for $G[M_i]$.
$\Im^c(G[M_i])$	A combination of independent sets formed by combining one selected uncontrolled solutions from each child $t, t \in \{1, \cdots, \mathscr{H}\}$ of M_i.
$\Im(G[M_i])$	The set of all possible combinations of $\Im^c(G[M_i])$ for $c \in \{1, \cdots, \prod_{t=1}^{\mathscr{H}} \gamma_t\}$.
$\pi = \{m_1, \ldots, m_h\}$	A partition of V into subsets.
$\mathscr{P} = \{M_1, \ldots, M_h\}$	The set of maximal strong modules of G; forms the modular partition.
$\mathrm{MD}(G)$	Modular decomposition tree of graph G.
η	A node in $\mathrm{MD}(G)$, corresponding to a subgraph/module.
$\deg(\eta)$	Number of children of node η in $\mathrm{MD}(G)$.
Ω_n	Set of all ordered integer partitions (sequences) of size n.
Ω_n^k	Set of ordered partitions of n into exactly k parts.
d_k^n	Dominant sequence: $(n - k + 1, 1, ..., 1)$.
b_k^n	Balanced sequence with elements $\lfloor \frac{n}{k} \rfloor$ and $\lceil \frac{n}{k} \rceil$ dominated by all elements in Ω_n^k.
Σp	Cost of sequence p, defined as $\sum_{i=1}^{k} i \cdot p[i]$.
$\widehat{\mathrm{UB}}$	Upper bound on the number of colors in a valid coloring.
$\widehat{\mathrm{LB}}$	Lower bound on the number of colors in a valid coloring.
RR	Reduction ratio: $\mathrm{RR} = \frac{n - \|R\|}{n}$, where $\|R\|$ is the size of the reduced graph.
CR	Challenge ratio: $\mathrm{CR} = \frac{\sum_{\omega \in \mathbb{R}} \|\omega\|}{n}$, where $\mathbb{R}$ is the set of prime nodes.