Exact algorithms for the Vertex Coloring Problem and its generalisations

Ian-Christopher Ternier

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'Université Paris-Dauphine

Résolution exacte du Problème de Coloration de Graphe et ses variantes

École doctorale de Dauphine – ED 543

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

Mme Virginie GABREL
Université Paris-Dauphine
Directrice de thèse

M. Fabio FURINI
Université Paris-Dauphine
Co-encadrant de thèse

M. Pablo SAN SEGUNDO
Universidad Politécnica de Madrid
Rapporteur

M. Roberto WOLFLER CALVO
Université Paris 13
Rapporteur

Mme Ivana LJUBIC
ESSEC Business School of Paris
Membre du jury

M. Christophe PICOULEAU
CNAM
Président du jury

Soutenue par
Ian-Christopher TERNIER
le **21.11.2017**

Dirigée par **Virginie GABREL**

Remerciements

Je remercie ma directrice de thèse Virginie Gabrel pour sa bienveillance naturelle, sa précision, ses conseils de recherche et de vie qui ont toujours été sages. Je remercie également Fabio Furini, qui m'a aussi encadré pendant cette thèse, avec qui j'ai appris la rigueur scientifique et qui a toujours su me montrer l'exemple avec sa discipline de travail. Je remercie Enrico Malaguti pour m'avoir chaleureusement acceuilli en Italie et pour son expertise. Je remercie également Sébastien Martin pour sa vivacité d'esprit, ses conseils et sa bonne humeur.

Je remercie aussi particulièrement Pablo San Segundo et Roberto Wolfler Calvo qui ont accepté d'être rapporteurs de ma thèse. Je remercie également Ivana Ljubic et Christophe Picouleau qui ont accepté d'être membres du jury.

Un bon enseignant peut parfois changer la vie d'une personne. Je repense à deux personnes rencontrées dans un cadre scolaire et qui m'ont beaucoup apporté. Je remercie Madame Allory pour m'avoir redonné le goût des études au lycée. Je remercie également Pierre Fouilhoux qui est le meilleur enseignant que j'ai jamais eu.

Je remercie également les membres du LAMSADE avec qui j'ai passé ces 3 ans. Mon khey Thomas pour son esprit critique et nos discussions toujours intéressantes. Meriem la princesse de Tunis pour sa personnalité étonnante et son humour. Justin le plus français des australiens pour ces heures passées à grimper, se reposer et débattre de tout et de rien. Ioannis mon grec préféré avec qui je trouve toujours de quoi rire. Fabien le chinois pour sa gentillesse naturelle. Anaëlle la polonaise qui égaye le laboratoire par sa simple présence. Khalil le voyageur de la nature et de l'esprit. Satya le râleur invétéré du 38. Marek l'éternel absent. Yassine pour sa personnalité si joviale. Tom Denat pour son nom magnifique. Maude mon ancienne collègue de yoga. Hiba l'éternelle inquiète. Marcel l'anarchiste de Dauphine. Saeed et Mahdi les champions de la corde à sauter. Maryam pour son gâteau au safran. Hossein mon voisin parti trop vite. Amin pour son sourire qui ne le quitte jamais. Diana la libanaise toujours assortie. Paolo-Henrique que l'on n'a vu que trop peu. Céline la nouvelle recrue tunisienne pour sa bonne humeur. Boris dont j'aurais apprécié la compagnie pendant ma thèse. Sylvia ma soeur de thèse. Mohamed Amine qui part toujours aussi vite qu'il arrive. Pedro qui m'a fait découvrir le Portugal. Anne, à qui je n'ai jamais parlé. Olivier, qui finalement est un doctorant, pour sa gentillesse. Je remercie les sages ex-doctorants qui ont accompagné les soirées du LAMSADE de leur sagesse. Je remercie Dr. Belhoul avec qui chaque discussion est un plaisir. Dr. Barrot le japonais roi de l'improvisation. Dr. Lacour le grand randonneur. Une pensée pour les ex-doctorants qui nous ont quitté trop tôt. Dr. Tlilane la stressée, Dr. Haddad la mère de famille, Dr. Magnouche le père de famille, Dr. Atif épouse Leroy, Dr. Kaddani le punchlineur. Je remercie également les dauphins et les dauphines qui m'ont changé les idées pendant les heures de cours.

Je n'aurais jamais complété cette thèse sans les heures de randonnée passées avec Jérémy en pleine nature ou devant un écran. Je remercie également mon insaisissable collègue du LIP6, Siao-Leu. Une pensée pour Yuan et Abdel de l'équipe ADRO.

Je remercie également toutes les personnes jouant aux jeux vidéo de 18 à 25 ans, ainsi que Juan Joya Borja. Je ne serais pas là où je suis aujourd'hui sans les Smasheurs français avec qui j'ai passé de si nombreuses années. Une pensée particulière pour les membres de la Cascade Du Seum : Slhoka, Lguy, Chikusho, *Zen, Crogo mais également Seb, Mahie et Samplay. Je repense également à mon italienne préférée, Domiziana, qui m'a beaucoup appris.

Je repense à l'équipe Nanterroise avec qui j'ai perdu beaucoup de temps et ai beaucoup mangé japonais : Cyril, Paulo, Brandao, Varoun, Merwann, Ghazi, Babo, Kevin.

Une pensée pour mes compagnons de marivaudage, que ce soit Fizzy l'amoureux ou Ted avec qui j'ai battu le pavé parisien.

Finalement, je souhaiterais remercier ma famille. Je remercie de tout mon coeur les deux personnes sans qui je ne serais pas la, ma mère et mon père, qui m'ont toujours soutenu et aimé, ainsi que ma soeur Jennifer.

Abstract

In this manuscript, we study and solve with exact methods combinatorial optimization problems. We focus in particular on NP-hard problems that we solve with dedicated algorithms based on Branch-and-Bound algorithms and Integer Linear Programming (ILP).

Given an undirected graph, the Vertex Coloring Problem (VCP) consists of assigning a color to each vertex of the graph such that two adjacent vertices do not share the same color and the total number of colors is minimized. DSATUR is an effective exact algorithm for the VCP. One of its main drawbacks is that a lower bound is computed only once at the root node of the branching scheme and it is never updated. We introduce a reduced graph which allows the computation of lower bounds at each node of the search tree. We compare the effectiveness of different classical VCP bounds, plus a new lower bound based one the 1-to-1 mapping between VCPs and Stable Set Problems. We introduce new Vertex Selection Rules designed for our new algorithm. Our new DSATUR outperforms the state of the art for random VCP instances with high density, significantly increasing the size of solvable instances. Similar results can be achieved for a subset of high density DIMACS instances.

We study three ILP formulations for the Minimum Sum Coloring Problem (MSCP). The problem is an extension of the classical Vertex Coloring Problem in which each color is represented by a positive natural number. The MSCP asks to minimize the sum of the cardinality of subsets of vertices receiving the same color, weighted by the index of the color, while ensuring that vertices linked by an edge receive different colors. The first and the second ILP formulations have a polynomial number of variables while the third one has an exponential number of variables and is tackled via column generation. We devise a complete Branch-and-Price algorithm aiming to solve efficiently the MSCP. Computational tests show that the third formulation is very competitive for a random benchmark of instances and shows a lot of potential for DIMACS instances. We further expand our study of the third formulation in order to make it more efficient. We start by introducing a new class of variables aimed at reducing the total number of columns. Then we propose a different branching rule to potentially reduce the size of the branching tree. Finally, we tackle the slow convergence of the column generation algorithm by adapting known methods from the literature.

Key words: Graph Coloring, DSATUR, Branch and Bound, Integer Linear Programming, Column Generation, Branch-and-Price.

Résumé

Dans ce manuscrit, nous étudions et résolvons des problèmes d'optimisation combinatoire à l'aide de méthodes exactes. Nous nous intéressons en particulier à des problèmes NP-difficile que nous résolvons avec des algorithmes basés sur des algorithmes de séparation et évaluation et sur de la Programmation Linéaire en Nombres Entiers (PLNE).

Dans un graphe non orienté, le Problème de Coloration de Graphe (PCG) consiste à assigner à chaque sommet du graphe une couleur de telle sorte qu'aucune paire de sommets adjacents n'aient la même couleur et le nombre total de couleurs est minimisé. DSATUR est un algorithme exact efficace pour résoudre le PCG. Un de ses défauts est qu'une borne inférieure est calculée une seule fois au noeud racine de l'algorithme de branchement, et n'est jamais mise à jour. Nous introduisons un graphe réduit qui permet le calcul de bornes inférieures à chaque noeud de l'arborescence. Nous comparons l'efficacité de différentes bornes classiques pour le PCG ainsi qu'une nouvelle borne basée sur une correspondance entre le PCG et le Problème du Stable Maximum. Nous introduisons de nouvelles règles de sélection de sommets conçues pour notre nouvel algorithme. Notre nouvelle version de DSATUR surpasse l'état de l'art pour un ensemble d'instances aléatoires à haute densité, augmentant significativement la taille des instances résolues. Des résultats similaires sont atteints pour un sous-ensemble d'instances DIMACS à haute densité.

Nous étudions trois formulations PLNE pour le Problème de la Somme Chromatique Minimale (PSCM). Ce problème est une variante du PCG où chaque couleur est représentée par un entier naturel. Le PSCM cherche à minimiser la somme des cardinalités des sous-ensembles des sommets recevant la même couleur, pondérés par l'entier correspondant à la couleur, de telle sorte que toute paire de sommets adjacents reçoive des couleurs différentes. Les deux premières formulations ont un nombre polynomial de variables et de contraintes tandis que la troisième possède un nombre exponentiel de variables et utilise un algorithme de génération de colonnes. Nous proposons un algorithme de Branch-and-Price destiné à résoudre le PSCM de manière exacte. Nos résultats expérimentaux nous indiquent que cette formulation est très efficace pour des instances aléatoires and montre du potentiel pour les instances DIMACS. Nous étendons notre étude de la troisième formulation pour la rendre plus efficace. Nous commençons par introduire un nouveau type de variable destiné à réduire le nombre total de variables. Nous proposons également une autre règle de branchement pour potentiellement réduire la taille de l'arborescence. Finalement, nous nous attaquons à la lente convergence de l'algorithme de génération de colonnes en adaptant des techniques issues de la littérature.

Mots clés: Coloration de graphe, DSATUR, Séparation et Evaluation, Programmation Linéaire en Nombres Entiers, Génération de colonnes, Algorithme de génération de colonnes et branchement.

Résumé long

Introduction

L'optimisation combinatoire est une branche de la recherche opérationelle reliée à l'informatique et aux mathématiques appliquées. Le but de ce domaine de recherche est d'étudier des problèmes d'optimisation où l'ensemble des solutions réalisables est discret ou peut être représenté par un ensemble discret.

De nombreuses méthodes ont été mises en oeuvre pour résoudre ces problèmes d'optimisation. Certaines de ces approches sont basées sur la théorie des graphes, tandis que d'autres sont basées sur la programmation linéaire et la programmation linéaire en nombres entiers.

De plus, beaucoup de problèmes issus de la vie courante peuvent être formulés comme des problèmes d'optimisation combinatoire. Ces problèmes sont applicables dans des domaines tels que la télécomunication, la planification de production industrielle ou encore le transport aérien. Les résultats théoriques liés à l'étude de ces problèmes depuis des années ont permis à l'optimisation combinatoire de se développer. De la même manière, ces progrès ont permis une meilleure résolution et une meilleure modélisation de ces problèmes.

La théorie de la complexité est une branche de l'informatique théorique et des mathématiques appliquées, dont les premiers résultats émanent de Cook [7], Edmonds [12] et Karp [29]. L'objectif de ce domaine est de donner une classification des problèmes combinatoires selon leur difficulté de résolution. Un problème NP-difficile ne possède pas d'algorithme de résolution exact exécutable en temps polynomial. Dans ce manuscrit nous nous intéressons à des problèmes caractérisés comme NP-difficile.

Un graphe est défini par un ensemble de sommets et un ensemble d'arêtes. Une arête peut potentiellement lier toute paire de sommets du graphe. Deux sommets liés par une arête sont adjacents. Une coloration d'un graphe est un ensemble de couleurs attribuées à chaque sommet du graphe de telle sorte que toute paire de sommets adjacents n'aient pas la même couleur. Le problème de coloration d'un graphe cherche à déterminer le nombre minimal de couleurs nécessaires pour produire une coloration réalisable d'un graphe.

Le problème de coloration de graphe (PCG) est un problème NP-difficile classique en théorie des graphes. On en trouve des applications notamment dans les domaines suivants : planification, création d'emploi du temps, affectation de fréquences, réseaux de communication, et de manière générale potentiellement tout problème possédant des incompatibilités. Le PCG a été longuement étudié depuis sa création. Un certain nombre d'articles se sont concentrés sur l'étude d'un algorithme de séparation et évaluation nommé DSATUR, introduit par Brélaz en 1979. DSATUR est un algorithme exact compétitif pour résoudre le PCG, notamment performant sur un ensemble d'instances aléatoires. Il est également utilisé en tant que procédure d'initialisation de beaucoup d'algorithmes exacts résolvant le PCG. Le but de ce manuscrit est d'étudier et de résoudre de manière exacte le PCG et des variantes de ce problème.

Nous commençons notre étude du PCG par l'étude de l'algorithme exact de séparation et évaluation DSATUR. Nous améliorons l'algorithme DSATUR en étudiant, en adaptant et en incorporant des bornes inférieures dans cet algorithme de séparation et évaluation. Nous commençons par une étude théorique et expérimentale des bornes inférieures pour le PCG. Grâce à cette étude, nous identifions les bornes les plus prometteuses à injecter dans DSATUR. Parmi ces bornes se trouvent notamment une borne basée sur une transformation de graphe du problème de coloration vers un autre problème combinatoire et le nombre chromatique fractionnaire. Nous définissons par la suite une procédure permettant d'injecter cette borne dans DSATUR à chaque noeud de l'arborescence. Cette procédure est également basée sur une transformation du graphe à partir des informations disponibles à chaque noeud de l'arborescence. Une fois ces nouvelles composantes de l'algorithme bien définies, nous comparons expérimentalement ce nouvel algorithme à DSATUR.

Dans une seconde partie, nous étudions une variante du PCG, appelé le problème de la somme chromatique minimale (PSCM). Etant donné un graphe et une coloration de ce graphe, on associe à chaque couleur une valeur entière. La valeur de cette coloration, appelée somme chromatique, est la somme de tous les entiers des sommets. Le PSCM cherche à déterminer la coloration de somme chromatique minimale.

Le PSCM a également des applications pratiques. Il correspond notamment à un problème de planification sur une seule machine où les tâches sont incompatibles et possèdent des dates de disponibilité. Pour résoudre ce problème, nous utilisons la Programmation Linéaire en Nombres Entiers (PLNE). Seule la programmation quadratique a été utilisée dans la littérature pour résoudre le PSCM. Nous présentons trois modélisations en PLNE du PSCM. La première et la seconde sont des formulations compactes avec un nombre polynomial de variables et de contraintes. La troisième est une formulation étendue basée sur une décomposition de Dantzig-Wolfe. Nous commençons par une étude des propriétés structurelles des solutions optimales du PSCM. Par la suite, nous présentons ces formulations en PLNE pour le PSCM et leurs principaux composants, notamment l'algorithme de Branch-and-Price pour la formulation étendue. Enfin, nous comparons ces formulations expérimentalement entre elles et avec la littérature sur le PSCM.

Ce manuscrit est organisé de la manière suivante. Le premier chapitre est dédié à un état de l'art sur le PCG. Nous présentons également des notions combinatoires et algorithmiques que nous allons étudier dans ce manuscrit. Le second chapitre concerne l'étude et l'amélioration de l'algorithme DSATUR destiné à résoudre le PCG.

Etat de l'art

Le premier chapitre est consacré à l'introduction de notions combinatoires et algorithmiques qui vont nous servir pour la suite du manuscrit. Nous introduisons les algorithmes de séparation et évaluation ainsi que la PLNE.

Une fois ces notions définies, nous présentons un état de l'art du PCG focalisé sur les formulations PLNE le modélisant. Nous commençons par présenter des formulations compactes pour le problème. Nous distinguons trois formulations compactes pour le PCG. La première utilise des variables assignant chaque sommet à une couleur précise. La seconde est basée sur une transformation du PCG en un problème de stable de poids maximum. La troisième utilise des variables liant chaque paire de sommets non adjacents et indiquant si ces deux sommets ont la même couleur. Nous finissons par introduire une formulation non-compacte comportant un nombre exponentiel de variables. Cette formulation utilise des variables correspondant à chaque stable du graphe. Nous présentons chaque formulation puis indiquons brièvement la littérature associée à chacune d'elle.

Améliorations de l'algorithme de Branch-and-Bound DSATUR pour le problème de coloration de graphes

Ce second chapitre est dédié à l'étude et l'amélioration de l'algorithme de séparation et évaluation DSATUR destiné à résoudre le PCG.

Dans un premier temps, nous expliquons les principaux composants de l'algorithme, à savoir, la règle de branchement, les bornes inférieures et supérieures calculées par l'algorithme, et la procédure de sélection du sommet à colorier à chaque noeud de l'arborescence. Par la suite nous introduisons un état de l'art sur DSATUR. Dans la littérature, les différentes versions de DSATUR diffèrent principalement par leur procédure de sélection du sommet à colorier. Sewell [54] et San Segundo [51] proposent des règles différentes visant à diminuer le nombre de noeuds de DSATUR, ce qui diminue également son temps de calcul.

Une fois DSATUR défini, nous présentons les bornes inférieures existantes pour le problème de coloration. Nous revenons en détail successivement sur la valeur de la clique maximum, une borne basée sur la valeur de stable maximum, le nombre de Lovász, le nombre chromatique fractionnaire et le nombre de Hoffman. Nous présentons également une nouvelle borne basée sur la transformation du PCG en un problème de stable de poids maximum.

Nous comparons ensuite les valeurs et les temps de calcul respectifs de ces bornes dans une section dédiée aux résultats expérimentaux. Les conclusions de cette section nous permettent d'identifier trois bornes potentielles à injecter dans l'algorithme DSATUR.

Par la suite, nous introduisons le graphe réduit qui va nous permettre de calculer toute borne inférieure sur le PCG à chaque noeud de l'arborescence. Ce graphe est calculé à partir du graphe de départ à colorier et d'une coloration partielle. Le graphe réduit crée un sommet représentant chaque couleur dont le voisinage est désormais l'union des voisinages de tous les sommets portant cette même couleur. Des arêtes sont également ajoutées à chaque paire de sommets représentant une couleur.

Une fois ce graphe réduit défini, nous présentons formellement le nouvel algorithme DSATUR. Nous introduisons également une fonction qui définit pour chaque noeud de l'arborescence si la borne est calculée. Cette fonction va nous permettre par la suite d'introduire des stratégies visant à limiter le nombre de bornes calculées dans le nouvel algorithme DSATUR$_{LB}$.

Dans la section suivante, nous nous intéressons à l'ensemble des graphes réduits associés à chaque noeud de l'arborescence. Il est possible que deux graphes réduits associés à deux noeuds distincts de l'arborescence soient isomorphes. Dans ce cas, cela implique que toute borne inférieure calculée au premier noeud possède la même valeur dans le second noeud, et dans ce cas, pourrait ne pas être recalculée, rendant ainsi le nouvel algorithme DSATUR$_{LB}$ plus efficace. Nous étudions ces situations en distinguant deux cas : le cas d'un noeud avec un de ses noeuds fils, et le cas de deux noeuds sans filiation. Nous proposons un critère simple pour reconnaitre le cas où deux graphes réduits sont isomorphes.

Nous proposons également une nouvelle procédure de sélection de sommet pour le nouvel algorithme DSATUR$_{LB}$. Nous introduisons ces procédures puis nous les testons avec les procédures de sélection de sommet utilisées dans la littérature.

Enfin, nous réalisons une comparaison expérimentale complète entre le nouvel algorithme DSATUR$_{LB}$ et DSATUR. Ces résultats expérimentaux nous indiquent que la nouvelle version de DSATUR semble plus compétitive que DSATUR en nombre d'instances résolues et en temps de calcul, surtout sur les instances de grande taille et de densité importante. Nos résultats expérimentaux montrent également qu'un certain nombre de bornes ne parviennent pas à couper des parties de l'arborescence. Nous proposons alors des stratégies afin de rendre le nouvel algorithme DSATUR plus efficace. Ces stratégies ont pour objet de sélectionner un ensemble de noeuds sur lesquels le calcul de bornes inférieures sera activé.

Modèles de programmation linéaire en nombres entiers pour le problème de la somme chromatique minimale

Dans le troisième chapitre, nous étudions le PSCM en proposant des formulations PLNE pour le résoudre de manière exacte.

Nous commençons ce chapitre avec un état de l'art sur le PSCM. Nous passons en revue des résultats classiques et remarquons qu'aucune formulation PLNE n'a été proposée et étudiée profondément pour le PSCM.

Dans la section suivante, nous présentons des propriétés structurelles caractérisant toute solution optimale du PSCM.

Une fois ces propriétés établies, nous proposons trois formulations PLNE pour le PSCM. Ces formulations sont des adaptations de formulations existantes pour le PCG définies dans le chapitre état de l'art de ce manuscrit. La première formulation est une formulation compacte qui utilise des variables assignant chaque sommet à une couleur précise. La seconde formulation est également compacte et utilise des variables pour chaque paire de sommets non adjacents et chaque couleur, qui définissent si les sommets ont la même couleur pour une couleur donnée. La dernière formulation est une formulation non-compacte qui utilise des variables pour chaque stable du graphe et chaque couleur. Une fois introduites, nous définissons les principaux composants de ces formulations. Pour la formulation étendue nous définissons la règle de branchement utilisée, l'initialisation des variables de la formulation et l'algorithme de génération de colonnes associé à la formulation.

La règle de branchement est un dérivé de la règle de Zykov [60] utilisée pour le PCG. A partir d'un noeud de l'arborescence, cette règle crée deux noeuds fils à partir d'une paire de sommets non adjacents. Un premier noeud fils définit un nouveau sous-problème où les deux sommets n'ont pas la même couleur. Un second noeud fils définit quant à lui un nouveau sous-problème où les deux sommets ont la même couleur.

L'initialisation des variables de la formulation se fait grâce à une heuristique produisant une solution réalisable du PSCM. Les variables associées à la structure de cette solution sont alors ajoutées à la formulation.

L'algorithme de génération de colonnes est composé d'un problème de pricing pour chaque couleur possible. Chaque problème de pricing est un problème de stable de poids maximum. Nous introduisons des procédures pour l'ajout de variables lors d'une itération de l'algorithme de génération de colonnes. Nous proposons plusieurs procédures dont une basée sur une stratégie existante de la littérature du PCG. Nous comparons ensuite ces procédures expérimentalement, et choisissons la plus performante pour le reste du manuscrit. A partir d'une variable à cout réduit minimum, la procédure consistant à ajouter cette variable et des variables proches dont le coût réduit est négatif est la plus performante.

Nous présentons alors notre étude expérimentale en comparant la première formulation compacte avec la formulation étendue pour le PSCM. Nous utilisons deux ensembles d'instances pour effectuer nos tests : un ensemble d'instances aléatoires et un ensemble d'instances DIMACS. Sur l'ensemble d'instances aléatoires la formulation étendue est plus performante au niveau du nombre d'instances résolues et du temps de calcul. Pour l'ensemble d'instances DIMACS, la formulation compacte parvient à résoudre plus d'instances mais la formulation étendue semble avoir plus de potentiel. La relaxation linéaire de la formulation étendue possède une valeur très compétitive mais elle est expérimentalement dure à obtenir. Dans la prochaine section nous présentons des méthodes que nous avons étudié dans l'espoir d'améliorer cette formulation étendue.

Nous commençons cette section en proposant une formulation retravaillée de la formulation étendue pour le PSCM. Un problème crucial de cette formulation est qu'à la différence du PCG, une solution optimale du PSCM ne peut pas être exprimée en tant que couverture par des stables mais seulement par une partition par des stables. Cela implique que pour un stable donné, tous les sous-stables doivent être générés dans la formulation. Nous avons donc plus d'itérations dans l'algorithme de génération de colonnes et un algorithme de résolution de la relaxation linéaire généralement plus lent. Nous proposons d'introduire un nouveau type de variables destiné à supprimer ce problème. Ces nouvelles variables permettent, à partir d'une variable associé à un stable et une couleur, de reconstruire tous les sous-stables dans la formulation. Grâce à l'introduction de ces variables, cette formulation retravaillée permet de résoudre la relaxation linéaire de la formulation étendue pour un plus grand nombre d'instances.

