A study on the Graph Coloring Problem - An attempt to achieve faster & better IP solution quality

SPRING 2020 - ISEN 668 COURSE PROJECT

Surya Narayan
UIN 928001157
narayan@tamu.edu
MS Industrial Engg
Texas A&M University

May 8, 2020

1 Introduction

A proper coloring of a graph $G(V, E)$ is an assignment of colors to the vertices of G so that no two adjacent vertices get the same color. Equivalently, a proper vertex coloring is a mapping f : $V \mapsto \{1, 2, ..., n\}$ such that for any $(i, j) \in E$, f (i) $\neq$ f (j). The minimum possible number of colors that are used to color the graph is also called as the Chromatic number of the graph. It can also be seen as a partition of the vertices into stable sets (also called as independent sets). Thus the chromatic number can also be seen as the minimum number of stable sets forming a vertex partition. We refer to the optimization problem of finding a minimum vertex coloring of a graph as the Graph Coloring Problem (GCP).

The GCP is a well-known NP-hard problem that has an important role in several theoretical fields and applicability in a wide variety of practical settings. It has been extensively studied from different perspectives by mathematicians, computer scientists, operations researchers, engineers, biologists and social scientists. It has applications in fields like scheduling, radio frequency allocation, timetabling, sequencing etc. Authors have studied the polyhedral structure of the problem and have applied various exact solution techniques like Branch-and-Cut, Column-Generation, Dynamic Programming, etc.

Initial Motivation: Recently, there has been a significant interest in using Machine Learning (ML) techniques with or without conventional IP solvers to solve Combinatorial Optimization/Integer Programming problems. Some of the tried techniques are - ML heuristics for variable selection, ML heuristics for approximation algorithms, learning to generate valid inequalities/cuts, Decision diagrams + Reinforcement learning as an optimization framework and Neural Convolutional Graph networks. We hope to revisit, explore and attack the problem with these latest techniques.

A note: At the beginning of the project, I was working with the Naveen Sakamuri. Later, we decided to fork the project and work separately. Hence, the sections until "NP-hardness of GCP" could be similar in our reports. The contribution is equal until these sections.

Literature Review: (TO ADD MORE CONTENT) Several authors have proposed exact algorithms for GCP, based on integer programming formulations.[Coll, Marenco, Mendez Dıaz Zabala] proposed a formulation with a polynomial number of variables corresponding to color assignments to vertices. The linear relaxation of this formulation is extremely weak. However, they identified powerful families of valid inequalities which significantly reduces the integrality gap. They design a cutting plane algorithm and test the performance of different subset of valid inequalities. Later [MZ 2008], they develop a branch-and-cut algorithm using the results from [MZ 2006]. Mehrotra and Trick (1996) proposed a formulation using variables associated to maximal

stable sets of the graph, which was exponential in size. Their solution technique was a Column-Generation scheme, where the master problem is a set partitioning problem and the pricing sub-problem is a maximum weighted stable set problem. Also, an implementation of a Branch-Price-and-Cut approach is reported in Hansen et al. (2009), where a family of valid inequalities that do not break the structure of the pricing sub-problem is introduced. Also due to the combinatorial nature of the GCP which can be exploited by constraint programming (CP), an exact solution has been proposed by (Stefano Gualandi, Federico Malucelli) using the hybrid approach of CP based Column-Generation, where the pricing problem is solved using CP, where they used the formulation used in (Mehrotra and Trick).

2 Integer programming (IP) formulations

Problem Instance:
Let $G = (V, E)$ be a graph and $V = \{1, ..., n\}$ be the set of vertices and E be the set of edges.

The GCP can then be formulated as follows:

2.1 Formulation 1 $(GC(St))$

Decision variables:

$$x_{ik} = \begin{cases} 1, & \text{if color k is assigned to node i} \\ 0, & \text{otherwise} \end{cases}$$

$$w_k = \begin{cases} 1, & \text{if color k is used} \\ 0, & \text{otherwise} \end{cases}$$

$$minimize \sum_{k=1}^{n} w_k \tag{1a}$$

$$subject\ to \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i \in V \tag{1b}$$

$$x_{ik} + x_{jk} \leq 1, \qquad \forall (i, j) \in E \tag{1c}$$

$$w_k \geq x_{ik}, \qquad \forall i \in V,\ k = 1, ..., n \tag{1d}$$

$$w_k, x_{ik} \in \{0, 1\}, \qquad \forall i \in V,\ k = 1, ..., n \tag{1e}$$

The objective function (1a) drives the search to minimize the number of colors used. Constraints (1b) ensure only one color is assigned to each node. Constraints (1c) ensure that no two adjacent nodes get the same color. Constraints (1d) ensure the activation of color k. Constraints (1e) are the integrality conditions.

2.2 Formulation 2 - Strengthening GC(St)

GC(St) is a symmetric formulation, because all color permutations yield feasible solutions with the same objective function value. To avoid symmetry in this model, [MZ 2006] propose the following model.

$$minimize \sum_{k=1}^{n} w_k \tag{2a}$$

$$subject\ to \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i \in V \tag{2b}$$

$$x_{ik} + x_{jk} \leq w_k, \qquad \forall (i,\ j) \in E \tag{2c}$$

$$w_j \leq \sum_{i=1}^{n} x_{ij} \qquad \forall 1 \leq j \leq n \tag{2d}$$

$$w_j \geq w_{j+1} \qquad \forall 1 \leq j \leq n-1 \tag{2e}$$

$$w_k, x_{ik} \in \{0,1\} \qquad \forall i \in V,\ k = 1, ..., n \tag{2f}$$

2.3 Formulation 3 - Set Partitioning Formulation $(GC(Part))$

It can be seen that each color class of the GCP corresponds to an stable set of G. With this we formulate the following formulation: Let $\mathcal{S}$ be the set of all stable sets of G, and $x_S = 1$ if the stable set S corresponds to a color class.

Decision variable:

$$x_S = \begin{cases} 1, & \text{if stable set } S \text{ is included in the partition} \\ 0, & \text{otherwise} \end{cases}$$

$$minimize \sum_{S \in \mathcal{S}} x_S \tag{3a}$$

$$subject\ to \sum_{S \in \mathcal{S}: v \in S} x_S = 1, \qquad \forall v \in V \tag{3b}$$

$$x_S \in \{0,1\}, \qquad \forall S \in \mathcal{S} \tag{3c}$$

The objective function (2a) chooses to minimize the number of stable sets. Constraints (2b) ensure that each node is exactly in one stable set.

2.4 Formulation 4 - Set Covering Formulation $(GC(Cov))$

From the above formulation, one can see that we don't necessarily have to partition V, but only covering it with a minimum number of stable sets is sufficient for the GCP. This implies that the stable sets have to be maximal to minimize the number of stable sets needed to cover. Let $\mathcal{S}_{max}$ be the family of maximal stable sets of G.

Decision variable:

$$x_S = \begin{cases} 1, & \text{if stable set } S \text{ is included in the partition} \\ 0, & \text{otherwise} \end{cases}$$

$$minimize \sum_{S \in \mathcal{S}_{max}} x_S \tag{4a}$$

$$subject\ to \sum_{S \in \mathcal{S}_{max}: v \in S} x_S \geq 1, \qquad \forall v \in V \tag{4b}$$

$$x_S \in \{0,1\}, \qquad \forall S \in \mathcal{S}_{max} \tag{4c}$$

The objective function (3a) chooses to minimize the number of maximal stable sets. Constraints (3b) ensure that for each vertex v, there is at-least 1 maximal stable set that covers it.

2.5 Formulation 5 - Reducing formulation size of GC(St)

Let $\delta(v) = |N(v)|$. By combining the edge constraints $x_{vj} + x_{kj} \leq w_j \ \forall k \in N(v)$ we obtain the valid Neighborhood inequality. The edge constraints are replaced by these Neighborhood inequalities as follows:

$$minimize \sum_{k=1}^{n} w_k \tag{5a}$$

$$subject\ to \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i \in V \tag{5b}$$

$$\sum_{k \in N(v)} x_{kj} + \delta(v) x_{vj} \leq \delta(v) w_j, \qquad \forall v \in V,\ \forall 1 \leq j \leq n \tag{5c}$$