Nous proposons par la suite une autre règle de branchement pour la formulation étendue. Cette nouvelle règle de branchement crée, à partir d'un sommet non-colorié et d'une couleur réalisable pour ce sommet, deux noeuds fils. Le premier noeud fils définit un nouveau sous-problème où la couleur est assignée au sommet. Le second noeud fils définit un nouveau sous-problème où la couleur ne peut pas être assignée au sommet.

Par la suite, nous introduisons une classe d'inégalités valides pour la formulation étendue. Ces inégalités sont basées sur une propriété structurelle d'une solution optimale du PCSM. Cette propriété indique que, considérant deux couleurs successives dans une solution optimale h et $h+1$, le nombre de sommets ayant reçu la couleur h doit être plus grand ou égal au nombre de sommets ayant reçu la couleur suivante.

Un élément également crucial de cette formulation est l'initialisation du nombre de couleurs possibles pour colorier le graphe. Dans notre formulation étendue, ce nombre définit à la fois le nombre de sous-problèmes de pricing que nous devrons résoudre et le nombre de variables. Plus ce nombre est proche du nombre de couleurs associé à une solution optimale du PSCM, plus la formulation sera performante. Nous proposons une procédure pour potentiellement mettre à jour ce nombre à chaque fois qu'une nouvelle solution réalisable pour le PSCM est trouvée par la formulation.

Un des problèmes majeurs de la formulation étendue est la lente convergence de l'algorithme de génération de colonnes. Des solutions à ce problème, caractéristique des formulations utilisant un algorithme de génération de colonne, ont été étudié dans la littérature. Nous essayons d'adapter certaines de ces solutions à notre formulation étendue pour le PSCM.

La première méthode utilisée est la Boxstep method. Une des causes des problèmes de convergence de l'algorithme de génération de colonnes est l'oscillation des valeurs de la solution optimale de la relaxation linéaire. La Boxstep method propose d'améliorer la convergence d'un algorithme de génération de colonnes en limitant l'oscillation des valeurs de la solution optimale de la relaxation linéaire. L'idée est d'introduire des variables et des contraintes dans la formulation qui vont pénaliser toute solution de la relaxation linéaire s'éloignant trop de la solution de la relaxation linéaire de l'itération précédente.

La seconde méthode utilisée est la Dual Ascent. Cette méthode a déjà été utilisée avec succès notamment sur des problèmes de partitionnement, qui sont des problèmes relativement proches du PSCM. Il s'agit d'opérer une relaxation paramétrique puis une relaxation lagrangienne de la formulation étendue. La nouvelle formulation obtenue est donc une relaxation de la formulation étendue. Dans la littérature, la résolution de cette nouvelle formulation est plus rapide et produit des bornes inférieures proches de la véritable valeur de la borne inférieure du problème. Ces deux dernières méthodes ne nous ont pas permis d'améliorer le calcul de la relaxation linéaire de la formulation étendue.

Conclusion

Nous clôturons ce manuscrit par une conclusion où nous revenons sur le travail effectué au cours de cette thèse. Nous proposons des pistes de recherche liées aux thèmes de recherche que nous avons abordés dans ce manuscrit.

Introduction

Combinatorial Optimization is a branch of operations research related to computer science and applied mathematics. Its purpose is the study of optimization problems where the set of feasible solutions is discrete or can be represented as a discrete one. Typically, the problems concerned with combinatorial optimization are those formulated as follows. Let $E = \{e_1, \ldots, e_n\}$ be a finite set called *basic set* where each element e_i is associated with a weight $c(e_i)$. Let $\mathcal{F}$ be a family of subsets of E. If $F \in \mathcal{F}$, then $c(F) = \sum\limits_{e_i \in F} c(e_i)$ denotes the weight of F. The problem consists in identifying an element F^* of $\mathcal{F}$ whose weight is minimum or maximum. In other words,

$$\min(or\max)\{c(F) : F \in \mathcal{F}\}.$$

Such a problem is called *combinatorial optimization problem.* The set $\mathcal{F}$ represents the set of feasible solutions of the problem.

The term *combinatorial* refers to the discrete structure of $\mathcal{F}$. In general, this structure is represented by a graph. The term *optimization* signifies that we are looking for the best element in the set of feasible solutions. This set generally contains an exponential number of solutions, therefore, one can not expect to solve a combinatorial optimization problem by exhaustively enumerate all its solutions. Such a problem is then considered as a hard problem.

Various effective approaches have been developed to tackle combinatorial optimization problems. Some of these approaches are based on graph theory, while others use linear and non-linear programming, integer programming and polyhedral approach. Besides, several practical problems arising in real life, can be formulated as combinatorial optimization problems. Their applications are in fields as diverse as telecommunications, transport, industrial production planing or staffing and scheduling in airline companies. Over the years, the discipline got thus enriched by the structural results related to these problems. And, conversely, the progress made in computer science have made combinatorial optimization approaches even more efficient on real-world problems.

In fact, combinatorial optimization is closely related to algorithm theory and computational complexity theory as well.

Computational complexity theory is a branch of theoretical computer science and mathematics, whose study started with works of Cook [7], Edmonds [12] and Karp [29]. Its objective is to give a classification of combinatorial problems depending on their difficulty. A plentiful literature can be find on this topic, see for example [19] for a detailed presentation of NP-completeness theory.

A *problem* is a question having some input parameters, and to which we aim to find an answer as an output. A problem is defined by giving a general description of its parameters, and by listing the properties that must be satisfied by a solution. An *instance* of the problem is obtained by giving a specific value to all its input parameters. An *algorithm* is a sequence of elementary operations that allows to solve the problem for a given instance. The number of input and output parameters necessary to describe an instance of a problem is the *size* of that problem.

An algorithm is said to be polynomial if the number of elementary operations necessary to solve an instance of size n is bounded by a polynomial function in n. We define the class P as the class gathering all the problems for which there exists some polynomial algorithm for each problem instance. A problem that belongs to the class P is said to be "easy" or "tractable".

A *decision problem* is a problem with a *yes* or *no* answer. Let $\mathcal{P}$ be a decision problem and $\mathcal{I}$ the set of instances such that their answer is yes. $\mathcal{P}$ belongs to the class NP (Nondeterministic Polynomial) if there exists a polynomial algorithm allowing to check if the answer is yes for all the instances of $\mathcal{I}$. It is clear that a problem belonging to the class P is also in the class NP. Although the difference between P and NP has not been shown, it is a highly probable conjecture.

In the class NP, we distinguish some problems that may be harder to solve than others. This particular set of problems is called *NP-complete.* To determine whether a problem is NP-complete, we need the notion of *polynomial reducibility.* A decision problem P_1 can be polynomially reduced (or transformed) into an other decision problem P_2, if there exists a polynomial function f such that for every instance I of P_1,

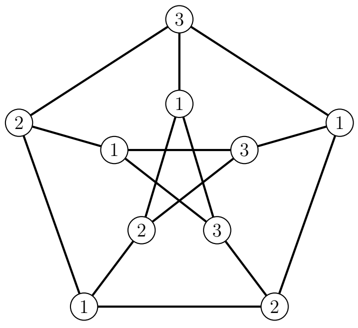

Figure 1: Example of an optimal *Vertex Coloring* in the Petersen graph.

the answer is "yes" if and only if the answer of $f(I)$ for P_2 is "yes". A problem $\mathcal{P}$ in NP is also NP-complete if every other problem in NP can be reduced into $\mathcal{P}$ in polynomial time. The Satisfiability Problem (SAT) is the first problem that was shown to be NP-complete (see [7]).

With every combinatorial optimization problem is associated a decision problem. Furthermore, each optimization problem whose decision problem is NP-complete is said to be *NP-hard.* Note that most of combinatorial optimization problems are NP-hard. One of the most efficient exact approaches developed to solve those problems is Branch-and Bound algorithms and Integer Linear Programming. We now introduce more precisely the subjects and themes discussed in this manuscript.

A *undirected graph* is defined by a set of *vertices* and a set of *edges.* An edge can potentially link any pair of vertices of the graph. Two vertices linked by an edge are *adjacent.* A coloring consists of assigning a color to each vertex of the graph in such a way that two adjacent vertices do not share the same color. The *Vertex Coloring Problem* (VCP) minimizes the total number of colors in any coloring of the graph. Figure 1 shows an example of an optimally colored graph (Petersen graph) with 10 vertices. Each number inside the vertex of this graph refers to the color it has been assigned. The optimal number of colors for this graph is 3.

The VCP is one of the classical NP-hard problems in graph theory with applications in many areas including: scheduling, timetabling, register allocation, frequency assignment, communication networks and many others. The aim of this manuscript is to study and solve the VCP and related combinatorial problems with exact methods. The VCP has received a large amount of attention in the last decades and many articles investigated an exact implicit enumeration algorithm called *DSATUR-based Branch-and-Bound algorithm* (DSATUR) first introduced by Brélaz in 1979. DSATUR is the leading VCP algorithm for a specific benchmark of random instances. It is also used as an initialization technique in many exact VCP algorithms, thus proving its relevancy.

We start by solving the VCP with a dedicated *Branch-and-Bound algorithm.* We improve DSATUR by investigating, adapting and incorporating lower bounding techniques into this Branch-and-Bound algorithm. We first study lower bounds for the VCP both theoretically and computationally. Thanks to this study, we identify the most relevant lower bounds to use in DSATUR. Among these bounds is a bound based on transforming the VCP into another combinatorial problem, and the classical fractional chromatic number. Then, we devise a procedure that enables DSATUR to use one of these bounds at each node of the branching tree. This procedure is also based on a graph transformation linked to the information available at each node of the branching tree. Once all these components have been clearly defined, we computationally compare DSATUR and our improved DSATUR algorithm.

We then focus our study on a variant of the VCP, the *Minimum Sum Coloring Problem* (MSCP). Given an undirected graph and a coloring of the graph, the *sum coloring* is the sum of all integers assigned to each vertex of the graph corresponding to the integer of the color it has been assigned. The MSCP minimizes the value of the sum coloring. Figure 2 shows an example optimal solution of the MSCP in a graph (Petersen graph) with 10 vertices. Each integer inside the vertex of this graph refers to the color it has been assigned. The optimal value of the MSCP is 19. It is a variant of the VCP where integer numbers are now assigned to each color and thus to each vertex of the graph. The sum of each integer assigned to each vertex of the graph has to be minimized to solve the MSCP optimally. This problem was first introduced by Kubicka and Schwenk in 1989. Most of the relevant literature on the MSCP focus on finding good upper bounds or complexity results.

The MSCP is a relevant problem from a practical point of view since it is a special case of the scheduling problem of incompatible jobs with release dates on a machine. We solve the MSCP using *Integer Linear Programming* (ILP) models. We present three dedicated ILP formulations for the MSCP. The first and the second one are compact formulations with a polynomial number of variables and constraints. The third one is an extended formulation based on a *Dantzig-Wolfe decomposition.* We first study structural properties of any optimal solution of the MSCP that define and shape our ILP formulations. Then we present the three formulations and the components of the algorithms we use to solve them, in particular the Branch-and-Price algorithm for the

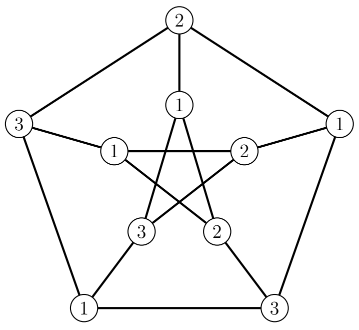

Figure 2: Example of an optimal *Minimum Sum Coloring* in the Petersen graph.

extended formulation. Finally, we computationally compare the formulations with each other and with the current literature.

The remainder of this manuscript is organized as follows. The next section will formally introduce the concepts, definitions and notations used in this manuscript. In Chapter 1, we present state-of-the-art ILP models and exact algorithms for solving the VCP. In Chapter 2 we present an improved version of DSATUR for solving the VCP with original lower bounding techniques. In Chapter 3, we introduce ILP formulations to solve the MSCP and we develop a Branch-and-Price algorithm. Finally, the last chapter will offer a conclusion to this manuscript.

Preliminary notations and definitions

Given an undirected graph $G = (V, E)$, V is the set of *vertices* and E is the set of *edges*. An edge e linking two vertices u and v can be denoted (u, v). We denote $|V| = n$ vertices and $|E| = m$ edges. The *neighborhood* $N(v)$ of a vertex $v \in V$ is the set of vertices such that $\forall u \in N(v)$, $(u, v) \in E$. The degree of a vertex $v \in V$ is the size of its neighborhood $N(v)$ denoted $|N(v)|$ or $d(v)$. When the context is ambiguous, we introduce the notation $N_G(v)$ which is the neighborhood of vertex v in graph G.

Given an undirected graph $G = (V, E)$, the *complementary graph* $\bar{G} = (V, \bar{E})$ of G is such that $\bar{E} = \{(u, v)|u, v \in V (u, v) \notin E\}$. We denote $|\bar{E}| = \bar{m}$ edges. The anti-neighborhood $\bar{N}(v)$ denoted $|\bar{N}(v)|$ or $\bar{d}(v)$ is the set of vertices such that $\forall u \in \bar{N}(v)$, $(u, v) \notin E$.

Given a directed graph $G = (V, \overrightarrow{E})$, $\overrightarrow{E}$ is the set of *arcs*. An arc linking two vertices u and v can be denoted (u, v). Since arcs are directed, (u, v) and (v, u) are two different arcs.

Given an undirected graph $G = (V, E)$, a *stable set* $S \subseteq V$ is a subset of vertices such that $\forall u, v \in S$, $(u, v) \notin E$. Let us denote the collection of all stable sets of G as $\mathcal{S} = \{S \subseteq V : (u, v) \notin E, u \in S, v \in S\}$. The cardinality of the maximum stable set of a graph G is denoted $\alpha(G)$ and called the *stability number*. Its associated optimization problem is the *Maximum Stable Set Problem* (MSSP). There is a variant of this problem, the *Maximum Weighted Stable Set Problem* (MWSSP), where weights w_u are assigned to each vertex $u \in V$. The optimal solution is a stable set S for which the value of the sum of the weights of its vertices $\sum\limits_{u \in S} w_u$ is maximum in G (see Figure 3).

Given an undirected graph $G = (V, E)$, a *clique* $K \subseteq V$ is a subset of vertices such that $\forall u, v \in K$, $(u, v) \in E$. The cardinality of the maximum clique of a graph G is denoted $\omega(G)$ and called the *clique number*. Its associated optimization problem is the *Maximum Clique Problem* (MCP). Trivially, $\omega(G) = \alpha(\bar{G})$. (see Figure 4)

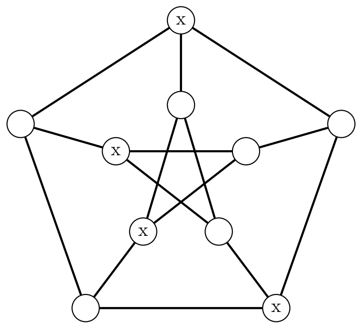

Figure 3: Example of a *Maximum Stable Set* in the Petersen graph G. The set of four vertices marked with an 'x' form a *Maximum Stable Set* and the *stability number* of G is $\alpha(G) = 4$.

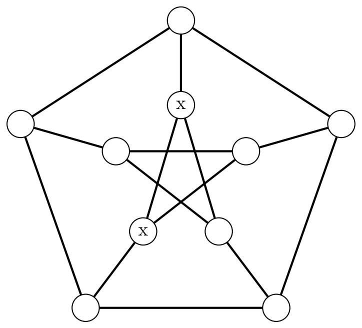

Figure 4: Example of a *Maximum Clique* in the Petersen graph G. The set of two vertices marked with an 'x' form a *Maximum Clique* and the *clique number* of G is $\omega(G) = 2$.

Given an undirected graph $G = (V, E)$, a *path* P is a sequence of distinct vertices $P = (v_1, v_2, \ldots, v_k)$ such that v_i and v_{i+1} are adjacent for $1 \leq i < k$. A *chord* in a path P is a pair of non-consecutive vertices $\{v_i, v_j\}$ such that $(v_i, v_j) \in E$ and $v_i \neq v_1, v_k$ and $v_j \neq v_1, v_k$. A path is *chordless* if for each pair of non-consecutive vertices $\{v_i, v_j\}$, $(v_i, v_j) \notin E$.

Given an undirected graph $G = (V, E)$, a *cycle* is a path $(v_1, v_2, \ldots, v_k)$ where v_1 and v_k are adjacent, denoted C. A *chord* in a cycle C is a pair of non-consecutive vertices $\{v_i, v_j\}$ such that $(v_i, v_j) \in E$. A cycle is *chordless* if for each pair of non-consecutive vertices $\{v_i, v_j\}$, $(v_i, v_j) \notin E$.

Given an undirected graph $G = (V, E)$ and considering a subset of vertices $T \subseteq V$, the *induced subgraph* $G(T)$ is the graph $G(T) = (T, E(T))$ where $E(T) = \{(u, v) | u \in$

$T, v \in T, (u, v) \in E\}$.

Given an undirected graph $G = (V, E)$ and two vertices $u, v \in V$ such that $(u, v) \notin E$, *merging* vertices u and v in G implies the following consequences in the new graph $G' = (V', E')$. v is removed from V (arbitrarily), i.e., $V' = V \backslash \{v\}$ and $N_{G'}(u) = N_G(u) \cup N_G(v)$, i.e., $E' = E \backslash \{N_G(u) \cap N_G(v)\}$. The merging of vertices u and v in G is denoted $\{u + v\}$.

Vertex Coloring Problem

Given an undirected graph $G = (V, E)$, a *coloring* C of G is a partition of V into k non empty stable sets: $C = \{V_1, \dots, V_k\}$, where all vertices belonging to V_i are colored with the same color i $(i = 1, \dots, k)$. The *chromatic number* of G, denoted by $\chi(G)$, is the minimum number of stable sets (or equivalently colors) in a coloring of G and the *Vertex Coloring Problem* (VCP) is the problem of determining the chromatic number of the graph G.

Given an undirected graph $G = (V, E)$, a *clique cover* $\bar{C}$ of G is a partition of V into k non empty cliques: $\bar{C} = (V_1, \dots, V_k)$. The *clique cover number* of G, denoted by $\bar{\chi}(G)$, is the minimum number of cliques in a clique cover of G and the *Clique Cover Problem* (CCP) is the problem of determining the cliquer cover number of the graph G. Trivially, $\chi(G) = \bar{\chi}(\bar{G})$.

Minimum Sum Coloring Problem

Given an undirected graph $G = (V, E)$ and a coloring $C = \{V_1, ..., V_k\}$ of G, the *sum coloring* of C is given by $\psi(G, C) = \sum_{i \in \{1,\dots,k\}} (i \cdot |V_i|)$. The *Minimum Sum Coloring Problem* (MSCP) consists of finding a coloring C with the minimum cost $\psi(G, C)$. This minimum cost has been denoted as $\Sigma(G)$ and called the *chromatic sum* of the graph G in the literature (see Kubicka and Schwenk [31]). Finally, the smallest number of colors (or equivalently stable sets) associated with the chromatic sum $\Sigma(G)$ is called the *strength* of the graph and denoted by $s(G)$.

Contents

Chapter 1

State of the art

In this chapter, we focus on the state-of-the-art exact methods used to solve the VCP. More precisely, we describe and restrict our study to ILP formulations for the VCP. These formulations can be divided into two families: compact formulations and non-compact formulations. Before introducing these ILP models, we will introduce Integer Linear Programming and Branch-and-Bound algorithms, which are a component of the algorithms solving ILP models. We will then review compact formulations followed by non-compact formulations.

Branch-and-Bound algorithm

Branch-and-Bound algorithms are designed to solve combinatorial optimization problems. The goal is to find the best solution among all possible solutions according to an objective function. At every step of the Branch-and-Bound algorithm, a feasible partial solution is being constructed. In every leaf of the branching tree, a new solution is found.

A Branch-and-Bound has two principal components: the branching rule and the bounding technique.

- The branching rule splits the current subproblem of the branching node into at least two new smaller subproblems. The optimal solution to the current subproblem must be contained inside one of the children subproblem of the current branching node.

- At each node of the branching tree a lower bound is computed. This lower bound potentially allows to cut a potential subtree of the branching tree. If a given lower bound of a branching node is larger than the best upper bound, then no children nodes are created.

We now present Integer Linear Programming, which uses Branch-and-Bound algorithms to solve optimization problems.

Integer Linear Programming

A Linear Program (LP) is a constrained optimization problem defined as follows:

$$(LP)\begin{cases} \text{Max } c^T x \\ \quad Ax \leq b \\ \quad x \in I\!R^n \end{cases}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$. Max $c^T x$ is the objective function of the LP, and $Ax \leq b$ define the constraints of the LP. $x \in I\!R^n$ define the domain of variables x and are also defined as constraints of the LP. A LP can be solved in polynomial time. An Integer Linear Program (ILP) is a LP where variables domain are discreet. It can be defined as follows:

$$(ILP)\begin{cases} \text{Max } c^T x \\ \quad Ax \leq b \\ \quad x \in I\!N^n \end{cases}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$. This problem is NP-hard. Two types of formulations can be distinguished. Compact formulations have a polynomial number of constraints and variables. Non-compact formulations have either an exponential number of constraints, an exponential number of variables or both an exponential number of constraints and variables.

Compact formulations can be solved directly through an ILP solver. Branch-and-Bound algorithms are the standard way to solve an ILP. A branching tree is created based on the ILP model. At every node of the branching tree, a branching decision is made, usually enforced by setting a value for at least one variable of the ILP, until an optimal solution is found. To prove the optimality of this solution and to cut potential subtrees of the branching tree, lower and upper bounds on the problem are computed throughout the algorithm. A LP relaxation of an ILP is obtained by replacing discreet constraints on x by continuous constraints. The usual lower bound is the optimal value of the associated LP relaxation of the ILP.

Non-compact formulations are solved by the same Branch-and-Bound algorithm. The difference lies in the ILP lower bound that is used in the algorithm. For non-compact formulations, dedicated algorithms are needed to obtain the value of the LP relaxation. For formulations with an exponential number of constraints, a cutting plane algorithm is required, and for a formulation with an exponential number of variables, a column generation algorithm is required.

When handling an exponential formulation, all the variables or constraints cannot be added to the model in the initialization procedure. Instead a restricted number of constraints or variables are initially added to the model. Then the column generation algorithm and the cutting plane algorithm principle is to respectively search for a violated constraint or a negative reduced cost variable. A violated constraint or a negative reduced cost variable means that the solution of the LP is not optimal. If there exists no violated constraint or negative reduced cost variables, the LP is optimal and the bound is valid.

We now present ILP models for the VCP.

ILP formulation with vertex-color variables

This formulation is one of the most classical ILP formulation designed to solve the VCP. It has been thoroughly studied by Méndez-Díaz et al. [40, 6, 41, 42]. Let us consider the following variables.

For all $(i, j) \in E$ and for all $k \in \{1, ..., n\}$,

$$x_{ik} = \begin{cases} 1 \text{ if vertex } i \text{ has been assigned color } k, \\ 0 \text{ otherwise.} \end{cases}$$

For all $k \in \{1, ..., n\}$,

$$w_k = \begin{cases} 1 \text{ if color } k \text{ is used in the solution,} \\ 0 \text{ otherwise.} \end{cases}$$

With these two families of variables, the VCP can be formulated as follows:

$$\min \sum_{k\in\{1,...,n\}} w_k \tag{1.1}$$

$$\sum_{k\in\{1,...,n\}} x_{ik} = 1, \qquad i \in V, \tag{1.2}$$

$$x_{ik} + x_{jk} \leq w_k, \qquad (i,j) \in E, k \in \{1,...,n\}, \tag{1.3}$$

$$x_{ik} \in \{0,1\}, \qquad i \in V, k \in \{1,...n\}, \tag{1.4}$$

$$w_k \in \{0,1\}, \qquad k \in \{1,...n\}. \tag{1.5}$$

The objective function (1.1) is the sum of all color variables which corresponds to the number of used colors in the solution. Constraint (1.2) implies that each vertex is assigned exactly one color. Constraint (1.3) implies that two adjacent vertices cannot have the same color for each possible color. Finally constraint (1.4) and constraint (1.5) are respectively the integrality constraints for variables x_{ik} and variables w_k. This formulation is compact as it contains $O(n^2)$ variables and $O(mn)$ constraints.