$$w_j \leq \sum_{i=1}^{n} x_{ij} \qquad \forall 1 \leq j \leq n \tag{5d}$$

$$w_j \geq w_{j+1} \qquad \forall 1 \leq j \leq n-1 \tag{5e}$$

$$w_k, x_{ik} \in \{0,1\} \qquad \forall i \in V,\ k = 1, ..., n \tag{5f}$$

3 Comparison of Formulations

Though the formulation $GC(St)$ is compact, the linear relaxation provides an inferior lower bound on the chromatic number $\chi(G)$. This can be verified by considering the fractional solution given by $x_{ik} = \frac{1}{2}, y_k = 1$ for all i if k = 1,2 and $x_{ik} = y_k = 0$ for all i if $k \geq 3$. This solution satisfies constraints (1c) irrespective the instance of the Graph, so the optimal value z^{*LP} is of no use. [HLS05] has shown that the LP relaxation bounds obtained through $GC(Part)$ and $GC(Cov)$ are of same quality and superior to $GC(St)$. Note that the formulations $GC(Part)$ and $GC(Cov)$ are exponential in size.

3.1 One-to-One correspondence of feasible solutions

The formulation $GC(St)$ is a straight-forward representation of the GCP, in the sense that no two adjacents nodes in G, can have the same color. Now, lets prove that $GC(Part)$ is correct. We show this by proving every feasible solution of the GCP is a feasible solution of the $GC(Part)$ and vice-versa,

Every feasible solution of GCP is a feasible solution of $GC(Part)$ **Proof** Suppose a solution of $GC(Part)$ has 2 adjacent nodes of the same color, which is infeasible in $GC(St)$. Then, By definition of stable set, no 2 adjacent nodes cannot be in the same stable set. In the $GC(Part)$ formulation, every stable set (no edges in the set) is a color class. This contradicts the statement that a solution of $GC(Part)$ has 2 adjacent nodes of the same color/are in the same stable set.

Every feasible solution of $GC(Part)$ is a feasible solution of GCP **Proof** Every feasible solution of the $GC(Part)$ is a partition of the vertices into stable sets. And each stable set corresponds to a color class. Thus, no two nodes in a stable set can have the same color.

Since we have shown that $GC(Part)$ which includes all stable sets in G, has a one-to-one correspondence with the GCP, it automatically holds true for $GC(Cov)$ also, as $S_{max} \subseteq \mathcal{S}$.

4 NP-hardness of GCP

The decision version of the GCP is the k-COLORING problem of finding whether there exists a proper coloring of the given graph, such that, the chromatic number, W(G) ¡= k. Given a candi-

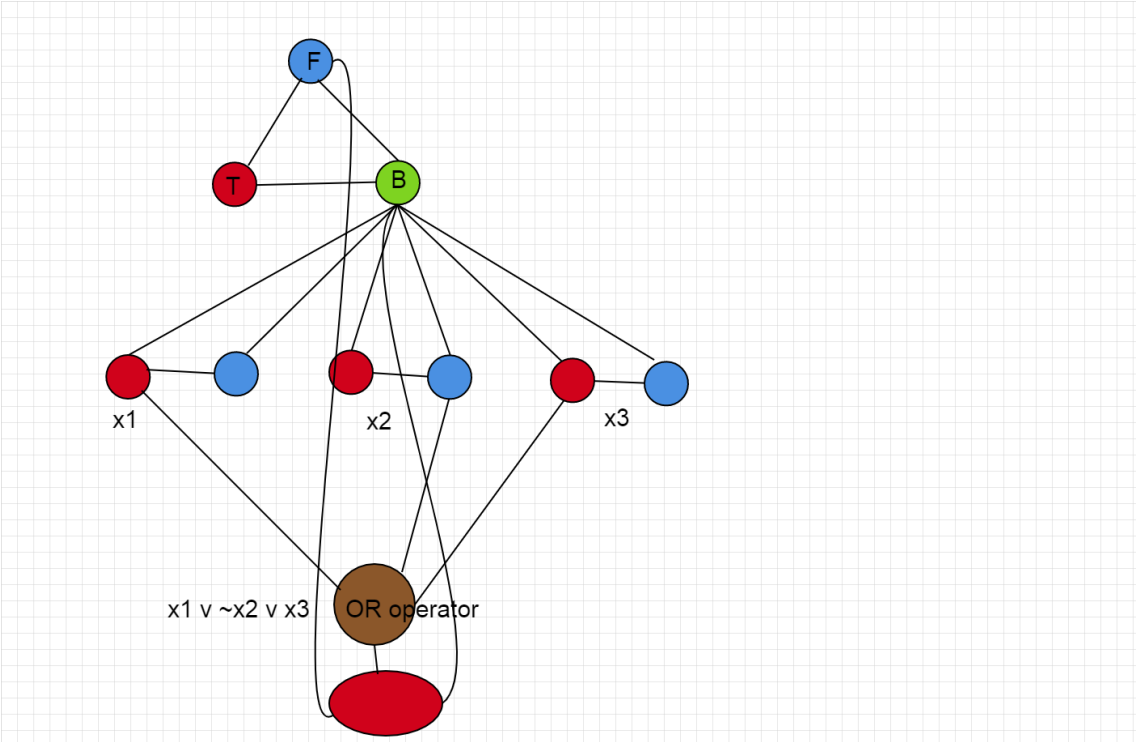
date solution, one can easily verify, in polynomial time, the feasibility of the solution by examining the vertex colors of each edge and count the number of distinct colors. This shows that the k-COLORING problem is in NP.

To establish NP-Hardness of the GCP, it is sufficient to show that the k-COLORING problem is NP-Complete (NPC), as NPC subset NP-Hard. To show that k-COLORING is NPC, it is sufficient to show that 3-COLORING is NPC, as 3-COLORING can be reduced in polynomial time to k-COLORING, by simply setting k = 3 in the k-COLORING problem.

3-SAT to 3-COLORING (ROUGH DRAFT - TO REFINE)

Karp 1972 [] proved that 3-COLORING problem is NPC, by showing a polynomial-time reduction from 3-SAT. (citation needed) 3-SAT is a well-known NP-Complete problem. Given a 3-SAT instance S, we construct an instance of 3-COLOR that is 3-colorable iff S is satisfiable.

Let S be a 3-SAT formula with n variables x1,..., xn and m clauses C1,...,Cm. We will create a graph G as follows: We have 3 vertices as base(B), true(T) , false(F) and all are connected. For all the variables x1. . . .xn we have vertices and their respective complement vertices connected and connect both of them to the base. We represent each clause in the 3-SAT problem by a logical-OR gate aka gadget. Each literal of the clause is connected to the inputs of the gadget. The output of the gadget is connected to the base(B) and false(F) vertices. Let the B, T, F be three different colours for base, true and false vertices respectively. The output of clause is thus either colour T or F. Also each of the vertex take either colour T or F.



A solution of the 3-COLORING on this constructed instance, thus, corresponds to a True or False assignment to the vertices $x_1, \ldots, x_n$.

Suppose, S is satisfiable, if xi is assigned True, the respective vertex color associated with xi is T and xi is F. Since S is satisfiable, every clause Ci = (l1l2l3) must be satisfiable, i.e. at least one of l1, l2, l3 is set to True. So the output vertex is of colour T. And because the output node is connected to the False and Base vertices which have respective vertices colours F and B, this is a proper 3-coloring. Conversely, suppose G is 3-colorable. We construct an assignment of the literals of S by setting xi to True if its vertex is coloured T and vice versa. Now suppose if it is not a satisfying assignment to S, then this means there exists at least one clause Ci = (l1 l2 l3) that was not satisfiable. That is, all of l1, l2, l3 were set to False. But if this is the case, then the output

node of clause Ci is false so its respective colour is false, but this output node is connected to the False vertex coloured F; thus contradicting the 3-colorability of G. Hence, this establishes the NP-completeness of 3-COLORING, thus k-COLORING is also NPC. For graphs with vertex-degree no greater than 3, the GCP is polynomial-time solvable [Garey Johnson, Computers Intractibility].

5 Theoretical Analysis

5.1 Bounds on the Chromatic Number

5.2 Lower Bounds

Let $\omega(G)$ denote the Clique number of G - number of vertices contained in the largest clique in G. Since $\omega(G)$ different colours will be needed to colour this clique, we deduce that $\chi(G) \geq \omega(G)$. From another perspective, we can also consider the independent sets of a graph. Let $\alpha(G)$ denote the independence number of a graph G, defined as the number of vertices contained in the largest independent set in G . In this case, $\chi(G)$ must be at least $\lceil n/\alpha(G) \rceil$ since to be less than this value would imply the existence of an independent set larger than $\alpha(G)$.

Combining these two,

$$\chi(G) \geq max\{\omega(G), \lceil n/\alpha(G) \rceil\}$$

.

Though, finding $\omega(G)$ and $\alpha(G)$ are themselves NP-Hard, we can use heuristics on these problems to get lower bounds. However, the quality of the bound could be very weak in certain cases like triangle-free graphs.