Méndez-Díaz and Zabala [40] begin the study of this formulation in the literature and give facet-defining inequalities of the polytope. Coll et al. [6] deepen this study and highlight other families of facet-defining inequalities, aimed at building a dedicated Branch-and-Cut algorithm for the VCP. Méndez-Díaz and Zabala [41] devise a Branch-and-Cut algorithm based on their previous work on the formulation's polytope. They build a complete exact algorithm with preprocessing procedures, upper bound initialization, branching rules and a cutting plane algorithm. Méndez-Díaz and Zabala [42] extend this work and introduce symmetry breaking inequalities for the formulation.

ILP formulation with induced cliques

This formulation is based on modeling the solution of the VCP by covering the complementary graph with cliques. The solution is constructed with the following variables.

For all $(i,j) \in \bar{E}$,

$$x_{ij} = \begin{cases} 1 \text{ if } (i,j) \text{ is in the solution} \\ 0 \text{ otherwise.} \end{cases}$$

The idea is that all chosen variables x_{ij} associated to complementary edges $(i,j) \in \bar{E}$ form a clique-covering of $\bar{G}$ which corresponds to a stable-covering of G which corresponds to a coloring of G. Palubeckis [47], and Cornaz and Jost [10] model the VCP as a set of edges spanning a clique-covering of the graph. The two formulations from Palubeckis [47] and Cornaz and Jost [10] are similar, we are going to present Palubeckis version since Cornaz and Jost version will be explained later in this manuscript. Let us consider an arbitrary ordering of the vertices $V = \{1, ..., n\}$. Let :

$$T = \{(i,j,k)|i \leq \min\{i,j,k\} \text{ and } (i,j,k) \text{ is a triangle in } \bar{G}\},$$

$$\Pi = \{(i,j,k)|(i,j),(j,k) \in \bar{E}, (i,k) \notin \bar{E}\},$$

and

$$\min \ n - \sum_{ij \in \bar{E}} x_{ij} \tag{1.6}$$

$$x_{ij} + x_{jk} \leq 1, \qquad (i,j,k) \in T \cup \Pi, \tag{1.7}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in \bar{E}, \tag{1.8}$$

be a formulation for the VCP based on the following theorem.

Theorem 1 (Palubeckis [47]) *Let G be a graph with n vertices. $\chi(G) + \alpha(H_G) = n$.*

$H_G = (V_H, E_H)$ is such that :

- $t_{ij} \in V_H$ if $(i,j) \in \bar{E}$
- $(t_{ij}, t_{jk}) \in E_H$ if $(i,j),(j,k) \in \bar{E}$ and $(i,j,k) \in \Pi$ or $(i,j,k) \in T$.

Formulation (1.6)-(1.8) solves the MSSP in H_G. According to Theorem 1, the objective function (1.6) is n minus the sum of all x_{ij} variables which are corresponding to the vertices of H_G. Constraint (1.7) is the edge constraint for the MSSP between vertices t_{ij} and t_{jk} of H_G. Finally, constraint (1.8) is the integrality constraint for variables x_{ij}. This formulation is compact and uses $|\bar{E}| = O(n^2)$ variables and $|T| + |\Pi| = O(n^3)$ constraints.

Palubeckis [48] studies the polyhedron associated to this formulation focusing on webs and antiwebs inequalities. Cornaz et al. [8] focus on studying coloring problems using similar graph transformations.

ILP formulation with representatives

This formulation is based on defining the color of a vertex relatively to every other vertex. A vertex can either be a representative of a given color or be represented by another vertex. There can be at most one representative to each color. To avoid symmetry, an arbitrary ordering of the vertices $V = \{1, ..., n\}$ is set so that if $u \prec v$, then v cannot represent u. Let us consider the following variables.

For each $u \in V$ and $v \in V$ such that $v \prec u$,

$$x_{uv} = \begin{cases} 1 \text{ if } u \text{ is represented by } v \text{ and thus, they have the same color,} \\ 0 \text{ otherwise,} \end{cases}$$

and for each $u \in V$,

$$x_{uu} = \begin{cases} 1 \text{ if } u \text{ represents a color,} \\ 0 \text{ otherwise.} \end{cases}$$

The VCP can be formulated as follows:

$$\begin{aligned}
\min \ & \sum_{u \in V} x_{uu} && && (1.9)\\
& \sum_{v \in \bar{N}_G(u)} x_{uv} + x_{uu} = 1, && u \in V, && (1.10)\\
& x_{uv} \leq x_{vv}, && u \in V, (u,v) \in E, && (1.11)\\
& x_{vu} + x_{wu} \leq x_{uu}, && (v,w) \in E, (u,v) \notin E, (u,w) \notin E. && (1.12)\\
& x_{uv} \in \{0,1\}, && u, v \in V, v \prec u, && (1.13)\\
& x_{uu} \in \{0,1\}, && u \in V. && (1.14)
\end{aligned}$$

The objective function (1.9) is the sum of all variables x_{uu} indicating how many representatives vertices are in the solution, i.e., the number of used colors. Constraint (1.10) indicates that a vertex u is either a representative or is being represented by another non-adjacent vertex v. Constraints (1.11) and (1.12) are adjacency constraints. Constraints (1.13) and (1.14) are respectively the integrality constraints for variables x_{uu} and x_{uv}. This formulation is compact and has $O(n^2)$ variables and $O(n^2)$ constraints. Campelo et al. [5] perform a facial study of this formulation including both a review of known inequalities and new inequalities.

ILP formulation with stable sets

Mehrotra and Trick [38] first introduce this formulation based on stable sets variables. Let us consider the following variables.

For all stable sets $S \in \mathcal{S}$,

$$\lambda_S = \begin{cases} 1 \text{ if stable set } S \text{ is in the solution,} \\ 0 \text{ otherwise.} \end{cases}$$

The VCP can be formulated as follows:

$$\min \sum_{S\in\mathcal{S}} \lambda_S \tag{1.15}$$

$$\sum_{S\in\mathcal{S}_v} \lambda_S \geq 1, \qquad u \in V, \tag{1.16}$$

$$\lambda_S \in \{0,1\}, \qquad S \in \mathcal{S}. \tag{1.17}$$

The objective function (1.15) is the sum of all variables λ_S which corresponds to all stable sets used in the solution which corresponds to the number of used colors in the solution. Constraint (1.16) is the covering constraint for all vertices of the graph, ensuring that every vertex is part of at least one stable set, i.e., one color. Constraint (1.17) is the integrality constraint for variables λ_S. This formulation is not compact and has $O(n)$ constraints and $O(2^n)$ variables.

Mehrotra and Trick [38] introduce this formulation and design its basic components. Hansen et al. [22] focus on variants of constraints (1.16) for this formulation. They compare the packing ($\leq$), partitioning ($=$) and covering ($\geq$) versions of this formulation. They also study the polyhedron associated to the covering version of this formulation and provide some facial results. Malaguti and Toth [37] publish a survey on the VCP. First, they computationally compare the state-of-the-art methods for the VCP. Then they tackle variants of the VCP and also provide a computational comparison with the state-of-the-art methods for the VCP. Malaguti et al. [36] tackle the Branch-and-Price algorithm by designing efficient methods to solve the pricing. A tabu-search based metaheuristic is used to provide good columns for the formulation during the column generation algorithm as well as an ILP based model. They show that their algorithm is one of the most efficient among all VCP algorithms, comparing themselves to Brélaz [4], and Méndez Díaz and Zabala [42]. Gualandi and Malucelli [21] work on this formulation and use among other things Constraint Programming to solve the pricing subproblem. They also devise pricing strategies to improve the efficiency of the model. Their algorithm is on par with the state-of-the-art algorithms like Malaguti et al. [37]. Held et al. [23] also work on this formulation tackling numerical difficulties in the context of column generation, deriving a way of computing numerically safe bounds. They devise a complete and efficient Branch-and-Price algorithm using their own Branch-and-Bound based algorithm for solving the pricing subproblem. Morrison et al. [44] focus on creating new branching rules that preserve the graph structure at each node of the branching tree. This branching rule allows them to produce a competitive Branch-and-Price algorithm, comparing themselves to both Malaguti et al. [37] and Held et al. [23]. Verma et al. [56] focus on solving the VCP on very large instances based on real life networks. Their work is based on reducing the instance while preserving optimality, thus solving smaller instances. Morrison et al. [45] focus on solving the pricing subproblem using zero-suppressed binary decision diagram. They compare their algorithm to Malaguti et al. [37], Gualandi and Malucelli [21], Held et al. [23] and Morrison et al. [44]. Their algorithm sometimes outperforms the previously mentioned state-of-the-art methods.

Conclusion

In this chapter, we review ILP models for the VCP. These models help us get an overview of the state-of-the-art exact method for solving the VCP. Some of these models will help us during the first chapter dedicated to solving the VCP with a Branch-and-Bound based algorithm and in the second chapter dedicated to solving the MSCP with ILP models.

Chapter 2

An Improved DSATUR-Based Branch-and-Bound Algorithm for the Vertex Coloring Problem

Introduction

The VCP has received a large amount of attention in the last decades and many articles investigated an exact implicit enumeration algorithm called *DSATUR-based Branch-and-Bound algorithm* (DSATUR), first introduced by Brélaz [4] and then improved by Sewell [54] and San Segundo [51]. It is a Branch-and-Bound algorithm where, at each node, the children nodes are created by assigning feasible colors to a uncolored vertex; thus at each node of the branching tree, we have a partial coloring of G and at each leaf we have a coloring of G. Formally, a *partial coloring* $\tilde{C}$ of G is a partition of a subset of vertices $\tilde{V} \subset V$ into $\tilde{k}$ non empty stable sets or colors ($\tilde{C} = \{\tilde{V}_1, \ldots, \tilde{V}_{\tilde{k}}\}$), while the remaining vertices $V \setminus \tilde{V}$ are uncolored. Many rules have been proposed in the literature to determine the sequence of vertices to color (see Section 2.2 for further details on DSATUR). It is worth mentioning that DSATUR has also been successfully applied to other variants of VCPs, see for example Méndez-Diaz et al. [39].

In all the DSATUR versions proposed in the literature, a lower bound is computed once at the root node of the algorithm as a heuristic maximal clique and it is never updated. A second trivial lower bound also used in the literature is the number of colors $\tilde{k}$ of a partial coloring $\tilde{C}$. The principal idea of this chapter consists of updating and improving the quality of the lower bound during the branching scheme. In order to do that, we introduce a *Reduced Graph* associated to a partial coloring which allows to update the lower bounds. We implement and compare the classical lower bounds for VCP, i.e, the clique number, a bound based on the stability number, the fractional chromatic number and the Hoffman bound. We also investigate a new bound based on a 1-to-1 mapping between VCPs and Stable Sets Problems. The mapping has been originally proposed in [10], for the VCPs and the Max Coloring Problem. In [9], the transformation is extended to the Equitable Coloring Problem and the Bin Packing Problem with Conflicts. These problems are then directly solved by reformulating them as Maximum Weighted Stable Set Problems on an associated graph (plus additional constraints). To the best of our knowledge, using the mapping instead to derive bounds for the chromatic number has not been done yet.

The remainder of the chapter is organized as follows. In Section 2.2, we recall DSATUR and present different vertex selection rules. In Section 2.3, we present and computationally compare the VCP lower bounds. In Section 2.4, we introduce the Reduced Graph used to compute VCP lower bounds starting from a partial coloring. In Section 2.5, we discuss potential improvements for the reduced graph. In Section 2.6, we introduce new vertex selection rules. In Section 2.7, we discuss extensive computational results and depict further possible lines of research on the topic.

State of the art: DSATUR-based Branch-and-Bound algorithm

DSATUR

In this section, we recall the DSATUR algorithm. We base our review on the notation offered by San Segundo [51]. The algorithm is based on DSATUR$_h$ (see Brélaz [4]) which is a greedy heuristic algorithm where each vertex $u \in V$ is iteratively colored with a feasible color. A color is considered feasible for a vertex u if none of the vertices in its neighborhood $N(u)$ have already taken that color.

Given a partial coloring $\tilde{C}$ and a vertex $u \in V$, the *saturation degree* DSAT$(u, \tilde{C})$ corresponds to the number of different colors in $N(u)$. At each iteration of DSATUR$_h$, the vertex with the highest DSAT value is colored until a feasible heuristic coloring of the entire graph G is obtained, the number of used colors is then a valid upper bound for $\chi(G)$.

An exact Branch-and-Bound algorithm can be derived from DSATUR$_h$. Given a partial coloring and an uncolored vertex $u \in V$, instead of fixing its color in a greedy way, a branching tree is created by coloring u with all the feasible colors already used in the partial coloring plus a new one. At each node of this branching tree, we are given a partial coloring $\tilde{C}$ with $\tilde{k}$ colors, an upper bound (UB) and a lower bound (LB) on $\chi(G)$. Trivially, $\tilde{k}$ can be used as a lower bound for $\chi_{\tilde{C}}(G)$, i.e, the chromatic number of G partially colored by $\tilde{C}$. Another LB can be found at the root node of the Branch-and-Bound algorithm by determining a clique in G. This LB is typically never updated in the classical implementation of DSATUR.

In Algorithm 1 and Algorithm 2, we give the pseudo-code of DSATUR. Precisely, Algorithm 1 receives in input the graph G to color, computes a maximal clique using a heuristic algorithm (a maximum clique if possible) in order to initialize LB and it produces in output the optimal coloring C^* of value $\chi(G)$. The algorithm used to compute the maximal clique is denoted by MaxClique(G) in the pseudo-code of Algorithm 1. In Algorithm 2, the mechanism to create the children nodes is described, i.e, after an uncolored vertex is selected and if $\max\{\hat{k}, LB\} < UB$ (where $\hat{k}$ is the number of different color in $\hat{C}$), up to $\tilde{k}+1$ children nodes are created by coloring the selected vertex with all the feasible colors in $\tilde{C}$ plus a new one. In case all vertices are colored and if $\tilde{k} < UB$, the best incumbent solution value and the best solution are updated respectively.

Algorithm 1: DSATUR

```
  **Data:** $G = (V, E)$: graph to color
  **Result:** optimal coloring $C^*$ of value $\chi(G)$
  $LB \leftarrow$ MaxClique($G$), $UB \leftarrow n$ ;
  DSATUR($\emptyset$);
  **return** $C^*$
```

Vertex selection rules for DSATUR

The basic Vertex Selection Rule (VSR), proposed in Brélaz [4], consists of coloring the vertex with the maximum DSAT value, thus it minimizes the number of children nodes.

During the execution of DSATUR, it often happens that many different vertices share the same maximum DSAT value, i.e., creating possible ties. The following tie-breaking rules have been introduced in the literature.

Algorithm 2: DSATUR($\tilde{C}$)

```
if all the vertices are colored then
    if $\tilde{k} < UB$ then
        $C^* \leftarrow \tilde{C}$, $UB \leftarrow \tilde{k}$;
    end
else
    select an uncolored vertex $v$;
    for every feasible color $i \in \tilde{C}$ plus a new one do
        $\widehat{C} \leftarrow \tilde{C}$, add $v$ in $\widehat{V}_i$;
        if $\max\{\widehat{k}, LB\} < UB$ then
            DSATUR($\widehat{C}$);
        end
    end
end
```

DSAT rule In Brélaz [4], the set of candidate vertices is $T = \max\limits_{u \in V \setminus \tilde{V}} \{DSAT(u, \tilde{C})\}$. If T has more than one vertex, ties are broken considering the maximum degree or, in case of further ties, the lexicographical order is used. The computational complexity of this rule is $O(n^2)$.

Sewell rule In Sewell [54], the original set of candidates is the same, i.e., $T = \max\limits_{u \in V \setminus \tilde{V}} \{DSAT(u, \tilde{C})\}$. The function $same(u, v)$ is defined as follows:

$$same(u, v) = \begin{cases} |F(u) \cap F(v)| \text{ if } u, v \in V \setminus \tilde{V} \\ 0 \text{ otherwise.} \end{cases}$$

$F(u)$ is the set of feasible colors for the vertex u in partial coloring $\tilde{C}$. $same(u, v)$ represents the number of common available colors for a given pair of vertices u and v.

Sewell chooses to break ties selecting the vertex maximizing the number of common available colors in the neighborhood of uncolored vertices. The selected vertex w is such that

$$w = \max_{u \in T}\Big(\sum_{\substack{v \in V \setminus \tilde{V} \\ u \neq v}} same(u, v)\Big).$$

The selected vertex w is minimizing the number of potential child nodes in the vertices in its neighborhood $N(w)$, allowing a reduction of the branching tree. Again, in case of further ties, the lexicographical order is used. The computational complexity of this rule is at worst $O(n^3)$. Using the Sewell rule is more time consuming than the DSAT rule especially in the early stages of the branching tree. Sewell reports that the Sewell rule is more efficient than the DSAT rule only when combined with a good starting upper bound.

PASS rule In San Segundo [51], the Sewell rule is extended considering only uncolored vertices that are also candidates in the tie. The selected vertex w is such that

$$w = \max_{u \in T}(\sum_{\substack{v \in T \\ u \neq v}} same(u, v)).$$

Again, in case of further ties, the lexicographical order is used. This makes the computation of the PASS rule faster on average since the computation of the rule is done on a smaller set of vertices (T instead of $V \backslash \tilde{V}$). The PASS rule is more precise than the Sewell rule because it is computed on the candidate set of vertices T, i.e., the vertices that will be colored in priority in the next steps of the algorithm. The reduction in potential child nodes in the vertices in its neighborhood $N(w)$ is more likely to be efficient (see [51] for further details).

The fastest DSATUR algorithm is the one proposed by San Segundo [51]. Since its implementation is not available, we re-implement the same algorithm following the description given in [51]. In particular:

- the same preprocessing is implemented. Dominated vertices are removed (a vertex $u \in V$ is dominated iff there exists a vertex $v \in V$ such that $N(u) \subseteq N(v)$). Vertices $v \in V$ such that $|N(v)| < LB$ are also removed;
- in [51] an exact algorithm (see [52]) for the MCP is used. Since determining the maximum clique can be computationally expensive, a time limit of 5 seconds is imposed in [51]. We solve the MCP using Cliquer [46] (since the implementation of the algorithm used in [51] is not available) and the same time limit. The vertices in this clique are colored before starting DSATUR;
- the vertex ordering is similar to the one used in [51], i.e., vertices belonging to the initial clique (LB) are ordered first in V and then the remaining vertices are sorted by non-increasing degree order ($|N(u)|, u \in V$).

In Table 2.1, we show that our implementation has the same computational efficiency than DSATUR [51] and it outperforms current leading exact column generation based algorithms.

We test the same benchmark of random instances introduced in [51]. Each line in Table 2.1 contains average values over 50 instances of a given vertex number n and a given density $d = \frac{2 \cdot m}{n \cdot (n-1)}$. The column identified by # reports the number of instances with the same value of n and d. For $n = 75, 80$ only 49 instances are available for each value of density and only 48 for $n = 70$ and $d = 0.7$. This entire benchmark of instances can be downloaded at `http://www.lamsade.dauphine.fr/coloring`. In Table 2.1, we report the tests of 3 exact algorithms, i.e., 2 different implementations of DSATUR [51] and 2 different Branch-and-Price algorithms. The table contains the following information imposing a common time limit of 1200 seconds:

- UB: the average number of colors (the chromatic number or an upper bound in case time limit is reached).
- LB: the average LB obtained (for MMT-BP [36], LB represents the bound obtained at the root node while, for Exactcolors, LB represents the global final bound).
- *nodes*: the average number of explored nodes (not available for DSATUR [51] and MMT-BP [36]). The averages are computed only for instances solved to proven optimality.
- *time*: the average computation time (the time limits are not considered in the averages).
- *fails*: the number of instances that cannot be solved within the time limit.

The first and the second group of columns are the results of 2 different implementations of DSATUR [51]: the first group reports the results directly taken from [51] while the second group reports the results of our re-implementation of the algorithm. Then we report the results of 2 Branch-and-Price algorithms. The first one is the Branch-and-Price algorithm described in [36] (called *MMT-BP*) and the second one is the Branch-and-Price algorithm described in [23] (called *Exactcolors*). Since the implementation of MMT-BP is not available, in Table 2.1 we report the results taken from [51]. For Exactcolors instead, the code is available at `https://github.com/heldstephan/exactcolors` and we test this algorithm on our machine (using default values for all the parameters).

			DSATUR							Branch-and-Price algorithm								
Instance			[51]			Reimplementation [16]				MMT-BP [36]				Exactcolors [23]				
n	d	#	UB	time	fails	UB	time	nodes	fails	LB	UB	time	fails	LB	UB	time	nodes	fails
60	0.1	50	4.0	0.0	0	4.0	0.0	63	0	4.0	4.0	0.2	0	4.0	4.1	0.1	3,639	4
60	0.2	50	5.5	0.0	0	5.5	0.0	1,724	0	5.1	5.5	149.2	3	5.5	5.6	57.1	4,022	4
60	0.3	50	7.0	0.0	0	7.0	0.0	6,986	0	7.0	7.0	34.3	1	7.0	7.9	286.8	25,256	35
60	0.4	50	8.9	0.1	0	8.9	0.0	51,035	0	8.3	8.9	203.9	2	8.9	10.0	85.4	29,131	41
60	0.5	50	10.7	0.2	0	10.7	0.1	212,856	0	10.1	10.7	106.0	1	10.7	11.8	150.5	25,552	33
60	0.6	50	12.9	0.2	0	12.9	0.2	282,743	0	12.5	12.9	24.9	0	12.9	14.2	347.2	29,407	36
60	0.7	50	15.6	0.2	0	15.6	0.2	214,544	0	15.3	15.6	9.7	0	15.6	16.6	186.5	17,808	21
60	0.8	50	19.1	0.1	0	19.1	0.1	127,348	0	19.0	19.1	2.3	0	19.1	19.1	72.1	7,897	0
60	0.9	50	25.7	0.0	0	25.7	0.0	5,457	0	25.7	25.7	0.2	0	25.7	25.7	4.2	725	0
70	0.1	50	4.0	0.0	0	4.0	0.0	139	0	4.0	4.0	0.2	0	4.0	4.1	60.5	4,924	7
70	0.2	50	6.0	0.0	0	6.0	0.0	2,717	0	5.8	6.0	36.5	0	6.0	6.8	0.8	15,644	39
70	0.3	50	7.8	0.2	0	7.8	0.1	133,275	0	7.1	7.8	674.0	20	7.8	9.0	105.6	19,287	47
70	0.4	50	9.7	1.3	0	9.7	1.0	1,395,809	0	9.1	9.7	523.5	14	9.7	11.2	429.2	21,300	43
70	0.5	50	11.8	4.3	0	11.8	3.7	4,686,463	0	11.2	11.8	487.0	13	11.8	13.6	258.3	22,978	43
70	0.6	50	14.1	6.5	0	14.1	5.5	6,456,304	0	13.6	14.1	174.2	4	14.1	16.1	600.3	24,455	39
70	0.7	48	17.2	11.1	0	17.2	9.8	10,434,712	0	16.9	17.2	63.9	0	17.2	18.8	158.3	15,259	25
70	0.8	50	21.4	2.0	0	21.4	2.3	2,283,368	0	21.2	21.4	14.4	0	21.4	21.8	182.7	29,335	8
70	0.9	50	28.5	0.1	0	28.5	0.2	175,378	0	28.5	28.5	0.5	0	28.5	28.5	18.4	2,667	0
75	0.1	49	4.0	0.0	0	4.0	0.0	290	0	4.0	4.0	17.9	0	4.0	4.1	74.4	3,108	5
75	0.2	49	6.0	0.0	0	6.0	0.0	7,401	0	6.0	6.0	40.2	1	6.0	7.1	40.8	14,137	46
75	0.3	49	8.0	0.2	0	8.0	0.1	207,328	0	7.5	8.0	202.2	12	8.0	9.4	42.2	16,213	48
75	0.4	49	10.0	5.3	0	10.0	4.3	5,561,378	0	9.5	10.0	398.2	11	10.0	12.0	-	18,997	49
75	0.5	49	12.1	27.6	0	12.1	23.6	27,595,210	0	11.8	12.1	182.1	4	12.1	14.4	802.7	20,161	41
75	0.6	49	14.9	56.3	0	14.9	49.0	53,590,843	0	14.3	14.9	543.0	13	14.9	17.2	337.6	19,980	41
75	0.7	49	18.0	59.7	0	18.0	50.4	50,638,728	0	17.6	18.0	146.2	2	18.0	19.6	264.0	13,849	24
75	0.8	49	22.4	13.4	0	22.3	16.5	15,326,140	0	22.1	22.4	36.3	0	22.3	23.2	253.7	34,586	13
75	0.9	49	1.0	1.2	0	29.9	2.6	2,376,532	0	29.9	1.0	23.3	0	29.9	30.0	74.6	11,929	1
80	0.1	49	4.3	0.0	0	4.3	0.0	1,035	0	4.0	4.3	372.8	10	4.3	4.6	87.3	3,609	11
80	0.2	49	6.3	0.1	0	6.3	0.1	95,440	0	6.0	6.3	431.3	15	6.1	7.4	1161.8	11,427	48
80	0.3	49	8.2	5.7	0	8.2	3.5	4,690,852	0	7.9	8.2	360.8	11	8.1	10.0	-	13,335	49

Table 2.1: Comparison between exact algorithms on the benchmark of instances introduced in [51]

In order to compare the computation times obtained on our machine and the machine used in [51], we execute the benchmark algorithm `dfmax` achieving the following results: the *user times* are 0.0, 0.18, 1.1 and 4.24 for instances r200.5, r300.5, r400.5 and r500.5 respectively. Comparing our results to the results reported in [51], we conclude that our machine is 1.28 times faster than their machine. Accordingly we scale the computation time taken from [51] in Table 2.1.