5.3 Lovasz-theta number and Sandwich theorem

Suppose, the chromatic number of the complement graph G is $\chi(\bar{G})$. Then, the famous sandwich theorem [2] states,

$$\alpha(G) \leq \theta(G) \leq \chi(\bar{G})$$

Accurate numerical approximations to this number can be computed in polynomial time by semidefinite programming and the ellipsoid method. The Lovasz-theta bound is better than the clique number bound.

5.4 Upper Bounds

[3] Upper bounds on the chromatic number of are often derived by considering the degrees of vertices in a graph. For instance, when a graph has a high density (that is, a high proportion of vertex pairs that are neighbouring), often a larger number of colors will be needed because a greater proportion of the vertex pairs will need to be separated into different colour classes.

Theorem 1 : *Let G be a connected graph with maximal degree $\Delta(G)$ (that is, $\Delta(G) = max\{deg(v) : v \in V\}$. Then $\chi(G) \leq \Delta(G) + 1$.*

[3] Another bound concerning vertex degrees can be calculated by examining all of a graph's subgraphs and identifying the minimal degree in each case, and then taking the maximum of these. For practical purposes this might be less useful than Theorem 1 for computing bounds quickly since the total number of subgraphs to consider might be prohibitively large. However, the following result still has its uses, particularly when it comes to colouring planar graphs and graphs representing circuit boards.

Theorem 2 : *Given a graph G, suppose that in every subgraph G' of G there is a vertex with degree less than or equal to δ. Then $\chi(G) \leq \delta + 1$.*

For proofs of these theorems, see [3].

6 Valid Inequalities & Facets

From [6],
We'll refer to formulation 2 as CP.

Proposition 1. The dimension of formulation 2, is $n^2 - \chi(G) - 1$.

Proposition 2. Let $i_0 \in V$, the following holds:
(a) Every non-negativity constraint $x_{i0n}0$ defines a facet of CP.
(b) For $j_0 = 1, ...n-1$, if $G - i_0$ is not a clique then $x_{oj_0} \geq 0$ defines a facet of formulation 2.

Proposition 3. Let j_0 be any color such that $\chi(G) \leq j_0 \leq n-2$, then the constraint $w_{j_0} \geq w_{j_{0+1}}$, then this is a facet of CP.

Proposition 4. The inequality $w_{j_0} \leq \sum_{i \in V} x_{ij_0}$ defines a facet of CP for all $j_0 = 1, ..., n-1$ iff there is some $(\chi(G))$-coloring lying on the face.

Proposition 5. Let K be a maximal clique. The Clique inequality

$$\sum_{v \in K} x_{vj_0} - w_{j_0} \leq 0$$

is a facet-defining inequality of CP.

Proposition 6. Let $v \in V$, $N(V)$ the neighborhood of of v and $delta(v) = |N(v)|$. By combining $x_{vj} + x_{kj} \leq w_j$ for all $k \in N(v)$ we obtain the Neighborhood Inequality. The Neighborhood inequality is valid for CP.

$$\sum_{k \in N(v)} x_{kj} + \delta(v) x_{vj} \leq \delta(v) w_j$$

.

Proposition 7. If r is the size of a maximum independent set in $N(v)$, no more than r vertices can be colored with the same color, then the above inequality can be strengthened and

$$\sum_{k \in N(v)} x_{kj} + r x_{vj} \leq r w_j$$

. is a valid inequality stronger than the previous one. This is not a facet-defining inequality but it becomes very useful to improve the LP-relaxation. The original model has mn edge constraints and this size is difficult to handle for large and dense graphs. These constraints can be replaced by the above inequalities in the original model. Though this relaxes the polytope, the authors in [6] report that computational experience shows it works better than the orginal formulation.

We'll attempt to verify this result.

7 Algorithmic Solutions

7.1 Heuristics

7.2 Sequential Colorings

We first consider a general type of coloring procedure which can be specialized in many ways.

a) Determine an order $O : v_1 < v_2 < ... < v_n$ of the nodes of G.
b) Color node v_1 with color 1; generally if $v_1, ..., v_{i-1}$ have been colored, give node v_i the smallest color which has not been used on any node $v_j (j < i)$ linked to v_i.