Table 2.1 shows that our reimplementation of DSATUR slightly outperforms the implementation in [51]. This is due to a careful re-implementation of all the data structures and functions of the algorithm. Our code is available at `www.lamsade.`

`dauphine.fr/coloring`. As far as the comparison between DSATUR and the Branch-and-Price algorithms is concerned, we obtain the same results as in [51], i.e., DSATUR is the best algorithm for this benchmark of instances.

An important class of instances for the VCP is the DIMACS benchmark library. We test our reimplementation on this entire testbed obtaining, also in this case, a similar computational behaviour of DSATUR [51]. For this set of instances, the Branch-and-Price algorithms remain the best approaches. More details on the DIMACS benchmark library are given in the Appendix of this manuscript.

In the following, since DSATUR is the best algorithm for random instances, we decide to focus our analysis on this testbed only.

Lower bounds for the Vertex Coloring Problem

In this section we review the classical lower bounds for the VCP. In addition we present a new bound based on a 1-to-1 mapping between VCPs and Stable Set Problems. For each bound we discuss the framework used to compute it and its computational complexity. The focus of this chapter is to explore the idea of using these lower bounds to speed up the convergence of DSATUR. Accordingly not only the strength of these bounds is important but also the computation time necessary to obtain them. Thus, we conclude this section with an extensive computational comparison with a special attention on their potential impact on the performances of DSATUR.

Lower bounds review

Clique number $\omega(G)$. The following holds:

$$\chi(G) \geq \omega(G). \tag{2.1}$$

This lower bound comes from the fact that in any clique all vertices should have different colors. Trivially any heuristically found clique of size $\omega^h(G)$ also provides a valid lower bound for the VCP ($\chi(G) \geq \omega^h(G)$). In our computation tests, $\omega^h(G)$ corresponds to the value of the heuristic clique produced by DSATUR$_h$. Computing the

Benchmark	DIMACS		Random instances [51]	
Algorithm	Cliquer [46]	MWSS [23]	Cliquer [46]	MWSS [23]
Instances solved	94%	94%	100%	100%
Fastest computing time	36%	3%	16%	31%

Table 2.2: Comparison between Cliquer [46] and MWSS [23] for computing $\omega(G)$.

clique number is NP-hard (see Garey and Johnson [19]) but in practice very effective exact solvers are available in the literature. Another relevant algorithm for computing $\omega(G)$ is MWSS [23].

We address the interested reader to Wu and Hao [59] for a recent survey on the topic. Table 2.2 summarizes the respective efficiency of these two algorithms on the DIMACS instances and the random instances [51]. For the first line, each entry in the table is the percentage of solved instances. For the second line, each entry in the table is the percentage of instances for which the algorithm in the corresponding column is faster than the other. When the time is substantially the same (i.e. within an epsilon of 0.001) the instance is not part of any entry, hence the numbers do not add up to a hundred percents. According to these results, Cliquer is more efficient for DIMACS instances while MWSS is more efficient for random instances [51]. The choice of the algorithm computing $\omega(G)$ will be set accordingly for all further computations.

Lower bound based on the stability number $\chi_\alpha(G)$. The following holds:

$$\chi(G) \geq \chi_\alpha(G) = \left\lceil \frac{n}{\alpha(G)} \right\rceil. \tag{2.2}$$

Since a coloring is a partition into stable sets, the best we can hope for is having all stable sets of maximum size (see Schrijver [50] for further details). Computing $\alpha(G)$ is an NP-hard problem (see Garey and Johnson [19]). Again, Table 2.3 summarizes the respective efficiency of these two algorithms on the DIMACS instances and the random instances [51]. See Table 2.2 for an explanation of the entries. For all further computations, Cliquer will be chosen to compute $\chi_\alpha(G)$ since it is the fastest.

Lovász Theta number $\vartheta(\bar{G})$. The Lovász Theta number, denoted as $\vartheta(\bar{G})$, on the complementary graph $\bar{G}$, is a lower bound for VCP proposed in Lovász [34]. The following holds (sandwich theorem):

Benchmark	DIMACS		Random instances [51]	
Algorithm	Cliquer [46]	MWSS [23]	Cliquer [46]	MWSS [23]
Instances solved	74%	74%	100%	100%
Fastest computing time	23%	13%	0%	2%

Table 2.3: Comparison between Cliquer [46] and MWSS [23] for computing $\chi_\alpha(G)$.

Theorem 2 (Sandwich theorem, Lovász [34])

$$\chi(G) \geq \vartheta(\bar{G}) \geq \omega(G). \tag{2.3}$$

Thus, we also have $\chi(G) \geq \vartheta^*(\bar{G}) = \lceil \vartheta(\bar{G}) \rceil$. This bound is based on a relaxation of the maximum stable set problem and can be computed as follows. Consider a set of variables x_u assigned to every vertex u of $\bar{G}$. We can model the maximum stable set problem with quadratic constraints by the following integer program:

$$\max \sum_{i \in V} x_i \tag{2.4}$$

$$x_i x_j = 0, \qquad (i,j) \in \bar{E}, \tag{2.5}$$

$$x_i \in \{0,1\}, \qquad i \in V. \tag{2.6}$$

Let $\bar{x}_i = 2x_i - 1$, i.e. $x_i = \frac{\bar{x}_i+1}{2}$. $\bar{x}_i$ is now such that $\bar{x}_i \in \{-1,1\}$. Let $x = (x_i)_{i \in V}$ and let $\mathbf{X} = \mathbf{x}\mathbf{x}^T$. Let :

$$\mathbf{Y} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}^T = \begin{pmatrix} 1 & \mathbf{x}^T \\ \mathbf{x} & \mathbf{X} \end{pmatrix}$$

The product of variables $x_i x_j$ is replaced by variable Y_{ij}, thus $Y_{ij} = x_i x_j$ and $Y_{0i} = x_i$. Replacing each Y_{0i} by $\frac{\bar{Y}_{0i}+1}{2}$, we obtain :

$$\vartheta(\bar{G}) = \max \ \frac{|V|}{2} + \frac{1}{2} \sum_{i \in V} \bar{Y}_{0i} \tag{2.7}$$

$$\bar{Y}_{ij} + \bar{Y}_{0i} + \bar{Y}_{0j} = 0, \qquad (i,j) \in \bar{E}, \tag{2.8}$$

$$Y_{ii} = 1, \qquad i \in V, \tag{2.9}$$

$$\mathbf{Y} \succeq 0. \tag{2.10}$$

Constraint (2.8) is still the edge constraint. Constraint (2.9) enforces that basic variables $Y_{0i} = 1$ or -1. Constraint (2.10) means that matrix $\mathbf{Y}$ is semi-definite positive.

The Lovász Theta number can be computed solving a Semi-Definite Program (SDP) in polynomial time. We use CSDP to solve this model and compute $\vartheta^*(\bar{G})$.

Fractional Coloring number $\chi_f(G)$. Following the notation proposed in Schrijver [50], the Fractional Coloring number $\chi_f(G)$ is the minimum value of $\lambda_1 + \cdots + \lambda_k$ with $\lambda_1, \ldots, \lambda_k \in \mathbb{R}_+$ such that there exist stable sets $S_1, \ldots, S_k$ with

$$\lambda_1 A^{S_1} + \cdots + \lambda_k A^{S_k} = 1,$$

where for any stable set S, A^S denotes the incidence vector of S in $\mathbb{R}^{|V|}$; that is for any $v \in G$:

$$A^S(v) := \begin{cases} 1 & \text{if } v \in S, \\ 0 & \text{otherwise.} \end{cases}$$

The Fractional Coloring number corresponds to the optimal solution value of the linear programming relaxation of the VCP formulation proposed by Mehrotra and Trick ([38]) and it is NP-hard to compute (see Grötschel et al. [20]). The following holds:

$$\chi(G) \geq \chi_f^*(G) = \lceil \chi_f(G) \rceil \tag{2.11}$$

It is well known that $\chi_f(G)$ provides strong VCP bounds and it requires Column Generation (CG) techniques to be computed. To obtain $\chi_f^*(G)$, we use the code available at `https://github.com/heldstephan/exactcolors`. The computational results in [23] show that this implementation is among the fastest in the literature.

Hoffman number $\chi_H(G)$. Hoffman proves that the following is a lower bound for $\chi(G)$ (see Hoffman [25]):

$$\chi(G) \geq \chi_H^*(G) = \lceil \chi_H(G) \rceil = 1 - \frac{\epsilon_{max}(H)}{\epsilon_{min}(H)} \tag{2.12}$$

where H is the adjacency matrix of G while ϵ_{max} and ϵ_{min} are the largest and the smallest eigenvalues of H respectively. To understand this bound, we need the following property. Let

$$\mathbf{A} = \begin{pmatrix} A_{1,1} & A_{1,2} & \ldots & A_{1,k} \\ A_{1,2}^T & A_{2,2} & \ldots & A_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1,k}^T & A_{2,k}^T & \ldots & A_{k,k} \end{pmatrix}$$

be a block-partitioned symmetric matrix with $k \geq 2$. Then :

$$(k-1)\lambda_{min}(\mathbf{A}) + \lambda_{max}(\mathbf{A}) \leq \sum_{i=1}^{k} \lambda_{max}(A_{i,i})$$

Let G be a k-colorable graph. The adjacency matrix $\mathbf{A}$ can be reordered as follows :

$$\mathbf{A} = \begin{pmatrix} 0 & A_{1,2} & \dots & A_{1,k} \\ A_{1,2}^T & 0 & \dots & A_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1,k}^T & A_{2,k}^T & \dots & 0 \end{pmatrix}$$

Applying the previous formula to matrix $\mathbf{A}$, we obtain :

$$(k-1)\lambda_{min}(\mathbf{A}) + \lambda_{max}(\mathbf{A}) \leq 0 \Leftrightarrow 1 - \frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})} \leq k$$

The eigenvalues can be computed in a polynomial time using the C++ LAPACK library.

A new lower bound

Cornaz and Jost [10] and Palubeckis [47] (see Section 1.4) prove a 1-to-1 correspondence between colorings in G and stable sets in an *auxiliary graph* $G_A = (V_A, E_A)$. The following theorem holds:

Theorem 3 (Cornaz and Jost [10]) *For any graph G and any acyclic orientation of its complementary graph, there is a one-to-one correspondence between the set of all colorings of G and the set of all stable sets of G_A. Moreover, for any coloring $\{V_1, \dots, V_k\}$ and its corresponding stable set $\tilde{S}$ in G_A, we have: $|\tilde{S}| + k = |V|$. In particular:*

$$\alpha(G_A) + \chi(G) = |V|.$$

To build the auxiliary graph G_A, it is necessary to define an acyclic orientation $\overrightarrow{G}$ of $\bar{G}$. Then G_A corresponds to the line-graph[1] $L(\overrightarrow{G})$ after the removal of all edges

[1]The line-graph $L(\overrightarrow{G})$ of $\overrightarrow{G}$ is defined as follows: each arc of $\overrightarrow{G}$ corresponds to a vertex of $L(\overrightarrow{G})$ and two vertices are linked by an edge in the $L(\overrightarrow{G})$ if they correspond to two adjacent arcs in $\overrightarrow{G}$.

between pairs of arcs which are simplicial[2] in $\overrightarrow{G}$. Precisely, given a simplicial pair of arcs $a = (v_i, v_j)$ and $b = (v_i, v_k)$ the corresponding arc (a, b) is removed from $L(\overrightarrow{G})$.

We now illustrate the construction of G_A using the example of Figure 2.1. The original graph consists of 5 nodes and 5 edges (part 1 of Figure 2.1). Then the acyclic orientation $\overrightarrow{G}$ is depicted in part 2 of Figure 2.1 where $(v_i, v_j) \in \overrightarrow{E}$ if $(v_i, v_j) \in \bar{E}$ and $i < j$. The next step consists of creating the line-graph $L(\overrightarrow{G})$ as depicted in part 3 of Figure 2.1. Finally the Auxiliary Graph G_A is given in part 4 of figure 2.1. Only one simplicial pair (in blue) is present in $\overrightarrow{G}$, i.e., (v_2, v_5) and (v_2, v_4), and accordingly the corresponding edge has been removed from $L(\overrightarrow{G})$. From Figure 2.1, it is clear that any vertex belonging to a stable set in G_A allows to reduce of one unity the upper bound $|V|$ on $\chi(G)$. In other words, if a vertex $(v_i v_j) \in G_A$ belongs to a stable set, it means that vertex v_j can be colored with the same color of v_i, i.e. "saving" in this manner a color. Finally for any simplicial pair of arcs (v_i, v_j) and (v_i, v_k) in $\overrightarrow{G}$, removing the arc $(v_i v_j, v_i v_k)$ in G_A reflects the fact that once v_k has been colored in the same way as v_i then also v_j can take the same color (and vice-versa).

Any upper bound $\bar{\alpha}(G_A)$ of the stability number $\alpha(G_A)$ gives us a valid lower bound for $\chi(G)$ denoted $\chi_{G_A}(G)$. The following holds:

$$\chi(G) \geq \chi_{G_A}(G) = |V| - \lfloor \bar{\alpha}(G_A) \rfloor. \tag{2.13}$$

Thanks to extensive computational tests, we identify that the best compromise between the quality of the upper bound $\bar{\alpha}(G_A)$ and the computation time can be achieved solving a relaxation of the *edge formulation* for the MSSP. The edge formulation is an ILP where $\alpha(G_A) = \max x$ over x in $STAB(G_A)$, that is, the set of vectors of $\mathbb{R}^{V_{G_A}}$ satisfying

$$x_u + x_v \leq 1 \qquad (u, v) \in E_A \tag{2.14}$$

$$x_v \in \{0, 1\} \qquad v \in V_A. \tag{2.15}$$

In order to solve this formulation, commercial ILP solvers start by computing the *fractional stable set value*, i.e. the optimal solution value of the linear programming relaxation. Then general purpose valid inequalities (*cuts*) are heuristically added in

[2]A pair of arcs $\{a, b\}$ of $\overrightarrow{G}$ is called a simplicial pair if $a = (u, v)$, $b = (u, w)$, and (v, w) or (w, v) is an arc of $\overrightarrow{G}$, for three distinct vertices u, v, w

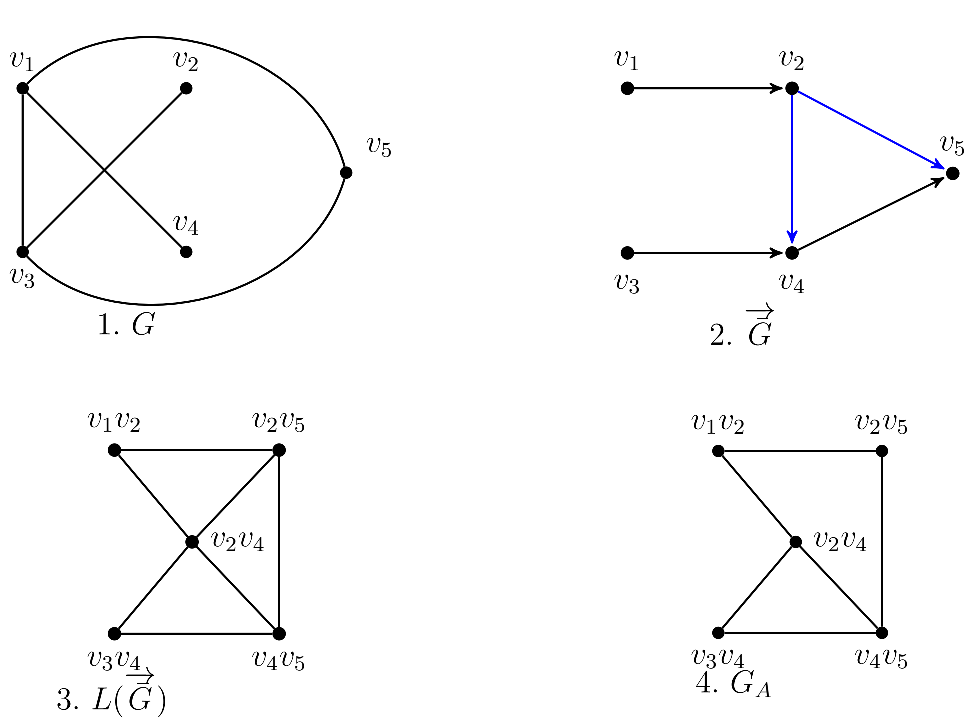

Figure 2.1: Transformation from a graph G to G_A

order to strengthen the bound before branching. The strength of this bound depends on the solver used and also on the policies designed to generate cuts at the root node. Thanks to preliminary computational tests, the bound computed by `CPLEX` version 12.6 (with default parameters and stopping the optimization at the root node) proves to be a very strong and fast bound for $\alpha(G_A)$.

We perform additional tests to check the possibility of obtaining a better compromise between strength and computing time, tuning the parameters of `CPLEX`. Thanks to extensive computational experiments, we identify that the only effective families of cuts are the *Clique Cuts*, the *Zero-half Cuts* and the *Gomory Fractional Cuts*. Since the cuts are heuristically added, repeating the generation several times can potentially produce better bounds. To check if better bounds are possible, we repeat the default cut generation of `CPLEX` up to 5 times. More than 90% of the times, the best bound value is obtained with only one iteration. We conclude that it is not worth to repeat the cut generation more than once. We finally try to limit the amount of time of the cut generation of `CPLEX`, but the quality of the bound largely deteriorates. Summarizing, the best `CPLEX` parameter configuration setting to compute χ_{G_A} is default, switching off the generation of all families of cuts except the *Clique Cuts*, the *Zero-half Cuts* and the *Gomory Fractional Cuts*. Finally, in order to avoid rounding errors possibly due to numerical imprecisions, we use the following formula to compute $\lfloor\bar{\alpha}(G_A)\rfloor = \lfloor(\kappa + \epsilon)\rfloor$, where $\epsilon = 10^{-3}$ and κ is the upper bound value returned by `CPLEX`. Since we are only interested in the upper bound value, we switch off all the routines of `CPLEX` to compute heuristic solutions.

The quality and the computing time of this bound is discussed in the next section and compared to the other bounds.

Comparison between lower bounds

To compare and test the lower bounds, we use the same benchmark of random instances introduced in San Segundo [51] and already used in Section 2.2. Each line in Table 2.4 reports average results in the same fashion of Table 2.1. Table 2.4 presents the average bound values and the average computation times for χ_f^*, χ_{G_A}, ϑ, ω, χ_α, χ_H^* and ω^h.

In Table 2.4, we computationally prove that the strongest lower bound values are provided by χ_{G_A} and χ_f^*. On the other side, these two bounds are also the most time consuming ones. Among the fast but weaker lower bounds, ω clearly dominates χ_α, χ_H^* and ω^h in terms of lower bound values. ϑ is the slowest bound and only the third in terms of value which makes it very impracticable. As far as the strongest bounds are concerned, χ_{G_A} is equal to χ_f^* in most of the instances and, in the remaining cases, the difference between the values does not exceed 1.

According to the construction of graph G_A, the more dense the graph is the faster the bound is computed, since the number of nodes $|V_A|$ of G_A is equal to the number of edges of $\bar{G}$. While it gets faster, it preserves its quality and, accordingly, its pruning potential once included in DSATUR.

From these results we selected three bounds to inject into DSATUR:

- χ_f^* which is among the strongest bounds and nearly the fastest bound,
- χ_{G_A} which is among the strongest bounds,
- ω which is has the strongest value among the fastest bounds.

Instance			Average Values							Average Times						
n	d	#	χ_f^*	χ_{G_A}	ϑ^*	ω	χ_α	χ_H^*	ω^h	χ_f^*	χ_{G_A}	ϑ^*	ω	χ_α	χ_H^*	ω^h
60	0.1	50	4.00	4.00	3.98	3.36	2.00	2.00	1.32	0.03	1.56	2.54	0.00	0.00	0.00	0.00
60	0.2	50	5.06	5.00	4.90	4.46	3.18	2.98	1.60	0.07	3.24	1.83	0.00	0.00	0.00	0.00
60	0.3	50	6.98	6.18	6.04	5.48	4.74	3.34	1.94	0.08	2.77	1.29	0.00	0.00	0.00	0.00
60	0.4	50	8.28	7.96	7.12	6.46	5.80	4.00	1.92	0.07	2.01	0.90	0.00	0.00	0.00	0.00
60	0.5	50	10.12	9.98	9.00	8.02	7.02	4.88	2.42	0.06	1.32	0.56	0.00	0.00	0.00	0.00
60	0.6	50	12.50	12.16	10.94	9.70	8.76	5.80	2.86	0.05	0.62	0.33	0.00	0.00	0.00	0.00
60	0.7	50	15.30	15.24	13.68	12.34	11.20	6.50	3.54	0.05	0.26	0.18	0.00	0.00	0.00	0.00
60	0.8	50	19.04	19.10	17.56	16.14	13.92	8.02	5.80	0.04	0.11	0.09	0.00	0.00	0.00	0.00
60	0.9	50	25.66	25.66	24.62	23.56	18.30	11.18	9.04	0.01	0.00	0.05	0.00	0.00	0.00	0.00
70	0.1	50	4.00	4.00	4.00	3.60	2.00	2.10	2.10	0.07	4.24	5.75	0.00	0.00	0.00	0.00
70	0.2	50	5.84	5.04	5.06	4.60	3.68	3.00	2.04	0.16	9.49	4.19	0.00	0.00	0.00	0.00
70	0.3	50	7.10	6.62	6.10	5.60	4.98	3.68	2.22	0.14	6.70	3.07	0.00	0.00	0.00	0.00
70	0.4	50	9.10	8.34	7.90	7.02	6.74	4.26	2.46	0.11	4.94	2.04	0.00	0.00	0.00	0.00
70	0.5	50	11.18	10.62	9.46	8.34	7.82	5.04	2.60	0.09	3.10	1.24	0.00	0.00	0.00	0.00
70	0.6	50	13.64	13.18	11.64	10.22	10.00	6.04	2.94	0.08	1.53	0.71	0.00	0.00	0.00	0.00
70	0.7	48	16.88	16.73	14.63	12.90	11.94	7.19	3.42	0.07	0.47	0.37	0.00	0.00	0.00	0.00
70	0.8	50	21.22	21.30	19.00	17.32	14.96	8.80	5.04	0.06	0.17	0.16	0.00	0.00	0.00	0.00
70	0.9	50	28.50	28.52	27.02	25.60	19.76	12.84	8.94	0.02	0.02	0.07	0.00	0.00	0.00	0.00
75	0.1	49	4.00	4.00	4.00	3.61	2.10	2.10	1.31	0.09	7.44	8.48	0.00	0.00	0.00	0.00
75	0.2	49	6.00	5.08	5.06	4.59	3.78	3.00	1.63	0.20	15.12	6.31	0.00	0.00	0.00	0.00
75	0.3	49	7.53	6.86	6.37	5.73	5.06	3.88	1.61	0.18	9.75	4.53	0.00	0.00	0.00	0.00
75	0.4	49	9.47	8.59	8.00	7.02	6.63	4.06	2.16	0.14	7.41	2.91	0.00	0.00	0.00	0.00
75	0.5	49	11.80	10.98	9.86	8.61	8.59	5.00	2.45	0.12	4.84	1.80	0.00	0.00	0.00	0.00
75	0.6	49	14.29	13.92	12.02	10.41	9.98	5.98	2.71	0.10	2.24	1.01	0.00	0.00	0.00	0.00
75	0.7	49	17.63	17.49	14.96	13.33	12.47	7.37	3.49	0.09	0.67	0.51	0.00	0.00	0.00	0.00
75	0.8	49	22.10	22.08	19.59	17.67	15.92	9.02	5.16	0.08	0.21	0.22	0.00	0.00	0.00	0.00
75	0.9	49	29.92	29.92	28.08	26.39	19.86	13.53	7.69	0.03	0.03	0.09	0.00	0.00	0.00	0.00
80	0.1	49	4.02	4.00	4.00	3.76	2.22	2.06	2.02	0.13	11.24	11.51	0.00	0.01	0.00	0.00
80	0.2	49	6.00	5.14	5.14	4.82	3.98	3.00	2.14	0.25	27.63	8.94	0.00	0.00	0.00	0.00
80	0.3	49	7.94	6.98	6.69	5.84	5.41	3.92	2.06	0.22	14.36	6.24	0.00	0.00	0.00	0.00

Table 2.4: Comparison between different lower bounds for random instances.

In the appendix, we report the lower bounds for the DIMACS instances. The results are similar to the ones obtained for random instances. We can observe that the overall quality of χ_α, χ_H^* and ω^h is very poor, while χ_{G_A} and χ_f^* provides the best lower bound for many instances. For dense instances, χ_{G_A} and χ_f^* are particularly efficient and χ_{G_A} dominates χ_f^* in terms of computation time. Unfortunately just a couple of instances of density $d \geq 0.9$ are present in the DIMACS benchmark (see the appendix for further details).

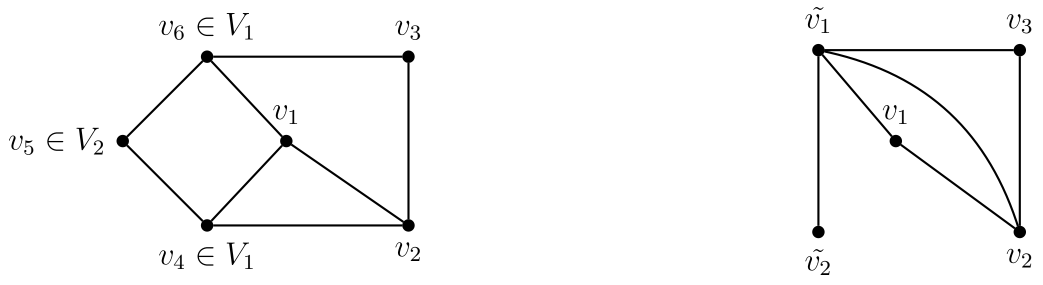

Figure 2.2: A partially colored graph G and the Reduced Graph $G^{\tilde{C}}$

Reduced graph and the improved DSATUR-based Branch-and-Bound algorithm

In order to make lower bounds dependent on a partial coloring $\tilde{C}$ obtained during the execution of DSATUR, we introduce a new graph. This *Reduced Graph* $G^{\tilde{C}} = (V^{\tilde{C}}, E^{\tilde{C}})$ is composed of the sub-graph of G induced by the uncolored vertices plus $\tilde{k}$ vertices, one for each color. Each new vertex $\tilde{v}_i$, representing color i, is connected to all the uncolored neighbours of the vertices of V_i and to all the others $\tilde{k}-1$ new vertices. Thus, the subgraph of $G^{\tilde{C}}$ induced by the $\tilde{k}$ new vertices is a clique. The reduced graph becomes smaller increasing the number of colored vertices thus also the lower bounds become easier to compute. An example using a partially colored graph of 6 vertices is given in Figure 2.2, where two colors (1 and 2) are used and three vertices are uncolored. The Reduced Graph has 5 vertices, two representing the classes of colors plus the three original uncolored vertices.

Recalling that we denote by $\chi_{\tilde{C}}(G)$ the chromatic number of G partially colored by $\tilde{C}$, the following holds:

Proposition 1 $\chi_{\tilde{C}}(G) = \chi_{\tilde{C}}(G^{\tilde{C}})$

Proof.

Any feasible coloring $\hat{C} = \{\hat{C}_1, \dots, \hat{C}_k\}$ in G containing the coloring $\tilde{C}$ (inducing $k \geq \tilde{k}$) can be mapped into a unique feasible coloring $\{\hat{C}'_1, \dots, \hat{C}'_k\}$ in $G^{\tilde{C}}$ using the same number of colors k. For each color i, $i = 1, \dots, k$, two cases arise: if $\hat{C}_i \subseteq V \setminus \tilde{C}$, then $\hat{C}'_i \leftarrow \hat{C}_i$ (since the subgraphs induced by $\hat{C}_i$ are equivalent in G and $G^{\tilde{C}}$), otherwise it exists a unique stable set j such that $\tilde{C}_j \subseteq \hat{C}_i$, then $\hat{C}'_i \leftarrow \{\tilde{v}_j\} \cup \{\hat{C}_i \setminus \tilde{C}_j\}$ with

$\tilde{v}_j$ representing $\tilde{C}_j$ in $G^{\tilde{C}}$. By construction, $\hat{C}'_i$ is a stable set $G^{\tilde{C}}$. Any feasible coloring $\{\hat{C}'_1, \ldots, \hat{C}'_k\}$ in $G^{\tilde{C}}$ can be mapped into a unique feasible coloring $\{\hat{C}_1, \ldots, \hat{C}_k\}$ containing the coloring $\tilde{C}$ in G. For each color i, $i = 1, \ldots, k$, two cases arise: if $\hat{C}'_i \subseteq V \setminus \tilde{C}$, then $\hat{C}_i \leftarrow \hat{C}'_i$, otherwise a unique vertex $\tilde{v}_j$ representing $\tilde{C}_j$ belongs to $\hat{C}'_i$ (let us recall that the $\tilde{k}$ vertices representing $\tilde{C}$ in $G^{\tilde{C}}$ form a clique), and then $\hat{C}_i \leftarrow \tilde{C}_j \cup \{\hat{C}'_i \setminus \{\tilde{v}_j\}\}$. By construction, $\hat{C}_i$ is a stable set in G. □

Trivially, we have: $\chi_{\tilde{C}}(G^{\tilde{C}}) = \chi(G^{\tilde{C}})$. Thus, from lemma 1, a lower bound denoted by $LB^{\tilde{C}}$ for $\chi(G^{\tilde{C}})$ is a lower bound for $\chi_{\tilde{C}}(G)$.

We present now the *improved DSATUR-based Branch-and-Bound algorithm.* The new DSATUR algorithm, denoted DSATUR$_{LB}$ is obtained by replacing in Algorithm 1 the call to Algorithm 2 by a call to the new Algorithm 3 with the corresponding lower bound LB. According to Section 2.3.3, we chose three promising bounds to incorporate into the new DSATUR$_{LB}$ algorithm: $\omega(G)$, $\chi_{G_A}(G)$ and $\chi^*_f(G)$. The key element of the new algorithm is the introduction of either $\chi^*_f(G^{\tilde{C}})$, $\chi_{G_A}(G^{\tilde{C}})$ or $\omega(G^{\tilde{C}})$ as the lower bound, computed thanks to the reduced graph $G^{\tilde{C}}$ at nodes of the branching tree.

Since updating the lower bound can be computationally expensive, we propose strategies to compute it only in "promising" nodes of the branching tree. The term "promising" is linked to two different aspects. Clearly, the chances to prune a node are higher when the incumbent solution value UB is close to the chromatic number $\chi(G)$. Similarly, the lower bound is more likely to prune a node when the difference between the node lower bound ($\tilde{k}$) and upper bound (UB) is small. Accordingly, we propose a function $\phi_{u,g}(\tilde{C})$ which controls the computation of the bound according to two predefined parameters u and g. The first parameter u corresponds to the number of times the incumbent solution value UB has been updated and it is then a measure of the quality of the current incumbent value UB. The second parameter g corresponds to the gap between the incumbent value UB and the lower bound $\tilde{k}$ and it is related to the probability that the bound can prune a specific node. In Algorithm 3, the lower bound $LB(\tilde{C})$ associated to the coloring $\tilde{C}$ of a given node is initialized with the maximum between the global lower bound LB and the current number of colors $\tilde{k}$. If the function $\phi_{u,g}(\tilde{C})$ returns true, the computation of the bound is performed in order to improve the value of $LB(\tilde{C})$. Then, if $LB(\tilde{C})$ is greater than the best incumbent solution value UB, the children nodes of the current node are pruned. The rest of the algorithm remains unchanged.

During an execution of DSATUR$_{LB}$, the following situation can occur. Two reduced

Algorithm 3: DSATUR$_{LB}(\tilde{C})$

```
if all the vertices are colored then
    if $\tilde{k} < UB$ then
        $C^* \leftarrow \tilde{C}$, $UB \leftarrow \tilde{k}$;
    end
else
    $LB(\tilde{C}) \leftarrow \max\{LB, \tilde{k}\}$;
    if $\phi_{u,g}(\tilde{C}) =$ true then
        $LB(\tilde{C}) \leftarrow \max\{LB(\tilde{C}), \chi_{G_A}(G^{\tilde{C}})\}$;
        if $LB(\tilde{C}) \geq UB$ then
            return
        end
    end
    select an uncolored vertex $v$;
    for every feasible color $i \in \tilde{C}$ plus a new one do
        $\widehat{C} \leftarrow \tilde{C}$, add $v$ in $\widehat{V}_i$;
        if $\max\{\widehat{k}, LB(\tilde{C})\} < UB$ then
            DSATUR$_{LB}(\widehat{C})$;
        end
    end
end
```

graphs $G^{\tilde{C}_1}$ and $G^{\tilde{C}_2}$ associated with two branching nodes and two partial colorings $\tilde{C}_1$ and $\tilde{C}_2$ are isomorphic and thus any VCP lower bound on $G^{\tilde{C}_1}$ holds the same value in $G^{\tilde{C}_2}$. In these cases, the same lower bound for the same graph is computed twice. While the complexity of graph isomorphism is an open question, it is possible to recognize two similar graphs $G^{\tilde{C}_1}$ and $G^{\tilde{C}_2}$ in specific situations in DSATUR$_{LB}$. In the next section, we present these cases.

Reduced graph and isomorphism

In this section, we study the changes between the reduced graphs associated to each nodes of the branching tree.

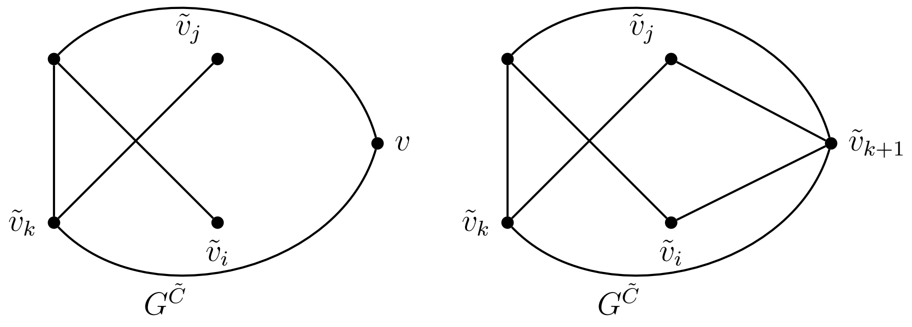

Figure 2.3: Changes between $G^{\tilde{C}}$ and $G^{\widehat{C}}$ when v is colored with a new color $k+1$

Case of a branching node and its children nodes

First let us inspect the case where two reduced graphs are respectfully associated to a given branching node and one of its children nodes. There is only one graph operation between these two consecutive reduced graphs.

Considering a branching node of DSATUR, a vertex v to be colored, a partial coloring $\tilde{C} = \{\tilde{V}_1, \ldots, \tilde{V}_{\tilde{k}}\}$ of size $\tilde{k}$ and the current reduced graph $G^{\tilde{C}}$, two cases must be distinguished for the construction of the new reduced graph $G^{\widehat{C}}$ where $\widehat{C}$ is the coloring $\tilde{C}$ with vertex v:

- Case 1 (see Figure 2.3): v is colored with a new color, i.e., $v \in \widehat{V}_{\tilde{k}+1}$. The new set $\widehat{V}_{\tilde{k}+1} = \{v\}$ is created and v is the only vertex in $\widehat{V}_{\tilde{k}+1}$ in $G^{\widehat{C}}$. Following the procedure described in Section 2.4, for each $\tilde{v}_i$, $\forall i \in \{1, \ldots, \tilde{k}\}$, edges $(v, \tilde{v}_i)$ have to be added if they do not already exist. Thus, $|\{\tilde{v}_1, \ldots, \tilde{v}_{\tilde{k}}\} \backslash N_{G^{\tilde{C}}}(v)|$ edges are added. If $\forall i \in \{1, \ldots, \tilde{k}\}$, $(v, \tilde{v}_i) \in E^{\tilde{C}}$, i.e., $\{\tilde{v}_1, \ldots, \tilde{v}_{\tilde{k}}\} \subseteq N_{G^{\tilde{C}}}(v)$, then no edges are added.
- Case 2 (see Figure 2.4): v is colored with an existing color k, i.e., $v \in \tilde{V}_k$. No edge need to be added to form a clique since $\tilde{v}_k$ is already adjacent to all vertices $\tilde{v}_i$, $\forall i \in \{1, \ldots, \tilde{k}\}$. Vertices v and $\tilde{v}_k$ are merging, thus vertex v is removed in $G^{\widehat{C}}$ (arbitrarily). Vertex $\tilde{v}_k$ gains the neighborhood $N_{G^{\tilde{C}}}(v)$ of v which means for each $u \in N_{G^{\tilde{C}}}(v)$, edge $(u, \tilde{v}_k)$ is added if it does not exist already. Thus the vertices u such that $(u, v) \in E^{\tilde{C}}$ and $(u, \tilde{v}_k) \in E^{\tilde{C}}$ are represented in $G^{\widehat{C}}$ by one edge instead of two. $|N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)|$ edges are removed to build $G^{\widehat{C}}$. If v and $\tilde{v}_k$ have no edges in common, i.e., $|N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)| = 0$, no edges are removed and $|E^{\tilde{C}}| = |E^{\widehat{C}}|$.

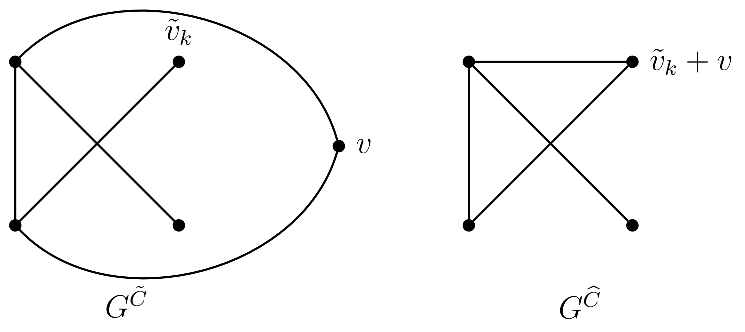

Figure 2.4: Changes between $G^{\tilde{C}}$ and $G^{\widehat{C}}$ when v is colored with an existing color k

From this study, we can identify when two consecutive reduced graphs are isomorphic. We have $G^{\tilde{C}} = G^{\widehat{C}}$ when $|\{\tilde{v}_1, \ldots, \tilde{v}_{\tilde{k}}\} \backslash N_{G^{\tilde{C}}}(v)| = 0$.

Case of two unrelated branching nodes

The second case occurs with two different reduced graphs in two unrelated branching nodes in the branching tree. Since partial colorings are designated by integer numbers in DSATUR, two different partial colorings can actually refer to the same partial colorings if the integer numbers are switched. This symmetry, as symmetry problems in general, is a known difficulty of the VCP.

The following procedure provides an easy way to detect some cases. Given a graph $G = (V, E)$, an ordering of the set of vertices σ_V, and a partial coloring $\tilde{C}$, the representative vertex of $v \in \tilde{V}_i$ is defined as $r^v_{\tilde{C}} = \min\limits_{w \in \tilde{V}_i} \sigma_V(w)$.

Proposition 2 *Given two reduced graphs $G^{\tilde{C}_1}$ and $G^{\tilde{C}_2}$, if for all vertices $v \in V$, $r^v_{\tilde{C}_1} = r^v_{\tilde{C}_2}$, then $G^{\tilde{C}_1}$ and $G^{\tilde{C}_2}$ are isomorphic.*

This proposition allows recognizing two reduced graphs associated to two similar partial colorings. Preliminary computational results show that this particular situation does not occur a significant number of times during an execution of DSATUR$_{LB}$. This procedure does not allow the algorithm to reduce the computation time or the number of nodes of DSATUR$_{LB}$.

Vertex Selection Rule for DSATUR$_{LB}$

In this section, we present new VSR aiming at improving the efficiency of DSATUR$_\omega$ and DSATUR$_{\chi_{G_A}}$. State-of-the-art VSR (see Section 2.2) are designed to reduce the number of children nodes in the branching tree of DSATUR. These VSR have no relation to the reduced graph computed at each node of the branching tree. A simple VSR could not be designed for DSATUR$_\chi$, it is one of our research line in the future. We start by presenting the new VSR for DSATUR$_\omega$, denoted VSR$_\omega$.

Vertex Selection Rule for DSATUR$_\omega$

To design VSR$_\omega$, we present a series of propositions aiming at identifying how much ω can gain in terms of absolute value from a given branching node to one of its children nodes.

Considering a branching node of DSATUR and one of its children nodes, the two following propositions give us the maximum value ω can gain from the branching node to its children node.

Proposition 3 *Considering a branching node of DSATUR, a vertex v to be colored, a partial coloring $\tilde{C} = \{\tilde{V}_1, \ldots, V_{\tilde{k}}\}$ of size $\tilde{k}$, the current reduced graph $G^{\tilde{C}}$ and the new reduced graph $G^{\widehat{C}}$, we have $\omega(G^{\widehat{C}}) \leq \omega(G^{\tilde{C}}) + \tilde{k}$.*

Proof. Let us respectively denote $\tilde{K}$ and $\widehat{K}$ the subset of vertices corresponding to $\omega(G^{\tilde{C}})$ and $\omega(G^{\widehat{C}})$. Let us suppose that $\omega(G^{\widehat{C}}) = \omega(G^{\tilde{C}}) + \tilde{k} + 1$, hence $\tilde{k} + 1$ new vertices are added to the maximum clique $\omega(G^{\widehat{C}})$. At best, each vertex $u \in \widehat{K} \backslash \tilde{K}$ is adjacent to all vertices in $\tilde{K}$ except one. For each $u \in \widehat{K} \backslash \tilde{K}$, at least one edge has to be added to make u part of $\widehat{K}$, $\tilde{k} + 1$ edges in total. Let us assume that $\{\tilde{v}_1, \ldots, \tilde{v}_k\} \cap N_{G^{\tilde{C}}}(v) = \emptyset$. If v is colored with a new color, then, as previously stated, $|\{\tilde{v}_1, \ldots, \tilde{v}_k\} \backslash N_{G^{\tilde{C}}}(v)| = |\{\tilde{v}_1, \ldots, \tilde{v}_k\}| = \tilde{k} < \tilde{k} + 1$ edges are added at most. Thus, $\tilde{k} + 1$ vertices cannot be added into $\widehat{K}$ and $\omega(G^{\widehat{C}}) \neq \omega(G^{\tilde{C}}) + \tilde{k} + 1$.

The following configuration:

- $v \in \omega(G^{\tilde{C}})$,
- $\forall i \in \{1, \ldots, \tilde{k}\}$, $(v, \tilde{v}_i) \notin E^{\tilde{C}}$,

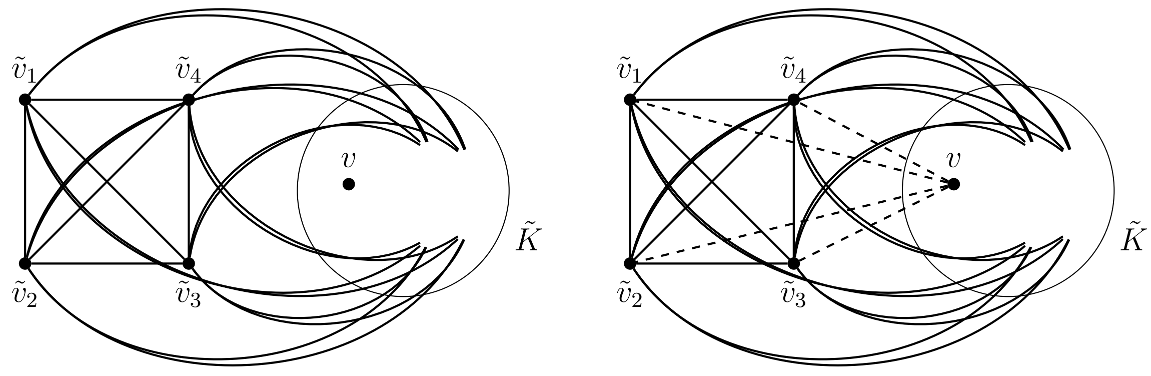

Figure 2.5: Changes between $G^{\tilde{C}}$ and $G^{\widehat{C}}$ when v is colored with a new color $k+1$

- $\forall u \in \omega(G^{\tilde{C}})$, $\forall \tilde{v}_i \in \{\tilde{v}_1, \ldots, \tilde{v}_k\}$, $u \neq v$, $(\tilde{v}_i, u) \in E^{\tilde{C}}$,

give us $\omega(G^{\widehat{C}}) = \omega(G^{\tilde{C}}) + \tilde{k}$ (see Figure 2.5). $\square$

Proposition 4 *Considering a branching node of DSATUR, a vertex v to be colored, a partial coloring $\tilde{C} = \{\tilde{V}_1, \ldots, V_{\tilde{k}}\}$ of size $\tilde{k}$, the current reduced graph $G^{\tilde{C}}$ and the new reduced graph $G^{\widehat{C}}$, we have $\omega(G^{\widehat{C}}) \leq \omega(G^{\tilde{C}}) + \lfloor \frac{\delta_{max}+1}{2} \rfloor$.*

Proof. The maximum value for $\omega(G^{\widehat{C}})$ is $\delta_{max}+1$. This value can be reached when $\tilde{k} = \lfloor \frac{\delta_{max}+1}{2} \rfloor$ and $\omega(G^{\tilde{C}}) = \lceil \frac{\delta_{max}+1}{2} \rceil$. $\tilde{k}$ cannot be greater than $\omega(G^{\tilde{C}})$ since $\{\tilde{v}_1, \ldots, \tilde{v}_k\}$ is a clique. $\square$

The computation time of ω is among the lowest of all the bounds we previously studied. Moreover, since the value of ω is weak, VSR$_\omega$ is designed to maximize the potential increase in ω in all the children nodes of a given node, thanks to the previous propositions.

Let T be the set of tied candidate vertices, a vertex maximizing the decrease of $|N_{G^{\tilde{C}}}(u) \cap N_{G^{\tilde{C}}}(\tilde{v}_i)|$ is such that:

$$v = \max_{u \in T} |\{\tilde{v}_1, \ldots, \tilde{v}_k\} \backslash N_{G^{\tilde{C}}}(u)|$$

VSR$_\omega$ will be computationally compared to PASS for DSATUR$_\omega$ in another section.

Vertex Selection Rule for DSATUR$_{\chi_{G_A}}$

In this subsection, we present a new VSR aiming at reducing the computation time of χ_{G_A}. Reducing the size of the instance on which χ_{G_A} is computed, i.e., reducing the size of auxiliary graph G_A, can lead to a decrease in the computation time of χ_{G_A}. Recalling that building G_A follows this scheme (more details in Section 2.3):

$$G^{\tilde{C}} \to \bar{G}^{\tilde{C}} \to L(\overrightarrow{\bar{G}^{\tilde{C}}}) \to G_A,$$

let us define the possible operations affecting the construction of G_A. Considering a branching node of DSATUR, a vertex v to be colored, a partial coloring $\tilde{C} = \{\tilde{V}_1, \dots, \tilde{V}_{\tilde{k}}\}$ of size $\tilde{k}$ and the current reduced graph $G^{\tilde{C}}$ and according to the construction of graph $G^{\widehat{C}}$, we present the corresponding consequences of these cases in the graph G_A:

- if v is colored with a new color, $|\{\tilde{v}_1, \dots, \tilde{v}_k\} \backslash N_{G^{\tilde{C}}}(v)|$ vertices are removed from G_A.
- if v is colored with an existing color k (see Figure 2.6), there are

$$\begin{aligned}\frac{|V^{\widehat{C}}| * (|V^{\widehat{C}}| - 1)}{2} - |E^{\widehat{C}}| &= \frac{(|V^{\tilde{C}}| - 1) * (|V^{\tilde{C}}| - 2)}{2} - (|E^{\tilde{C}}| - |N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)|) \\ &= \frac{(|V^{\tilde{C}}| - 1) * (|V^{\tilde{C}}| - 2)}{2} - |E^{\tilde{C}}| + |N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)|\end{aligned}$$

  edges in $\bar{G}^{\widehat{C}}$. Thus there is the same number of vertices in $L(\overrightarrow{G}^{\widehat{C}})$, as in G_A.