This general procedure is called a sequential coloring procedure based on order O (or shortly SC(O)). Needless to say that the coloring obtained with an SC will depend on the order chosen in a. The use of such procedure is justified by the following:

Proposition 1: *For any graph G, there exists an order O of the nodes for which SC(O) produces a coloring in $\chi(G)$ colors.*

This can be easily seen by taking any coloring $S_1, ...S_k$ of G in $\chi(G) = k$ colors and ordering the nodes in such a way that whenever $i < j$ the nodes in S_i come in the order before the nodes in S_j.

More generally we shall say that a (heuristic) procedure $H(p)$ characterized by a family p of parameters is *acceptable* for solving a problem P if the set of solutions $S(H)$ obtained by varying the parameters in all possible ways contains some optimal solution of P.

In the case of $SC(O)$ we have $p = O$; for solving the coloring problem P, the method $SC(O)$ is acceptable from Proposition 1.

Proposition 2: *For a complete k-partite G, SC(O) gives a coloring in $\chi(G)$ colors with any order O.*

Now, we define some techniques based on sequential coloring.

7.2.1 DSATUR - a sequential coloring technique

In order to develop heuristic procedures which are hopefully efficient for general graphs, we may try to extend some SC methods which are exact for a simplified problem type . It is reasonable to concentrate on the bipartite graph coloring problem since for such graphs finding the chromatic number is easy.

A simple labeling technique for checking whether a graph is bipartite will give us directly a bi-coloring:

1) label an arbitrary node x with +; x is then labelled, all remaining nodes are unlabelled
2) apply as long as possible the following rule; if a node is labelled with + (resp. -) then label its neighbors with - (resp. +).

If all nodes receive a unique label, the graph is bipartite, otherwise it contains an odd cycle.

The nodes will be labelled one after the other and if the graph is bipartite, there will be some Order O in which the nodes are labelled. The procedure can be extended to an $SC(O)$ procedure for an arbitrary graph as follows: at each step i of the coloring procedure we define for each node x the *degree of saturation* $ds_i(x)$ as the number of different colors already used for the neighbors of node x when nodes $v_1, v_2, ..., v_{i-1}$ have been colored. Initially (i.e, when no node is colored yet we take $ds_0(x) = 0$.

Consider now an order $O : v_1 < v_2 < ... < v_n$ and apply the $SC(O)$ procedure. We will get a coloring of G with at most

$$DS(O) = 1 + max_{1 \leq i \leq n} ds_i(v_i)$$

colors. $DS(O)$ depends on the order O. For getting a value of $DS(O)$ which is small, a reasonable procedure consists in choosing at each step i of the $SC(O$ procedure a node v_i such that

$$ds_i(v_i) = max_{xuncolored} ds_i(x)$$

. This procedure differs from other $SC(O)$ procedures by the fact that the order is constructed dynamically during the coloring process.

Proposition 3: *The SC(O) procedure DSATUR is exact for bipartite, wheel and cactus graphs.*

7.3 Exact Algorithms

7.4 Branch and Cut Algorithm

Two exact methods were developed. In both the methods, Formulation 2 (Standard Edge formulation, strengthened by adding constraints to remove symmetry in the model) was used.

7.5 Method 1

We began trying to reproduce the results from [5]. However, we ended up with some modifications and obtained better results in the instances we ran. We first tried to perform the following preprocessing techniques proposed in the paper and were implemented in code. However, they were not used for the results reported.

7.6 Preprocessing

1. Procedure 1: Processing vertices adjacent to clique K
Let $v \in V \setminus K$ and $w \in K$ such that $w \notin N(v)$. If $N(v) \subset N(w)$, then a coloring of $G - v$ can be extended to a coloring of G by assigning to v the same color of w. Based on this property, we consider vertices in increasing value of their edge degrees and use a sequential procedure to identify and remove vertices meeting the condition above.

2. Procedure 2: Processing vertices by degree Let $v \in V \setminus K$ be such that the degree of v is less than the $n_cli - 1$ (best clique size found minus 1). This also guarantees that $\chi(G[V \setminus v]) = \chi(G)$. These vertices are recursively deleted until all vertices left have degrees greater than $n_cli - 2$.

7.7 Improving the LP relaxation

After the reductions above, the proposed model involves $m\bar{\chi} + n$ constraints. Since, this gets too large for even moderately sized graphs, we replace the edge constraints by the following Neighborhood Inequalities. Note that, in [5], they used a weaker version of the Neighborhood inequalities, however, we do not modify the Neighborhood inequalities.