We have:

- For $G^{\tilde{C}}$, $|V^{\tilde{C}}| = 5$, $|E^{\tilde{C}}| = 5$, $|N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(v_k)| = 1$.
- For $G^{\widehat{C}}$, $|V^{\widehat{C}}| = |V^{\tilde{C}}| - 1 = 5 - 1 = 4$, $|E^{\widehat{C}}| = |E^{\tilde{C}}| - |N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(v_k)| = 5 - 1 = 4$.
- For $\bar{G}^{\widehat{C}}$, $|\bar{E}^{\widehat{C}}| = \frac{(|V^{\tilde{C}}|-1)*(|V^{\tilde{C}}|-2)}{2} - |E^{\tilde{C}}| + |N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)| = \frac{4*3}{2} - 5 + 1 = 2$

Since both $|V^{\tilde{C}}|$ and $|E^{\tilde{C}}|$ are independent from the chosen color for v, the only variable we can affect with the choice of v and its color is $|N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)|$. Decreasing $|N_{G^{\tilde{C}}}(v) \cap N_{G^{\tilde{C}}}(\tilde{v}_k)|$ decreases $|E^{\widehat{C}}|$ and thus decreases $|V_A|$. A smaller value of $|V_A|$ can potentially reduce the computing time of bound χ_{G_A}.

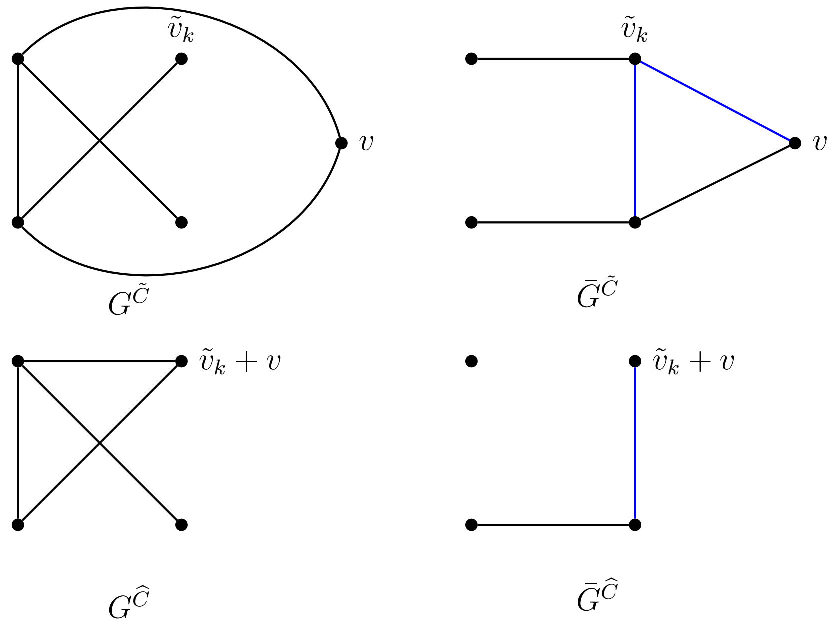

Figure 2.6: Changes between $G^{\widehat{C}}$ and $\bar{G}^{\widehat{C}}$ when v is colored with an existing color k

Considering these changes in $G^{\tilde{C}}$, and thus G_A, the following VSR can be used. Let T be the set of tied candidate vertices, a VSR maximizing the decrease of $|N_{G^{\tilde{C}}}(u) \cap N_{G^{\tilde{C}}}(\tilde{v}_i)|$ would choose a vertex v such that:

$$v = \min_{u \in T} \sum_{i \in \{1,\ldots,\tilde{k}\}} |N_{G^{\tilde{C}}}(u) \cap N_{G^{\tilde{C}}}(\tilde{v}_i)|$$

This VSR enables us to choose a vertex maximizing the size of each potential graph G_A in each created child node of the current node. This VSR, denoted $\text{VSR}_{\chi_{G_A}}$, can also be applied as a whole vertex selection procedure. Let us remark that:

$$DSAT(u, \tilde{C}) = |\{\tilde{v}_1, \ldots, \tilde{v}_k\} \cap N_{G^{\tilde{C}}}(u)|.$$

Choosing the vertex v such that $v = \min\limits_{u \in V \setminus \tilde{V}} \sum\limits_{i \in \{1,\ldots,\tilde{k}\}} |N_{G^{\tilde{C}}}(u) \cap N_{G^{\tilde{C}}}(\tilde{v}_i)|$ accomplishes the opposite of the DSAT rule. The DSAT rule colors in priority vertices that have a lot of colored neighbours, thus iteratively forming cliques. This rule works the opposite way and colors in priority non-adjacent vertices since the vertex with the least colored neighbours is chosen. Instead of iteratively forming cliques, it iteratively forms stable sets.

$\text{VSR}_{\chi_{G_A}}$ will be computationally compared to PASS for $\text{DSATUR}_{\chi_{G_A}}$ in another section.

Computational results

All the algorithms are coded in C/C++, and run on a PC with an Intel(R) Core(TM) i7-4770 CPU at 3.40GHz and 16 GB RAM memory, under Linux Ubuntu 14.04 64-bit.

We implemented an additional improvement designed to enhance the performances of the new DSATUR$_{LB}$ algorithm. The UB is now initialized using the upper bound provided by the standard DSATUR algorithm after a time limit of 3600 seconds. The rationale behind this choice is to improve the quality of the initial solution in order to increase the chances to prune the nodes in the initial phase of the algorithm, i.e., before high quality (optimal) solutions are found. Moreover, this initialization method for UB has been used for ILP based algorithm for the VCP, see Malaguti et al. [37]. This method is also used for DSATUR to allow a fair comparison. The lower bounds are computed at each node of the branching tree to test the impact in reducing the number of explored nodes. Accordingly, in Algorithm 3, the function $\phi_{u,g}(\tilde{C})$ always returns true.

In order to demonstrate the performances of DSATUR$_{LB}$, we propose the following testbed. A first part is composed by the random instances of [51] with $n = 60, 70, 75, 80$. A second part is composed by larger high density random graphs since bounds χ_{G_A} and χ_f^* both show the most potential for very dense instances. To fully test the potential of DSATUR$_{LB}$, we generate 50 additional instances for each graph size $n = 80, 85, 90, 95, 100, 110, 120, 130$ with $d = 0.7, 0.8, 0.9$ and $n = 140$ and $d = 0.9$. The time limit of 1200 seconds is imposed. Each line in the following tables reports average results in the same fashion of Table 2.1. The following additional average information are reported for each version of algorithm DSATUR$_{LB}$:

- UB: the average number of colors (the chromatic number or an upper bound in case time limit is reached).
- *nodes*: the average number of explored nodes.
- *time*: the average computation time.
- *fails*: the number of instances that cannot be solved within the time limit.

First, we computationally choose the best VSR for each version of DSATUR$_{LB}$. Second, we compare three versions of DSATUR$_{LB}$: DSATUR$_{\chi_f^*}$, DSATUR$_{\chi_{G_A}}$ and DSATUR$_{\omega}$ and select the most efficient one in Subsection 2.7.2. Third, we test the impact of the proposed bounding procedure in updating the lower bound at each node of the branching scheme. Then we discuss possible enhancements based on the function ϕ to select a promising subset of nodes in which the lower bound is computed (Subsection 2.7.4).

Vertex Selection Rule for DSATUR$_{LB}$

DSATUR$_\omega$

In this subsection, we computationally compare the impact of VSR$_\omega$ on the performance of DSATUR$_\omega$.

First we present the specificities of DSATUR$_\omega$. The first additional component of DSATUR$_\omega$ concerns the stopping criteria for the computing of ω. The search for $\omega(G)$ is stopped as soon as the value of the size of the current clique C is such that $C \geq UB$. This is possible thanks to the Cliquer algorithm's dedicated feature [46].

For any size and density, DSATUR$_\omega$ with VSR$_\omega$ is slower than DSATUR$_\omega$ with PASS. From any density and $n = 60$, $n = 70$ and $d \leq 0.3$, $n = 75$ and $d \leq 0.3$, $n = 80$ $d = 0.1, 0.2, 0.9$, DSATUR$_\omega$ with PASS and DSATUR$_\omega$ with VSR$_\omega$ have the same number of fails. DSATUR$_\omega$ with PASS has consistently a smaller computation time and number of nodes. This is due to the fact that ω is too weak to cut many nodes of the branching tree, as expected, leading to a larger computation time. VSR$_\omega$ does not reduce the number of nodes as we hoped, instead, the number of nodes is greater for DSATUR$_\omega$ with VSR$_\omega$ for any size and density. For all other entries of the table, DSATUR$_\omega$ with VSR$_\omega$ has at least a greater number of fails than DSATUR$_\omega$ with PASS.

The goal of VSR$_\omega$ is to create children nodes with the largest value of ω. Unfortunately, this does not allow the algorithm to be more efficient, as even the most refined value of ω is too weak to cut many potential subtrees. For all further computations, DSATUR$_\omega$ will be computed with PASS, as it is overall more efficient than VSR$_\omega$.

DSATUR$_{\chi_{G_A}}$

In this subsection, we computationally compare the impact of VSR$_{\chi_{G_A}}$ on the performance of DSATUR$_{\chi_{G_A}}$.

Instance			PASS				VSR$_\omega$			
n	d	#	*UB*	time	nodes	fails	*UB*	time	nodes	fails
60	0.1	50	4.00	0.00	8	0	4.00	0.00	9	0
60	0.2	50	5.42	0.04	866	0	5.42	0.08	1,808	0
60	0.3	50	7.00	0.04	657	0	7.00	0.06	1,071	0
60	0.4	50	8.86	1.57	24,218	0	8.86	2.49	38,714	0
60	0.5	50	10.70	4.55	65,312	0	10.70	6.53	94,643	0
60	0.6	50	12.90	5.27	71,949	0	12.90	6.86	94,862	0
60	0.7	50	15.58	2.21	28,582	0	15.58	2.95	38,822	0
60	0.8	50	19.14	0.53	6,401	0	19.14	0.69	8,492	0
60	0.9	50	25.66	0.05	702	0	25.66	0.08	1,117	0
70	0.1	50	4.00	0.00	3	0	4.00	0.00	3	0
70	0.2	50	6.00	0.04	672	0	6.00	0.08	1,267	0
70	0.3	50	7.84	4.26	54,512	0	7.84	7.89	101,917	0
70	0.4	50	9.68	55.91	643,706	0	9.68	95.62	1,104,013	0
70	0.5	50	11.81	124.80	1,326,686	2	11.81	169.81	1,807,151	3
70	0.6	50	14.08	90.50	858,924	1	14.08	125.45	1,206,216	1
70	0.7	48	17.19	134.68	1,231,382	1	17.17	154.82	1,437,064	2
70	0.8	50	21.38	25.52	212,054	0	21.38	35.54	303,880	0
70	0.9	50	28.52	1.01	6,953	0	28.52	1.63	11,557	0
75	0.1	49	4.00	0.00	3	0	4.00	0.00	3	0
75	0.2	49	6.02	0.07	936	0	6.02	0.15	2,015	0
75	0.3	49	7.98	4.28	46,917	0	7.98	8.54	94,635	0
75	0.4	49	10.02	73.58	714,541	3	10.00	88.50	867,661	4
75	0.5	49	12.02	105.37	883,363	2	11.98	102.65	861,695	4
75	0.6	49	14.83	392.64	3,200,942	14	14.79	322.72	2,586,822	21
75	0.7	49	17.90	310.17	2,149,942	7	17.86	263.21	1,829,003	12
75	0.8	49	22.34	151.94	959,239	2	22.33	172.30	1,104,192	3
75	0.9	49	29.94	27.86	180,079	0	29.92	11.85	73,812	1
80	0.1	49	4.33	0.03	532	0	4.33	0.13	1,911	0
80	0.2	49	6.29	3.01	36,416	0	6.29	8.15	101,004	0
80	0.3	49	8.23	84.33	860,906	1	8.20	89.35	910,424	3
80	0.7	50	18.67	690.64	3,887,580	38	18.57	580.72	3,330,085	43
80	0.8	50	23.46	300.34	1,330,490	13	23.50	278.46	1,226,224	18
80	0.9	50	31.34	24.62	83,232	0	31.34	43.41	153,183	0
85	0.7	50	18.75	551.16	2,289,969	46	18.67	574.45	2,527,693	47
85	0.8	50	24.13	570.99	1,792,286	34	24.08	558.25	1,789,701	38
85	0.9	50	32.75	149.27	358,950	2	32.70	195.46	514,749	4
90	0.7	50	-	-	-	50	-	-	-	50
90	0.8	50	25.00	1046.77	2,690,432	49	-	-	-	50
90	0.9	50	34.00	450.25	632,900	11	34.11	449.08	635,672	22
95	0.7	50	-	-	-	50	-	-	-	50
95	0.8	50	-	-	-	50	-	-	-	50
95	0.9	50	35.69	539.04	587,885	37	35.56	604.37	652,083	41
100	0.7	50	-	-	-	50	-	-	-	50
100	0.8	50	-	-	-	50	-	-	-	50
100	0.9	50	37.50	863.41	824,026	46	37.00	662.59	717,876	49
105	0.7	50	-	-	-	50	-	-	-	50
105	0.8	50	-	-	-	50	-	-	-	50
105	0.9	50	-	-	-	50	-	-	-	50
110	0.7	50	-	-	-	50	-	-	-	50
110	0.8	50	-	-	-	50	-	-	-	50
110	0.9	50	-	-	-	50	-	-	-	50
120	0.7	50	-	-	-	50	-	-	-	50
120	0.8	50	-	-	-	50	-	-	-	50
120	0.9	50	-	-	-	50	-	-	-	50
130	0.7	50	-	-	-	50	-	-	-	50
130	0.8	50	-	-	-	50	-	-	-	50
130	0.9	50	-	-	-	50	-	-	-	50
140	0.9	50	-	-	-	50	-	-	-	50

Table 2.5: DSATUR$_\omega$ with PASS and VSR$_\omega$

In order to enhance the performances of DSATUR$_{\chi_{G_A}}$, the cut separation performed by `CPLEX` at the root node used to compute $\bar{\alpha}(G_A)$ (see Section 2.3.2) is interrupted as soon as $\lfloor \bar{\alpha}(G_A) \rfloor \geq \tilde{n} - UB$ (implemented thanks to the `CPLEX cutcallback`). This improvement is particularly important since it allows to prune nodes as soon as the value of the bound is strong enough, avoiding in this manner unnecessary computational efforts.

Table 2.6 shows that from $n = 60$ to $n = 85$, DSATUR$_{\chi_{G_A}}$ with PASS is more efficient than DSATUR$_{\chi_{G_A}}$ with VSR$_{\chi_{G_A}}$. DSATUR$_{\chi_{G_A}}$ with PASS has either the same or a smaller number of fails than DSATUR$_{\chi_{G_A}}$ with VSR$_{\chi_{G_A}}$ (except for $n = 75$ and $d = 0.2$). From $n = 90$, the opposite occurs as DSATUR$_{\chi_{G_A}}$ with VSR$_{\chi_{G_A}}$ starts to slowly has at least a smaller number of fails than DSATUR$_{\chi_{G_A}}$ with PASS (except for $n = 120$ and $d = 0.8$). When both algorithms have the same number of fails, DSATUR$_{\chi_{G_A}}$ with VSR$_{\chi_{G_A}}$ is still slightly faster than DSATUR$_{\chi_{G_A}}$ with PASS (except for $n = 140$ and $d = 0.9$).

For all further computations, DSATUR$_{\chi_{G_A}}$ will be computed with VSR$_{\chi_{G_A}}$, as it is overall more efficient than PASS.

DSATURχ_f^*

We could not devise a simple VSR dedicated to DSATUR$_{\chi_f^*}$. Instead, in this subsection, we computationally compare the impact of PASS, VSR$_\omega$ and VSR$_{\chi_{G_A}}$ on the performance of DSATUR$_{\chi_f^*}$. The goal of this subsection is to choose the best VSR for DSATUR$_{\chi_f^*}$.

Table 2.7 presents the results of DSATUR$_{\chi_f^*}$ with PASS, VSRω and VSR$_{\chi_{G_A}}$. The number of fails is the same up for all versions of DSATUR$_{\chi_f^*}$ up to $n = 90$. From $n = 95$ to $n = 110$ either DSATUR$_{\chi_f^*}$ with PASS or DSATUR$_{\chi_f^*}$ with VSR$_{\chi_{G_A}}$ has the least number of fails, going back and forth between the two versions. However, from $n = 120$, DSATUR$_{\chi_f^*}$ with VSR$_{\chi_{G_A}}$ always has the least number of fails. In terms of computation time, DSATUR$_{\chi_f^*}$ with PASS and DSATUR$_{\chi_f^*}$ with VSR$_{\chi_{G_A}}$ both have similar average computation time when comparable, i.e., when the two algorithms have the same number of fails. DSATUR$_{\chi_f^*}$ with VSR$_\omega$, while close, is always slower than the two other algorithms. Finally, the number of times DSATUR$_{\chi_f^*}$ with VSR$_{\chi_{G_A}}$ (53) has less fails than DSATUR$_{\chi_f^*}$ (52) with PASS is slightly greater, but as the size of the graphs increase, DSATUR$_{\chi_f^*}$ with VSR$_{\chi_{G_A}}$ seems to consistently have the less fails.

Thus for all following computational results, the VSR for DSATUR$_{\chi_f^*}$ is VSR$_{\chi_{G_A}}$.

Instance			PASS				$\text{VSR}_{\chi_{G_A}}$			
n	d	#	UB	time	nodes	fails	UB	time	nodes	fails
60	0.1	50	4.00	0.74	1	0	4.00	0.81	1	0
60	0.2	50	5.40	152.83	108	2	5.34	216.82	159	6
60	0.3	50	7.00	90.55	51	0	7.00	70.69	39	3
60	0.4	50	8.84	272.88	298	6	8.82	301.84	330	11
60	0.5	50	10.70	167.31	311	0	10.69	167.97	311	2
60	0.6	50	12.90	18.31	56	0	12.90	21.37	65	0
60	0.7	50	15.58	1.04	7	0	15.58	1.13	7	0
60	0.8	50	19.14	0.07	2	0	19.14	0.07	2	0
60	0.9	50	25.66	0.00	1	0	25.66	0.00	1	0
70	0.1	50	4.00	1.21	1	0	4.00	1.28	1	0
70	0.2	50	6.00	406.42	71	4	6.00	412.82	66	17
70	0.3	50	7.27	272.95	68	39	7.20	306.84	76	40
70	0.4	50	9.11	273.99	104	32	9.07	213.92	72	35
70	0.5	50	11.64	301.30	236	25	11.61	313.71	238	27
70	0.6	50	14.08	145.49	235	2	14.06	146.44	217	3
70	0.7	48	17.21	9.36	35	0	17.21	10.26	37	0
70	0.8	50	21.38	0.30	3	0	21.38	0.29	3	0
70	0.9	50	28.52	0.00	1	0	28.52	0.01	1	0
75	0.1	49	4.00	1.93	1	0	4.00	2.10	1	0
75	0.2	49	6.00	405.57	37	9	6.00	356.54	33	18
75	0.3	49	7.83	619.53	106	43	7.75	578.97	107	45
75	0.4	49	10.00	558.87	139	45	10.00	697.49	175	46
75	0.5	49	11.97	390.01	163	18	11.96	408.06	169	21
75	0.6	49	14.81	216.18	247	18	14.81	259.57	293	18
75	0.7	49	17.98	45.25	125	0	17.98	51.15	141	0
75	0.8	49	22.35	1.02	9	0	22.35	1.03	8	0
75	0.9	49	29.94	0.01	1	0	29.94	0.01	1	0
80	0.1	49	4.20	134.98	27	8	4.00	3.30	1	16
80	0.2	49	6.00	388.36	20	23	6.00	419.81	22	28
80	0.3	49	8.00	689.75	73	43	8.00	794.01	90	45
80	0.7	50	18.96	117.03	281	2	18.96	131.39	319	2
80	0.8	50	23.48	1.33	11	0	23.48	1.60	13	0
80	0.9	50	31.34	0.02	1	0	31.34	0.02	2	0
85	0.7	50	19.45	189.64	338	8	19.42	207.80	385	7
85	0.8	50	24.42	7.76	53	0	24.42	7.72	53	0
85	0.9	50	32.80	0.01	1	0	32.80	0.01	1	0
90	0.7	50	20.36	523.71	854	28	20.26	400.69	667	27
90	0.8	50	25.40	32.59	197	0	25.40	30.69	193	0
90	0.9	50	34.14	0.03	2	0	34.14	0.03	3	0
95	0.7	50	21.00	455.60	445	43	21.00	552.58	495	42
95	0.8	50	26.56	104.16	521	0	26.56	84.94	432	0
95	0.9	50	35.70	0.24	17	0	35.70	0.21	17	0
100	0.7	50	22.00	335.14	325	49	22.00	763.28	823	49
100	0.8	50	27.50	256.58	1,088	2	27.49	251.67	1,060	1
100	0.9	50	37.06	0.68	43	0	37.06	0.64	44	0
105	0.7	50	-	-	-	50	-	-	-	50
105	0.8	50	28.41	445.75	1,650	6	28.42	401.30	1,454	5
105	0.9	50	37.96	1.98	99	0	37.96	1.82	101	0
110	0.7	50	-	-	-	50	-	-	-	50
110	0.8	50	29.24	551.69	1,872	25	29.22	558.31	1,903	18
110	0.9	50	39.28	2.72	123	0	39.28	2.69	127	0
120	0.7	50	-	-	-	50	-	-	-	50
120	0.8	50	31.67	862.05	1,994	47	32.00	853.05	2,393	48
120	0.9	50	41.94	10.50	296	0	41.90	9.44	292	11
130	0.7	50	-	-	-	50	-	-	-	50
130	0.8	50	-	-	-	50	44.56	19.90	494	41
130	0.9	50	44.46	28.55	512	0	44.46	28.18	858	0
140	0.9	50	47.00	61.16	785	0	47.00	68.07	893	0

Table 2.6: $\text{DSATUR}_{\chi_{G_A}}$ with PASS and $\text{VSR}_{\chi_{G_A}}$

Instance			PASS				VSR$_\omega$				VSR$_{\chi_{G_A}}$			
n	d	#	UB	time	nodes	fails	UB	time	nodes	fails	UB	time	nodes	fails
60	0.1	50	4.00	0.03	1	0	4.00	0.03	1	0	4.00	0.02	1	0
60	0.2	50	5.42	1.14	27	0	5.42	1.86	46	0	5.42	1.20	30	0
60	0.3	50	7.00	0.09	1	0	7.00	0.09	1	0	7.00	0.08	1	0
60	0.4	50	8.86	2.06	52	0	8.86	2.68	67	0	8.86	2.12	54	0
60	0.5	50	10.70	1.32	35	0	10.70	1.58	41	0	10.70	1.37	35	0
60	0.6	50	12.90	0.51	14	0	12.90	0.59	15	0	12.90	0.54	15	0
60	0.7	50	15.58	0.22	7	0	15.58	0.23	7	0	15.58	0.24	7	0
60	0.8	50	19.14	0.09	3	0	19.14	0.09	3	0	19.14	0.09	3	0
60	0.9	50	25.66	0.01	1	0	25.66	0.01	1	0	25.66	0.01	1	0
70	0.1	50	4.00	0.06	1	0	4.00	0.07	1	0	4.00	0.07	1	0
70	0.2	50	6.00	0.43	3	0	6.00	0.48	4	0	6.00	0.44	3	0
70	0.3	50	7.84	10.66	133	0	7.84	16.79	209	0	7.84	11.21	137	0
70	0.4	50	9.68	23.34	371	0	9.68	34.19	542	0	9.68	24.86	390	0
70	0.5	50	11.82	21.10	368	0	11.82	27.65	481	0	11.82	22.08	383	0
70	0.6	50	14.10	4.21	81	0	14.10	5.46	105	0	14.10	4.47	85	0
70	0.7	48	17.21	0.89	16	0	17.21	0.96	17	0	17.21	0.91	16	0
70	0.8	50	21.38	0.22	6	0	21.38	0.25	6	0	21.38	0.28	7	0
70	0.9	50	28.52	0.02	1	0	28.52	0.03	1	0	28.52	0.03	1	0
75	0.1	49	4.00	0.08	1	0	4.00	0.10	1	0	4.00	0.10	1	0
75	0.2	49	6.02	0.59	5	0	6.02	0.79	6	0	6.02	0.66	5	0
75	0.3	49	7.98	2.51	18	0	7.98	2.90	21	0	7.98	2.68	19	0
75	0.4	49	10.02	22.25	247	1	10.02	35.64	397	1	10.02	22.00	243	1
75	0.5	49	12.06	17.96	253	0	12.06	25.58	361	0	12.06	19.02	264	0
75	0.6	49	14.88	15.79	244	0	14.88	18.81	289	0	14.88	15.99	246	0
75	0.7	49	17.98	2.37	37	0	17.98	2.91	46	0	17.98	2.57	39	0
75	0.8	49	22.35	0.59	11	0	22.35	0.61	11	0	22.35	0.63	12	0
75	0.9	49	29.94	0.03	1	0	29.94	0.04	1	0	29.94	0.03	1	0
80	0.1	49	4.33	3.39	32	0	4.33	10.05	92	0	4.33	3.94	34	0
80	0.2	49	6.29	61.00	538	0	6.27	131.98	1,113	1	6.29	72.00	618	0
80	0.3	49	8.20	75.91	663	3	8.18	84.73	707	4	8.20	79.91	689	3
80	0.7	50	18.96	10.85	150	0	18.96	12.64	174	0	18.96	10.64	146	0
80	0.8	50	23.48	0.77	13	0	23.48	0.89	15	0	23.48	0.92	16	0
80	0.9	50	31.34	0.07	2	0	31.34	0.08	2	0	31.34	0.08	3	0
85	0.7	50	19.46	29.87	362	0	19.46	24.60	288	0	19.46	29.70	355	0
85	0.8	50	24.42	3.48	53	0	24.42	3.69	57	0	24.42	3.38	51	0
85	0.9	50	32.80	0.05	1	0	32.80	0.05	1	0	32.80	0.05	1	0
90	0.7	50	20.31	129.44	1,375	2	20.31	138.32	1,474	2	20.31	136.68	1,466	2
90	0.8	50	25.40	13.11	182	0	25.40	13.24	186	0	25.40	10.98	154	0
90	0.9	50	34.14	0.09	3	0	34.14	0.08	3	0	34.14	0.08	3	0
95	0.7	50	21.15	401.70	3,685	4	21.18	329.51	2,967	6	21.18	296.58	2,683	6
95	0.8	50	26.56	40.01	465	0	26.56	33.29	386	0	26.56	37.87	436	0
95	0.9	50	35.70	0.57	21	0	35.70	0.46	19	0	35.70	0.53	20	0
100	0.7	50	21.96	468.64	3,762	23	22.08	427.12	3,420	25	22.00	442.15	3,546	19
100	0.8	50	27.48	104.83	1,027	0	27.48	95.99	897	0	27.48	93.18	908	0
100	0.9	50	37.06	1.51	52	0	37.06	1.60	56	0	37.06	1.77	59	0
105	0.7	50	22.82	671.73	4,569	33	22.75	762.87	5,108	34	22.71	673.44	4,547	26
105	0.8	50	28.38	194.28	1,649	0	28.41	161.07	1,334	1	28.39	188.55	1,549	1
105	0.9	50	37.96	4.14	123	0	37.96	4.19	126	0	37.96	4.12	124	0
110	0.7	50	23.40	754.59	4,925	45	23.50	788.47	4,783	44	23.67	643.65	3,990	47
110	0.8	50	29.21	412.39	2,805	3	29.22	337.95	2,368	5	29.22	391.65	2,735	4
110	0.9	50	39.28	5.78	152	0	39.28	5.97	161	0	39.28	6.24	161	0
120	0.7	50	-	-	-	50	-	-	-	50	-	-	-	50
120	0.8	50	31.43	591.44	3,159	36	31.50	716.82	3,807	38	31.24	746.37	3,976	33
120	0.9	50	41.94	24.24	410	0	41.89	26.23	430	12	41.89	23.32	403	12
130	0.7	50	-	-	-	50	-	-	-	50	33.00	839.46	3,977	48
130	0.8	50	33.00	663.16	2,926	48	42.45	139.87	997	39	40.43	275.20	1,517	36
130	0.9	50	44.46	57.47	693	0	44.46	57.47	693	0	44.46	57.47	693	0
140	0.9	50	47.00	144.63	1,159	0	47.00	182.93	1,354	0	47.00	149.12	1,221	0

Table 2.7: DSATUR$_{\chi_f^*}$ with PASS, VSR$_\omega$ and VSR$_{\chi_{G_A}}$

DSATUR$_{LB}$

In this section, we compare DSATUR$_{\chi_f^*}$, DSATUR$_{\chi_{G_A}}$ and DSATUR$_{\omega}$.

As expected, Table 2.8 shows that the weakest version of DSATUR$_{LB}$ is DSATUR$_{\omega}$, followed by DSATUR$_{\chi_{G_A}}$. DSATUR$_{\chi_f^*}$ is the strongest version.

Following the conclusions of Section 2.3.3, ω is not able to improve the general efficiency of DSATUR. Despite the very fast computing time of ω, the value of the bound is not able to cut enough nodes. DSATUR$_{\omega}$ is the weakest version of DSATUR$_{LB}$.

DSATUR$_{\chi_f^*}$ is the most efficient version of DSATUR$_{LB}$. It systematically has the least number of fails out of the three versions of DSATUR$_{LB}$. For very dense instances ($d = 0.9$), DSATUR$_{\chi_{G_A}}$ is as efficient as DSATUR$_{\chi_f^*}$, the computing time, the number of nodes and the number of fails are very close between the two algorithms. According to Section 2.3.3, this result is expected since χ_{G_A} and χ_f^* have similar computing time and bound value for this class of instances.

In the next subsections, we will compare DSATUR$_{\chi_f^*}$ to DSATUR as it is the most efficient version of DSATUR$_{LB}$. In the Appendix of this manuscript, we compare DSATUR$_{\chi_{G_A}}$ to DSATUR.

Comparison between DSATUR$_{LB}$ and DSATUR

In this section, we compare the best version of DSATUR$_{LB}$, DSATUR$_{\chi_f^*}$ to DSATUR. Table 2.9 reports the results on the new test bed of instances. The following additional average information are reported for each algorithm:

- *#bounds*: the average number of times χ_f^* is computed.
- *#bounds**: the average number of times χ_f^* is able to prune a node

Table 2.9 clearly shows that DSATUR$_{\chi_f^*}$ drastically reduces the number of explored nodes. For dense instances ($d = 0.7, 0.8, 0.9$), DSATUR$_{\chi_f^*}$ is the fastest algorithm for any size except $n = 60$. For sparse instances ($d = 0.1, 0.2, 0.3$), DSATUR remains the fastest algorithm. From $n = 60$ to $n = 80$, DSATUR and DSATUR$_{\chi_f^*}$ have a similar number of fails. From $n = 85$, the difference between DSATUR and DSATUR$_{\chi_f^*}$ is

Instance			DSATUR$_{\chi_f^*}$				DSATUR$_{\chi_{G_A}}$				DSATUR$_\omega$			
n	d	#	UB	time	nodes	fails	UB	time	nodes	fails	UB	time	nodes	fails
60	0.1	50	4.00	0.02	1	0	4.00	0.81	1	0	4.00	0.00	8	0
60	0.2	50	5.42	1.20	30	0	5.34	216.82	159	6	5.42	0.04	866	0
60	0.3	50	7.00	0.08	1	0	7.00	70.69	39	3	7.00	0.04	657	0
60	0.4	50	8.86	2.12	54	0	8.82	301.84	330	11	8.86	1.57	24,218	0
60	0.5	50	10.70	1.37	35	0	10.69	167.97	311	2	10.70	4.55	65,312	0
60	0.6	50	12.90	0.54	15	0	12.90	21.37	65	0	12.90	5.27	71,949	0
60	0.7	50	15.58	0.24	7	0	15.58	1.13	7	0	15.58	2.21	28,582	0
60	0.8	50	19.14	0.09	3	0	19.14	0.07	2	0	19.14	0.53	6,401	0
60	0.9	50	25.66	0.01	1	0	25.66	0.00	1	0	25.66	0.05	702	0
70	0.1	50	4.00	0.07	1	0	4.00	1.28	1	0	4.00	0.00	3	0
70	0.2	50	6.00	0.44	3	0	6.00	412.82	66	17	6.00	0.04	672	0
70	0.3	50	7.84	11.21	137	0	7.20	306.84	76	40	7.84	4.26	54,512	0
70	0.4	50	9.68	24.86	390	0	9.07	213.92	72	35	9.68	55.91	643,706	0
70	0.5	50	11.82	22.08	383	0	11.61	313.71	238	27	11.81	124.80	1,326,686	2
70	0.6	50	14.10	4.47	85	0	14.06	146.44	217	3	14.08	90.50	858,924	1
70	0.7	48	17.21	0.91	16	0	17.21	10.26	37	0	17.19	134.68	1,231,382	1
70	0.8	50	21.38	0.28	7	0	21.38	0.29	3	0	21.38	25.52	212,054	0
70	0.9	50	28.52	0.03	1	0	28.52	0.01	1	0	28.52	1.01	6,953	0
75	0.1	49	4.00	0.10	1	0	4.00	2.10	1	0	4.00	0.00	3	0
75	0.2	49	6.02	0.66	5	0	6.00	356.54	33	18	6.02	0.07	936	0
75	0.3	49	7.98	2.68	19	0	7.75	578.97	107	45	7.98	4.28	46,917	0
75	0.4	49	10.02	22.00	243	1	10.00	697.49	175	46	10.02	73.58	714,541	3
75	0.5	49	12.06	19.02	264	0	11.96	408.06	169	21	12.02	105.37	883,363	2
75	0.6	49	14.88	15.99	246	0	14.81	259.57	293	18	14.83	392.64	3,200,942	14
75	0.7	49	17.98	2.57	39	0	17.98	51.15	141	0	17.90	310.17	2,149,942	7
75	0.8	49	22.35	0.63	12	0	22.35	1.03	8	0	22.34	151.94	959,239	2
75	0.9	49	29.94	0.03	1	0	29.94	0.01	1	0	29.94	27.86	180,079	0
80	0.1	49	4.33	3.94	34	0	4.00	3.30	1	16	4.33	0.03	532	0
80	0.2	49	6.29	72.00	618	0	6.00	419.81	22	28	6.29	3.01	36,416	0
80	0.3	49	8.20	79.91	689	3	8.00	794.01	90	45	8.23	84.33	860,906	1
80	0.7	50	18.96	10.64	146	0	18.96	131.39	319	2	18.67	690.64	3,887,580	38
80	0.8	50	23.48	0.92	16	0	23.48	1.60	13	0	23.46	300.34	1,330,490	13
80	0.9	50	31.34	0.08	3	0	31.34	0.02	2	0	31.34	24.62	83,232	0
85	0.7	50	19.46	29.70	355	0	19.42	207.80	385	7	18.75	551.16	2,289,969	46
85	0.8	50	24.42	3.38	51	0	24.42	7.72	53	0	24.13	570.99	1,792,286	34
85	0.9	50	32.80	0.05	1	0	32.80	0.01	1	0	32.75	149.27	358,950	2
90	0.7	50	20.31	136.68	1,466	2	20.26	400.69	667	27	-	-	-	50
90	0.8	50	25.40	10.98	154	0	25.40	30.69	193	0	25.00	1046.77	2,690,432	49
90	0.9	50	34.14	0.08	3	0	34.14	0.03	3	0	34.00	450.25	632,900	11
95	0.7	50	21.18	296.58	2,683	6	21.00	552.58	495	42	-	-	-	50
95	0.8	50	26.56	37.87	436	0	26.56	84.94	432	0	-	-	-	50
95	0.9	50	35.70	0.53	20	0	35.70	0.21	17	0	35.69	539.04	587,885	37
100	0.7	50	22.00	442.15	3,546	19	22.00	763.28	823	49	-	-	-	50
100	0.8	50	27.48	93.18	908	0	27.49	251.67	1,060	1	-	-	-	50
100	0.9	50	37.06	1.77	59	0	37.06	0.64	44	0	37.50	863.41	824,026	46
105	0.7	50	22.71	673.44	4,547	26	-	-	-	50	-	-	-	50
105	0.8	50	28.39	188.55	1,549	1	28.42	401.30	1,454	5	-	-	-	50
105	0.9	50	37.96	4.12	124	0	37.96	1.82	101	0	-	-	-	50
110	0.7	50	23.67	643.65	3,990	47	-	-	-	50	-	-	-	50
110	0.8	50	29.22	391.65	2,735	4	29.22	558.31	1,903	18	-	-	-	50
110	0.9	50	39.28	6.24	161	0	39.28	2.69	127	0	-	-	-	50
120	0.7	50	-	-	-	50	-	-	-	50	-	-	-	50
120	0.8	50	31.24	746.37	3,976	33	32.00	853.05	2,393	48	-	-	-	50
120	0.9	50	41.89	23.32	403	12	41.90	9.44	292	11	-	-	-	50
130	0.7	50	33.00	839.46	3,977	48	-	-	-	50	-	-	-	50
130	0.8	50	40.43	275.20	1,517	36	44.56	19.90	494	41	-	-	-	50
130	0.9	50	44.46	57.47	693	0	44.46	28.18	858	0	-	-	-	50
140	0.9	50	47.00	149.12	1,221	0	47.00	68.07	893	0	-	-	-	50

Table 2.8: Comparison between DSATUR$_{\chi_f^*}$, DSATUR$_\omega$ and DSATUR$_{\chi_{G_A}}$

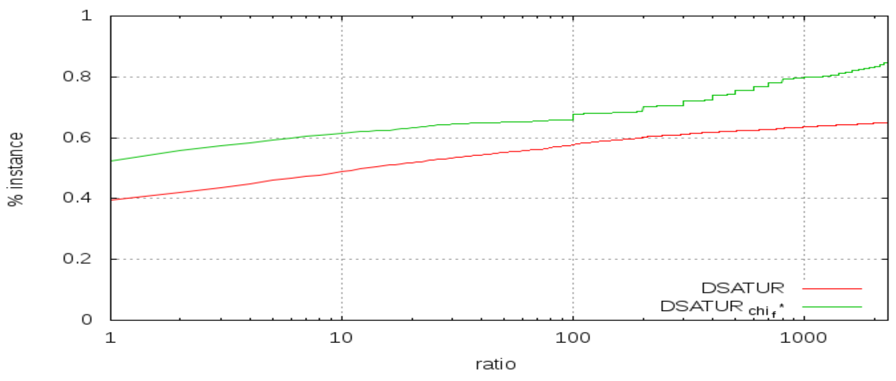

Figure 2.7: Performance profile of DSATUR and DSATUR$_{\chi_f^*}$ on a benchmark of random instances.

becoming larger, DSATUR$_{\chi_f^*}$ always has less fails than DSATUR. From $n = 85$ to $n = 90$, DSATUR totalizes 107 fails against only 2 for DSATUR$_{\chi_f^*}$. This behavior keeps getting worse since starting from $n = 120$, DSATUR cannot solve any instance for any density. Starting from $n = 110$, DSATUR$_{\chi_f^*}$ struggles with instances with density $d = 0.7$, solving almost no instances. For $d = 0.9$, DSATUR$_{\chi_f^*}$ remains very efficient, solving all instances (except for $n = 120$).

Figure 2.7 presents the performance profile of algorithms DSATUR and DSATUR$_{\chi_f^*}$ for the random benchmark of instances. For a complete explanation of performances profiles, see the Appendix. This performance profile shows that DSATUR$_{\chi_f^*}$ is more efficient than DSATUR no matter the time ratio. For every possible value of the ratio, MIP_E is able to be more efficient on a larger percentage of instances than MIP_C. Finally, DSATUR$_{\chi_f^*}$ is able to solve 89% of the benchmark against 66% for DSATUR.

In Table 2.9, we can see that only a fraction of the calculated bounds (*#bounds*) are able to prune the nodes (*#bounds**). This fact suggests that better performance can be achieved using the function $\phi_{u,g}(\tilde{C})$, i.e., trying to limit the number of useless bound computations. In addition, since the computational effort to compute the bounds represents the majority of the time of DSATUR$_{\chi_f^*}$ (on average, 86% of the total time), finding effective strategies to trigger the computation only in promising nodes becomes a crucial issue. In the following section we discuss the proposed strategies for $\phi_{u,g}(\tilde{C})$ and their impact on the performance of DSATUR$_{\chi_f^*}$.

Instance			DSATUR Reimplementation[16]				$\text{DSATUR}_{\chi_f^*}$				
n	d	#	UB	time	nodes	fails	UB	time	nodes	#bounds* \ #bounds	fails
60	0.1	50	4.00	0.00	11	0	4.00	0.02	1	1.00\ 1.00	0
60	0.2	50	5.54	0.00	1,141	0	5.42	1.20	30	14.18\ 29.68	0
60	0.3	50	7.00	0.00	1,054	0	7.00	0.08	1	1.12\ 1.20	0
60	0.4	50	8.86	0.02	41,490	0	8.86	2.12	54	32.02\ 54.36	0
60	0.5	50	10.70	0.10	140,806	0	10.70	1.37	35	22.30\ 35.48	0
60	0.6	50	12.90	0.16	200,882	0	12.90	0.54	15	9.42\ 14.82	0
60	0.7	50	15.58	0.09	120,228	0	15.58	0.24	7	4.22\ 6.50	0
60	0.8	50	19.14	0.02	35,216	0	19.14	0.09	3	2.08\ 2.88	0
60	0.9	50	25.66	0.00	2,957	0	25.66	0.01	1	1.00\ 1.00	0
70	0.1	50	4.00	0.00	4	0	4.00	0.07	1	1.00\ 1.00	0
70	0.2	50	6.00	0.00	970	0	6.00	0.44	3	2.06\ 3.30	0
70	0.3	50	7.84	0.06	83,812	0	7.84	11.21	137	79.04\ 137.22	0
70	0.4	50	9.68	0.94	1,163,336	0	9.68	24.86	390	239.34\ 390.34	0
70	0.5	50	11.82	3.42	3,959,720	0	11.82	22.08	383	244.98\ 383.40	0
70	0.6	50	14.10	3.20	3,518,339	0	14.10	4.47	85	55.52\ 84.82	0
70	0.7	48	17.21	5.64	6,144,280	0	17.21	0.91	16	10.63\ 16.21	0
70	0.8	50	21.38	1.18	1,306,601	0	21.38	0.28	7	4.44\ 6.76	0
70	0.9	50	28.52	0.04	43,187	0	28.52	0.03	1	1.08\ 1.16	0
75	0.1	49	4.00	0.00	4	0	4.00	0.10	1	1.00\ 1.00	0
75	0.2	49	6.02	0.00	1,393	0	6.02	0.66	5	2.92\ 4.84	0
75	0.3	49	7.98	0.05	73,354	0	7.98	2.68	19	11.98\ 19.43	0
75	0.4	49	10.04	3.91	4,636,774	0	10.02	22.00	243	155.73\ 242.60	1
75	0.5	49	12.06	13.69	14,912,458	0	12.06	19.02	264	180.84\ 264.29	0
75	0.6	49	14.88	26.14	27,291,675	0	14.88	15.99	246	164.18\ 245.57	0
75	0.7	49	17.98	28.78	29,945,679	0	17.98	2.57	39	26.29\ 39.47	0
75	0.8	49	22.35	7.82	8,072,599	0	22.35	0.63	12	7.59\ 11.73	0
75	0.9	49	29.94	1.82	1,931,718	0	29.94	0.03	1	1.02\ 1.06	0
80	0.1	49	4.33	0.00	641	0	4.33	3.94	34	14.94\ 33.73	0
80	0.2	49	6.29	0.04	50,285	0	6.29	72.00	618	303.33\ 617.84	0
80	0.3	49	8.24	2.24	2,767,076	0	8.20	79.91	689	389.48\ 689.02	3
80	0.7	50	18.96	213.54	211,412,127	2	18.96	10.64	146	95.86\ 146.36	0
80	0.8	50	23.48	72.93	71,749,271	0	23.48	0.92	16	10.06\ 16.08	0
80	0.9	50	31.34	0.70	655,041	0	31.34	0.08	3	1.88\ 2.58	0
85	0.7	50	19.31	395.63	373,397,208	15	19.46	29.70	355	235.54\ 355.14	0
85	0.8	50	24.40	134.99	123,858,470	10	24.42	3.38	51	28.72\ 50.96	0
85	0.9	50	32.80	22.91	20,330,498	0	32.80	0.05	1	1.00\ 1.00	0
90	0.7	50	20.25	499.72	453,007,895	46	20.31	136.68	1,466	972.35\ 1465.54	2
90	0.8	50	25.13	454.77	394,727,070	35	25.40	10.98	154	81.40\ 153.90	0
90	0.9	50	34.14	22.46	18,789,790	1	34.14	0.08	3	1.34\ 2.64	0
95	0.7	50	-	-	-	50	21.18	296.58	2,683	1786.48\ 2682.20	6
95	0.8	50	26.50	860.13	751,042,053	48	26.56	37.87	436	253.20\ 435.68	0
95	0.9	50	35.70	90.99	71,461,097	10	35.70	0.53	20	6.46\ 19.96	0
100	0.7	50	-	-	-	50	22.00	442.15	3,546	2346.39\ 3544.42	19
100	0.8	50	-	-	-	50	27.48	93.18	908	544.14\ 906.60	0
100	0.9	50	37.00	274.54	202,716,576	22	37.06	1.77	59	20.36\ 58.70	0
105	0.7	50	-	-	-	50	22.71	673.44	4,547	3023.54\ 4545.83	26
105	0.8	50	-	-	-	50	28.39	188.55	1,549	957.45\ 1547.00	1
105	0.9	50	38.11	450.11	321,842,577	41	37.96	4.12	124	37.58\ 123.14	0
110	0.7	50	-	-	-	50	23.67	643.65	3,990	2528.33\ 3988.67	47
110	0.8	50	-	-	-	50	29.22	391.65	2,735	1704.20\ 2733.35	4
110	0.9	50	39.33	674.60	472,118,968	47	39.28	6.24	161	50.82\ 159.88	0
120	0.7	50	-	-	-	50	-	-	-	-	50
120	0.8	50	-	-	-	50	31.24	746.37	3,976	2484.59\ 3973.71	33
120	0.9	50	-	-	-	50	41.89	23.32	403	151.08\ 400.79	12
130	0.7	50	-	-	-	50	33.00	839.46	3,977	2455.50\ 3973.00	48
130	0.8	50	-	-	-	50	40.43	275.20	1,517	792.21\ 1514.07	36
130	0.9	50	-	-	-	50	44.46	57.47	693	242.96\ 688.44	0
140	0.9	50	-	-	-	50	47.00	149.12	1,221	501.14\ 1215.78	0

Table 2.9: DSATUR and $\text{DSATUR}_{\chi_f^*}$ for random instances from [51]

Computing lower bounds only at specific nodes of the branching scheme

Given two input parameter u and g, the function $\phi_{u,g}$ is defined as follow:

$$\phi_{u,g}(\tilde{C}) = (u^* \geq u \text{ AND } \tilde{g} \geq g)$$

where u^* corresponds to the number of times the incumbent solution UB has been updated and, $\tilde{g} = UB - \tilde{k}$ (a measure of the difference between the node lower and upper bound). If $u = 0$ and $g = 0$, the corresponding function $\phi_{0,0}(\tilde{C})$ always returns true, and it corresponds to the case in which the bound is computed at each node of the tree.

The results of DSATUR$_{\chi_f^*}$ with different functions $\phi_{u,g}$ are reported in Table 2.10. We test different values of the parameters u and g. For parameter u we test $1, 2, 3$ while for the parameter g we test $2, 3, 4$.

The regular version of DSATUR$_{\chi_f^*}$ is the most efficient according to Table 2.10. This means that the key to improving DSATUR$_{\chi_f^*}$ has no link to parameter g of u. Changing the value of g does not improve the #bounds* $\setminus$ #bounds ratio of DSATUR$_{\chi_f^*}$. Each computed bound in DSATUR$_{\chi_f^*}$ matters for the efficiency of the algorithm. Changing the value of u does not improve #bounds* $\setminus$ #bounds ratio of DSATUR$_{\chi_f^*}$ either. Both the quality of the initial upper bound and the finding speed of the upper bounds from the leaves of the branching tree are good enough for DSATUR$_{\chi_f^*}$. Giving DSATUR$_{\chi_f^*}$ the possibility to compute a better upper bound does not improve its efficiency. The best version of DSATUR$_{\chi_f^*}$ is the regular version using VSR$_{\chi_{G_A}}$ with $\phi_{u,g} = true$.