TODO from cut plane zabala

7.8 Finding initial Bounds

We use the DSATUR heuristic to find a feasible solution of k-colors. A slight alteration is made to the DSATUR heuristic, where we try and find a maximal independent set as large as possible within 60 seconds. We then remove the independent set vertices from the graph and ask DSATUR for a feasible coloring. The independent set vertices are then manually given a color. We do so to reduce the number of colors. We then set k as the Upper Bound on $\chi(G)$. We then find a maximal clique and set its cardinality (n_cli) as the lower bound on $\chi(G)$.

The vertices in the found maximal clique are given fixed colors. We also find the Lovasz-Theta number $\theta(\bar{G})$ on the complement graph to obtain a stronger lower bound on $\chi(G)$. We add the following constraint to impose the $\theta(\bar{G})$ lower bound,

$$\sum_{k=1}^{UB} w_k \geq \lceil \theta(\bar{G}) \rceil$$

The feasible solution found by DSATUR heuristic is fed to the solver through MIP_start (warm start settings). This is done so that the solver could get an incumbent solution at the beginning and prune faster.

7.9 Method 2

We initially explored the idea of a Branch-and-Cut algorithm, focusing on adding User cuts and checking the effectiveness of the cuts. Clique inequalities were found to be the most effective in [6] for the above formulation (however with a weaker version of neighborhood inequalities). We then

set out to survey various heuristic separation procedures for finding violated clique inequalities in the literature. In [4], an efficient separation heuristic is given to find violated clique inequalities. The heuristic unlike Hoffman and Padberg [1] procedures, is not derived from an exact framework to solve the maximum clique problem. Differently, the search for violated cliques is guided by the purpose of building a collection of violated cliques that cover all the edges of the graph.

We implemented the procedure and tried adding User cuts. However, we didn't get expected results. The number of nodes explored in the branch-and-bound tree was higher than regular branch-and-bound. Also, the GUROBI 9.0 solver doesn't allow us to control the number of user cuts being added at the tree nodes. We then realized, more flexible solvers/frameworks like COIN-OR's CBC and SYMPHONY are to be tried to explore in this direction, which needs somewhat a more involved work with insufficiently documented COIN-OR libraries, and hence stopped pursuing this direction. However, the heuristic procedure was indeed working efficiently as stated in [4].

8 Computational Experiments

We ran our experiments on DELL G7 laptop with Intel i7 (12 threads) and 16 GB RAM. We used GUROBI 9.0 as the IP solver. Python 3.7 was used as the programming language. We also used C++ initially, only to realize later it was hard to quickly prototype code and dropped it. Model building was done using GurobiPy, an API to the Gurobi solver.

We used the following settings for Method 1:
1. Heurstics = DEFAULT
2. Node Relaxation Heuristics = DEFAULT
3. Gurobi Cuts = DEFAULT
4. Pre-crush = OFF (A parameter that disables some pre-solve heuristics to prevent original mapping of the decision variables, we used it initially when adding user cuts. We left it turned off.)
5. User-cuts = OFF (No callbacks were used)

The formulation was used as described in Method 1. We didn't however use the pre-processing technique described. For the DSATUR heuristic and finding maximal cliques, we used the algorithms from Python networkx package. We set a maximum time for finding cliques DSATUR heuristic for 60 seconds each. For Lovasz-Theta computation, we used Dan Stahlke's Python implementation found in [7]. We set a time limit of 10 minutes for the solver.

We compare it with the pure Branch-and-Cut technique with default settings of the solver.

All time units are in seconds. We mark NA when we run out of system memory.

TABLE 1 - Results from Method 1
1. $\theta(\bar{G})$ - Lovasz-Theta number on the complement graph
2. LB - Lower bound (n_cli)
3. UB - Upper bound from DSATUR heuristic
4. RRObj - Root Relaxation Objective
5. RR_t - Root Relaxation time
6. nodes - Number of explored by the Branch-and-bound tree
7. $Gap\%$ - MIP Integrality Gap
8. Elasped - Total time elapsed

Instance	$\theta(G)$	LB	UB	RRObj	RR_t	nodes	Gap%	Elapsed
DSJC250.9	55.5	33	85	85	1.36	1	0	20
DSJC500.5	20.542	13	66	66	3	0	0	60
DSJR500.1c	83.74	59	90	90	33.7	0	0	33.7
DSJC1000.1	8.307	6	27	25	6.15	0	0	33.7
DSJC1000.5	31.89	14	116	NA	NA	NA	100	600

TABLE 2 - Results from pure Branch-and-Cut (Formulation + Neighborhood Inequalities)
1. presolved - Number of rows and columns ("-" removed, "+" added)
2. BCnodes - Number of Tree nodes explored
3. BC_RRObj - Root Relaxation Objective
4. BC_GAP % - MIP Integrality Gap
5. BC_Elapsed - Total Elapsed time

Instance	presolved	BCnodes	BC_RRObj	BC_Gap%	BC_Elapsed
DSJC250.9	(0,0)	NA	2.84	100	600
DSJC500.5	(0,0)	NA	NA	100	485
DSJR500.1c	(+500, +500)	NA	NA	100	450
DSJC1000.1	(0,0)	NA	NA	100	0
DSJC1000.5	(0,0)	NA	NA	100	600

8.1 Analysis

The Method 1 we proposed, clearly outperforms the naive Branch-and-cut method of the solver. The formulation when used in conjunction with heuristic approaches works fairly well for the instances we experimented on. 4 out of 5 instances were solved within a minute in the root relaxation phase of the solver. Some of these instances were considered hard in the literature (however, they were old).
All the instances were solved to optimality. With DSJC1000.5 we encountered RAM "Out-of-memory" issue. This is largely due to Python's memory management more than the problem size.

With the pure Branch-and-Cut approach, the solver struggles to solve even the root relaxation. Since the edge constraints were dropped and replaced by Neighborhood inequalities, which reduces the problem size, to see an improvement in CPU times, we expected the solver would be able to solve it efficiently, especially with the cuts parameter in its default settings, to its advantage. It is not able to presolve the rows and columns as it did with our Method 1. We feel that the variable fixing part of Method 1 is largely responsible for the presolve-ability of the solver. Python ran out of memory for all the instances. It is interesting to see Presolve added 500 rows and columns to DSJR500.1c.

9 Conclusion

We explored the Polyhedral structure of the GCP and understood that the IP formulation for the problem could get too large to handle even for moderately sized problems. Though, there are efficient heuristic algorithms, the NP-hardness makes it difficult to prove optimality or even attain approximation bound guarantee of solutions provided by such algorithms. This is where one resorts to IP. However, without careful and smart formulation techniques, one could easily abuse the state-of-the-art IP solvers. In our study, we experienced how one could use various bounds and warm-start techniques to reduce the problem size and hope the problem to be tractable.

9.1 Limitations in the study

While we considered Lovasz-theta number for the lower bounds, we could have also considered using $\alpha(G)$ to compute the lower bounds. Also, we only solved for a small set of DIMACS instances. Further study is needed for larger problems. With a more flexible Branch-and-Cut framework, future study on Lovasz-theta number for computing bounds in the branch-and-bound tree is necessary. And combining it with our initial goal of using an ML-based heuristic or an ML-based branching strategy would be interesting. Furthermore, the results obtained through this study needs to be compared with more recent benchmarks. This work will be continued.

References

[1] Karla L Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management science*, 39(6):657–682, 1993.

[2] Donald E. Knuth. The sandwich theorem, 1993.

[3] R.M. R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[4] Francesca Marzi, Fabrizio Rossi, and Stefano Smriglio. Computational study of separation algorithms for clique inequalities. *Soft Computing*, 23(9):3013–3027, 2019.

[5] Isabel Méndez-Díaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Appl. Math.*, 154(5):826–847, April 2006.

[6] Isabel Méndez-Díaz and Paula Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159–179, 2008.

[7] Dan Stahlke. Lovasz number. https://gist.github.com/dstahlke/6895643, 2013.

10 References

• Mehrotra, A., M.A. Trick. 1996. A column generation approach for graph coloring. INFORMS J. Comput. 8 344–354
• P. Coll, J. Marenco, I. Mendez Dıaz, and P. Zabala. Facets of the graph coloring polytope. Annals of Operations Research, 116(1-4):79–90, 2002.
• Exact Solution of Graph Coloring Problems via Constraint Programming and Column Generation Stefano Gualandi, Federico Malucelli . 2011
• Set covering and packing formulations of graph coloring: algorithms and first polyhedral results P. Hansen ,M. Labbe† D. Schindl.
• Clique, independent set, and graph coloring . Jeff Pattillo and Sergiy Butenko
• A branch-and-cut algorithm for graph coloring I Méndez-Díaz, P Zabala 2006