Conclusion

In this chapter, we have devised an improved Branch-and-Bound algorithm based on DSATUR to solve the VCP. We introduced the use of lower bounds in DSATUR. To achieve that, we define a graph transformation that allows us to compute a lower bound on the current subproblem at each node of the branching tree. We compared our algorithm to the regular version of DSATUR and our new algorithm is more efficient than DSATUR on a benchmark of random instances.

This new version of DSATUR, DSATUR$_{LB}$, still has a lot of drawbacks. The reduced graph and the associated lower bound computed at each node of the branching tree

Instance			$\phi_{0,0}$		$\phi_{0,2}$		$\phi_{0,3}$		$\phi_{0,4}$		$\phi_{1,1}$		$\phi_{2,1}$		$\phi_{3,1}$	
n	d	#	time	fails	time	fails	time	fails	time	fails	time	fails	time	fails	time	fails
60	0.1	50	0.02	0	0.02	0	0.02	0	0.02	0	0.02	0	0.02	0	0.02	0
60	0.2	50	1.20	0	0.16	0	0.05	0	0.05	0	0.06	0	0.06	0	0.06	0
60	0.3	50	0.08	0	0.07	0	0.06	0	0.06	0	0.07	0	0.07	0	0.07	0
60	0.4	50	2.12	0	0.70	0	0.15	0	0.08	0	0.08	0	0.08	0	0.08	0
60	0.5	50	1.37	0	0.73	0	0.27	0	0.14	0	0.14	0	0.14	0	0.14	0
60	0.6	50	0.54	0	0.39	0	0.22	0	0.16	0	0.15	0	0.15	0	0.15	0
60	0.7	50	0.24	0	0.17	0	0.11	0	0.08	0	0.07	0	0.07	0	0.07	0
60	0.8	50	0.09	0	0.07	0	0.04	0	0.03	0	0.04	0	0.04	0	0.04	0
60	0.9	50	0.01	0	0.01	0	0.00	0	0.00	0	0.01	0	0.01	0	0.01	0
70	0.1	50	0.07	0	0.06	0	0.05	0	0.05	0	0.06	0	0.06	0	0.06	0
70	0.2	50	0.44	0	0.19	0	0.13	0	0.13	0	0.14	0	0.14	0	0.14	0
70	0.3	50	11.21	0	2.57	0	0.35	0	0.17	0	0.18	0	0.18	0	0.18	0
70	0.4	50	24.86	0	7.45	0	1.66	0	0.92	0	0.91	0	0.91	0	0.97	0
70	0.5	50	22.08	0	10.41	0	4.50	0	3.40	0	3.16	0	3.16	0	3.30	0
70	0.6	50	4.47	0	3.15	0	2.28	0	2.30	0	2.26	0	2.27	0	2.39	0
70	0.7	48	0.91	0	0.82	0	1.01	0	1.87	0	4.23	0	4.24	0	4.45	0
70	0.8	50	0.28	0	0.24	0	0.19	0	0.30	0	0.59	0	0.59	0	0.62	0
70	0.9	50	0.03	0	0.02	0	0.02	0	0.02	0	0.02	0	0.02	0	0.02	0
75	0.1	49	0.10	0	0.08	0	0.07	0	0.07	0	0.08	0	0.08	0	0.08	0
75	0.2	49	0.66	0	0.24	0	0.17	0	0.17	0	0.18	0	0.18	0	0.19	0
75	0.3	49	2.68	0	1.03	0	0.30	0	0.20	0	0.21	0	0.21	0	0.21	0
75	0.4	49	22.00	1	15.41	0	5.14	0	4.05	0	3.93	0	3.93	0	4.12	0
75	0.5	49	19.02	0	13.55	0	11.48	0	12.45	0	10.86	0	10.83	0	11.55	0
75	0.6	49	15.99	0	12.84	0	14.05	0	17.99	0	21.90	0	21.84	0	22.76	0
75	0.7	49	2.57	0	2.84	0	6.33	0	12.24	0	22.52	0	22.46	0	24.70	0
75	0.8	49	0.63	0	0.55	0	0.58	0	0.96	0	3.26	0	3.27	0	3.49	0
75	0.9	49	0.03	0	0.02	0	0.02	0	0.02	0	0.03	0	0.03	0	0.03	0
80	0.1	49	3.94	0	0.17	0	0.11	0	0.11	0	0.12	0	0.12	0	0.12	0
80	0.2	49	72.00	0	2.64	0	0.25	0	0.25	0	0.27	0	0.27	0	0.28	0
80	0.3	49	79.91	3	25.58	0	3.07	0	2.07	0	2.08	0	2.07	0	2.20	0
80	0.7	50	10.64	0	13.63	0	39.11	0	81.41	1	168.34	2	168.44	2	158.09	3
80	0.8	50	0.92	0	0.96	0	2.44	0	12.54	0	38.32	0	38.23	0	41.39	0
80	0.9	50	0.08	0	0.07	0	0.06	0	0.08	0	0.14	0	0.14	0	0.16	0
85	0.7	50	29.70	0	59.06	0	105.05	2	132.33	6	271.02	12	271.00	12	287.01	12
85	0.8	50	3.38	0	3.94	0	18.62	0	43.21	1	103.25	6	77.61	7	83.06	7
85	0.9	50	0.05	0	0.04	0	0.03	0	0.04	0	0.04	0	0.04	0	0.04	0
90	0.7	50	136.68	2	199.05	4	290.56	18	337.39	26	338.94	37	265.54	40	282.08	40
90	0.8	50	10.98	0	13.16	0	58.82	1	146.41	4	211.77	23	210.42	26	152.00	28
90	0.9	50	0.08	0	0.07	0	0.06	0	0.06	0	1.52	0	1.53	0	1.68	0
95	0.7	50	296.58	6	323.59	17	319.11	34	412.84	41	478.47	40	276.05	47	291.10	47
95	0.8	50	37.87	0	50.05	0	209.87	6	241.01	19	463.53	36	419.62	41	347.08	42
95	0.9	50	0.53	0	0.47	0	0.67	0	3.03	0	67.58	5	67.56	5	48.52	6
100	0.7	50	442.15	19	375.37	33	402.47	43	449.43	48	585.20	44	-	50	-	50
100	0.8	50	93.18	0	198.55	0	190.40	18	333.18	31	418.59	42	-	50	-	50
100	0.9	50	1.77	0	1.61	0	2.64	0	20.59	0	105.56	13	120.95	15	130.10	16
105	0.7	50	673.44	26	511.73	45	371.80	48	1006.36	49	740.67	48	-	50	-	50
105	0.8	50	188.55	1	229.82	8	269.33	27	383.49	40	321.58	40	579.61	48	-	50
105	0.9	50	4.12	0	3.73	0	4.38	0	22.52	0	263.94	23	278.72	29	175.18	32
110	0.7	50	643.65	47	1007.16	49	-	50	-	50	431.82	49	-	50	-	50
110	0.8	50	391.65	4	411.73	19	329.94	36	553.16	45	424.16	42	-	50	-	50
110	0.9	50	6.24	0	6.23	0	12.80	0	56.29	1	358.14	34	287.34	38	126.49	40
120	0.7	50	-	50	-	50	-	50	-	50	-	50	-	50	-	50
120	0.8	50	746.37	33	683.31	43	287.71	49	-	50	368.07	46	680.19	48	-	50
120	0.9	50	23.32	12	24.34	0	49.73	3	102.15	9	262.07	24	142.93	40	86.28	44
130	0.7	50	839.46	48	-	50	-	50	-	50	-	50	-	50	-	50
130	0.8	50	275.20	36	775.89	45	-	50	-	50	-	50	-	50	-	50
130	0.9	50	57.47	0	69.22	0	54.38	3	188.43	7	351.63	29	287.70	46	509.44	48
140	0.9	50	149.12	0	143.88	1	221.25	3	240.43	19	227.81	32	298.19	44	267.83	46

Table 2.10: DSATUR$_{\chi_f^*}$ and DSATUR$_{\chi_f^*}$ with different functions $\phi_{u,g}$

are the key of the algorithm. An interesting stream of research could be to study more thoroughly the graph changes occurring in the branching tree of DSATUR$_{LB}$. The idea is to take advantage of those changes in order to reduce the number of times a given reduced graph and its associated lower bound has to be computed. A tailored VSR for DSATUR$_{\chi_f^*}$ could also potentially improve the its efficiency. A dedicated study of the literature on graph operations leading to an increase or a decrease of χ, and thus χ_f^*, could help us create such a VSR.

This work has been published in [17, 15, 14, 13].

In the next chapter, we study a variant of the VCP, the MSCP. We now use ILP models to solve this problem.

Chapter 3

ILP models for the Minimum Sum Coloring Problem

Introduction

The MSCP has been introduced in Kubicka and Schwenk [31] where it is shown that problem is NP-hard for arbitrary graphs. The MSCP corresponds to the scheduling problem of incompatible unitary length jobs on identical parallel-machines, where each job has a due date equal to zero, no preemption is allowed, and the objective is to determine a schedule so that the total tardiness is minimized. Two jobs are incompatible if they cannot be executed at the same time. A related scheduling problem has been addressed in [30].

Several articles propose lower bounds for both $\Sigma(G)$ and $s(G)$. Alavi et al. [55] proved that $\lceil\sqrt{8m}\rceil \leq \Sigma(G) \leq \lfloor\frac{3}{2}(m+1)\rfloor$ holds for any graph G. Sen et al. [53] study a generalization of the MSCP where each color has a given cost. Mitchem and Morriss [43] prove that $s(G) \leq \Delta(G)+1$ holds where $\Delta(G)$ is the maximum degree of the vertices of G. Salavatipour [49] study the MSCP and its variants, and prove complexity results for specific classes of graphs. Lecat et al. [32] propose a new set of upper bounds on $s(G)$ and compare their results computationally to other bounds.

Many heuristic algorithms have also been developed over the years to find good upper bounds for the MSCP. Li et al. [33] develop greedy algorithms adapting the VCP greedy algorithms DSATUR and RLF. Local search algorithms have been developed as well in [24]. Wu and Kao [57, 58] devise a sophisticated greedy algorithm combined with tabu search especially effective for large graphs. Different genetic algorithms and tabu search algorithms have been developed in [28, 27].

The current most successful algorithm for lower and upper bounds on the MSCP is called HESA and it has been proposed by Jin and Hao [27]; it is an algorithm based on stochastic hybrid evolutionary search. We address the interested reader to Jin et al. [26] for more information on these algorithms and a complete survey on the topic.

To the best of our knowledge, very little attention has been given to effective ILP formulations for the MSCP. The only ILP formulation has been proposed in [53] but no computational results are reported. For this reason in this manuscript we study and test three Integer Linear Programming (ILP) formulations for the MSCP. The first and the second ILP formulations have a polynomial number of variables and constraints; while the third formulation has an exponential number of variables and it is tackled via column generation.

The rest of this chapter is organized as follows. In the first section, we present some structural properties of any optimal solution of the MSCP. In the next section, we present three ILP formulations designed to solve the MSCP: two compact formulations and an extended formulation. The next few sections are presenting potential improvements of the extended formulation. Finally, the last section presents the computational results of our ILP models.

Structural properties

We now discuss some structural properties of MSCP optimal solutions. Some of these properties will help us design the formulations and the algorithms modeling and solving the MSCP.

Proposition 5 *Let $C = \{V_1, \dots, V_j, \dots, V_k\}$ be a coloring of G with $V_j = \{\emptyset\}$. Coloring C cannot be an optimal solution to the MSCP.*

Proof. Consider $C' = \{V_1, \dots, V_{j-1}, V_{j+1}, \dots, V_k\}$. Clearly, $\psi(G, C) = \sum_{i=1}^{k} i|V_i| > \psi(G, C') = \sum_{i=1}^{j-1} i|V_i| + \sum_{i=j+1}^{k} (i-1)|V_i|$. $\square$

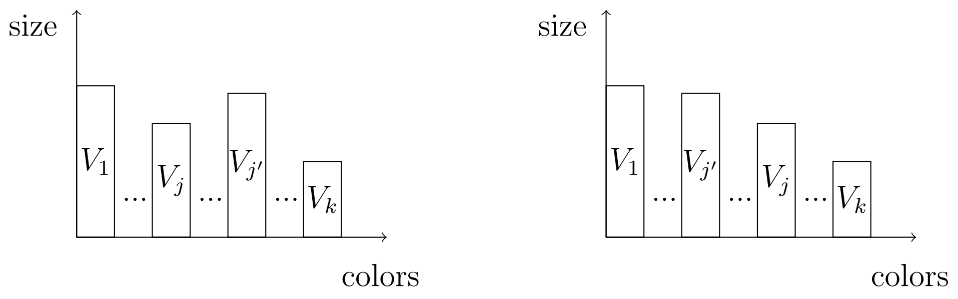

Figure 3.1: Example of Property 2

This property implies that the MSCP optimal coloring is a partition of V into $s(G)$ non-empty stable sets $C = \{V_1, ..., V_{s(G)}\}$.

Proposition 6 *Let $C = \{V_1, \dots, V_k\}$ be a coloring of G where $|V_j| < |V_{j'}|$, $j' > j$. Coloring C cannot be an optimal solution to the MSCP.*

Proof. Consider $C' = \{V_1', ..., V_k'\}$ where $V_i' = V_i$ for all $i \in \{1, ..., k\} \setminus \{j, j'\}$, $V_j' = V_{j'}$ and $V_{j'}' = V_j$. Clearly, $\psi(G, C) > \psi(G, C')$. $\square$

This structural property implies that, in an optimal solution of the MSCP, the cardinalities of the stable sets representing each color are non-increasing. An example is shown in Figure 3.1.

However this does not imply that in an optimal solution of the MSCP, the vertices in the first color class are defined as the maximum cardinality stable set, and that, iteratively, each vertices in each color V_j are defined as the maximum cardinality stable set in $G \backslash \{\cup_{i=1}^{j-1} V_i\}$. A counter-example is given in Figure 3.2. Two different colorings C^l and C^r are indicated by pairs of integer numbers close to each vertex, separated by a "-". The coloring C^l, encoded by the first number of each pair, defines a partition into 3 stables sets, while the coloring C^r, encoded by the second number of each pair, is a partition into 2 stable sets. Coloring C^r includes the maximum stable set V_1 of G, the maximum stable set V_2 of $G \setminus V_1$, and the maximum stable set V_3 of $G \setminus \{V_1 \cup V_2\}$, and has a total cost $\psi(G, C^r) = 16$. The optimal coloring C^r, not including the maximum stable set of G, has a total cost $\psi(G, C^r) = 15$.

Proposition 7 *Let $C = \{V_1, ..., V_k\}$ be an optimal coloring of the MSCP of G. For all $j \in \{1, ..., k\}$, V_j is a maximal stable set in $G \setminus (\cup_{i=1}^{j-1} V_i)$.*

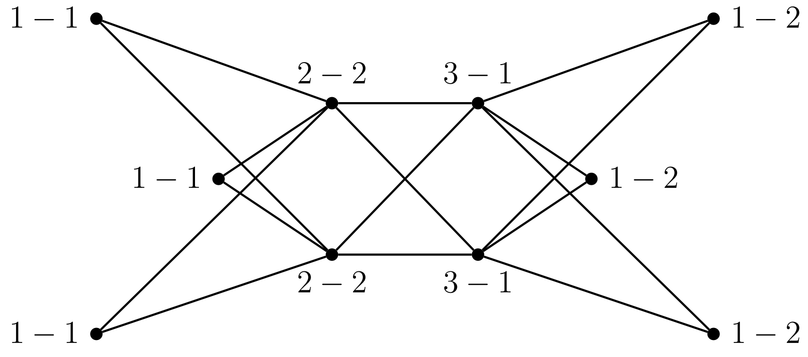

Figure 3.2: Example: two colorings of the same graph, where the first one (C^l) has cost 16, and the second one (C^r) has cost 15.

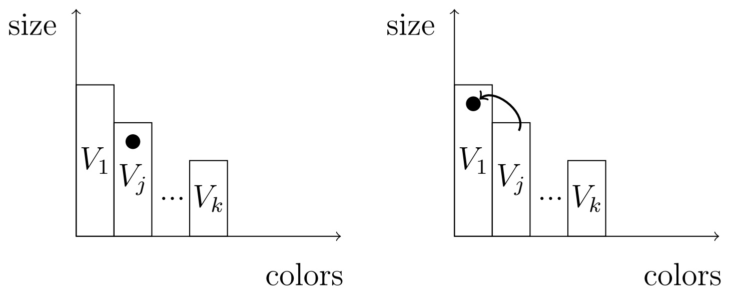

Figure 3.3: Example of Property 3

Proof. If there exists $j \in \{1, ..., k\}$ such that the stable set V_j is not maximal in $G \setminus (\cup_{i=1}^{j-1} V_i)$ then there exists a vertex $v \in V_{j'}$ where $j' > j$ such that the coloring $C' = \{V'_1, ..., V'_k\}$, where $V'_i = V_i$ for all $i \in \{1, ..., k\} \setminus \{j, j'\}$, $V'_j = V_j \cup \{v\}$ and $V'_{j'} = V_{j'} \setminus \{v\}$, is feasible. We have $\psi(G, C) = \psi(G, C') + (j' - j)$. $\square$

For any optimal coloring $C^\star = \{V_1^\star, ..., V_{s(G)}^\star\}$:

- $C^\star$ does not contain an empty color (or stable set).
- $|V_i^\star| \geq |V_j^\star|$ for all $i, j = 1, \dots, s(G), i > j$.
- $V_j^\star$ is a maximal stable in $G \setminus (\cup_{i=1}^{j-1} V_i)$ for all $j = 2, \dots, s(G)$.

In the next section, we present ILP formulations for the MSCP.

ILP Formulations

Let us denote by k an upper bound on the strength of the graph $s(G)$, the first ILP model is a compact formulation with $k \cdot n$ variables and $m + n$ constraints.

Compact Formulation 1

The compact formulation, called MIP_C in the following, uses binary variables x_v^i.

For all $i \in \{1, ..., k\}$ and $v \in V$,

$$x_v^i = \begin{cases} 1 \text{ if vertex } v \text{ has been assigned color } i, \\ 0 \text{ otherwise.} \end{cases}$$

The MSCP can be modeled as follows:

$$Z(MIP_C) \; = \min \sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i \tag{3.1}$$

$$x_u^i + x_v^i \leq 1, \qquad (u,v) \in E, i \in \{1, ..., k\}, \tag{3.2}$$

$$\sum_{i \in \{1,...,k\}} x_v^i = 1, \qquad v \in V, \tag{3.3}$$

$$x_v^i \in \{0,1\}, \qquad v \in V, i \in \{1, ...k\}. \tag{3.4}$$

The objective function (3.1) corresponds to the sum of the costs i of each color times the number of vertices receiving the color. Constraints (3.2) ensure that for each edge of G and for each color $i \in \{1, ..., k\}$, at most one of the endpoints of the edge receives the color i. Constraints (3.3) impose each vertex is colored with exactly one color.

These partitioning constraints ("=") can be replaced by covering constraints ("$\geq$") since the sense of the objective function imposes that each vertex is colored by exactly one color (i.e., if a vertex takes more than one color the objective function increases). Finally, constraints (3.4) are the integrality constraints for variables x.

This formulation is an adaptation of the VCP formulation described in Section 1.3 defined by Sen et al. [53]. To the best of our knowledge, the computational performance of MIP_C has not been tested yet. It is worth mentioning that the cost of each color is different. This fact makes MIP_C less sensitive to symmetry problems which are one of the main drawbacks of this formulation when used to solve the classical Vertex

Coloring Problem (VCP). In Section 3.4, we computationally prove that MIP_C is capable of solving to optimality many MSCP instances of the literature.

Replacing constraints (3.4) with

$$x_v^i \geq 0, \qquad v \in V, i \in \{1, ...k\},$$

we obtain the Linear Programming relaxation of MIP_C, denoted as LP_C.

Upper bound on $z(LP_C)$

An upper bound on the value of the lower bound $z(LP_C)$ can be computed, showing that this lower bound is potentially very weak.

Proposition 8 *The maximum value of $z(LP_C)$ is $n + \frac{n}{2}$.*

Proof. Let x^* denote a vector of variables x_v^i with:

$$\begin{aligned} x_v^{i*} &= 0.5, & \forall v \in V, i \in \{1, 2\}, \\ x_v^{i*} &= 0, & \forall v \in V, i \geq 3. \end{aligned}$$

For each pair of adjacent vertices (u, v), we have $x_v^{i*} + x_w^{i*} = 1 \leq 1$ for $1 \leq i \leq 2$, hence no constraint (3.2) is violated. We also have $\sum\limits_{1 \leq i \leq 2} x_v^i = 1$, for each $v \in V$ hence no constraint (3.3) is violated. For color 1, the objective function is equal to $\frac{n}{2} * 1$ and for color 2, the objective function is equal to $\frac{n}{2} * 2 = n$, hence we have $n + \frac{n}{2}$ since only those two colors are used.

$\square$

Compact Formulation 2

This compact formulation, called MIP_C^2 in the following, uses binary variables x_{uv}^i.

For all $i \in \{1, ..., k\}$ and $u \in V$, $v \in \bar{N}(u)$,

$$x_{uv}^i = \begin{cases} 1 \text{ if } u \text{ is represented by } v \text{ and they have color } i, \\ 0 \text{ otherwise,} \end{cases}$$

and for each $u \in V$ and $i \in \{1, ..., k\}$,

$$x^i_{uu} = \begin{cases} 1 \text{ if } u \text{ represents color } i, \\ 0 \text{ otherwise.} \end{cases}$$

The MSCP can be formulated as follows:

$$Z(MIP^2_C) = \min \sum_{i \in \{1,...,k\}} \sum_{u \in V} \sum_{v \in \bar{N}(u)} i \cdot x^i_{uv} \tag{3.5}$$

$$\sum_{i \in \{1,...,k\}} \left(\sum_{v \in \bar{N}(u)} x^i_{vu} + x^i_{uu} \right) \geq 1, \qquad u \in V, \tag{3.6}$$

$$x^i_{uv} + x^i_{uw} \leq x^i_{uu}, \qquad u \in V, (v,w) \notin E, i \in \{1, ..., k\}, \tag{3.7}$$

$$\sum_{i \in \{1,...,k\}} x^i_{uv} \leq 1, \qquad u \in V, v \in \bar{N}(u), \tag{3.8}$$

$$\sum_{u \in V} x^i_{uu} \leq 1, \qquad i \in \{1, ..., k\}, \tag{3.9}$$

$$x^i_{uv} \in \{0,1\}, \qquad i \in \{1, ..., k\}, u \in V, v \in \bar{N}(u), \tag{3.10}$$

$$x^i_{uu} \in \{0,1\}, \qquad i \in \{1, ..., k\}, u \in V. \tag{3.11}$$

The objective function (3.5) corresponds to the sum of the cost of each vertex of the graph. Constraints (3.6) are covering constraints ensuring that each vertex is assigned one color. Constraints (3.7) are the edge constraints which force two adjacent vertices to have different colors, if this color is being used. Constraints (3.8) indicate that each vertex can only be represented by one vertex at most. Constraints (3.9) limits the number of representative vertices each color can have to at most one. This formulation has $O(n^3)$ variables and $O(n^4)$ constraints. Constraints (3.10) and (3.11) are the integrality constraints for respectively variables x^i_{uv} and x^i_{uu}. Constraints (3.7) can be written as:

$$\sum_{i \in \{1,...,k\}} \left(x^i_{uv} + x^i_{uw} \right) \leq \sum_{i \in \{1,...,k\}} x^i_{uu}, \qquad u \in V, (v,w) \notin E,$$

which allows the model to reduce the total number of constraints.

This formulation is an adaptation of the VCP formulation described in Section 1.5. For the VCP, this formulation is competitive since it allows the solution to avoid some symmetry problems. Even though symmetry problems are not as important for the MSCP, an arbitrary ordering of the vertices $V = \{1, ..., n\}$ can be set so that if $u \prec v$, then v cannot represent u to avoid symmetry. Thus, variables x^i_{uv} can be defined for all $i \in \{1, ..., k\}$, $u \in V$, $v \in \bar{N}(u)$ and $u \prec v$.

By replacing constraints (3.10) and (3.10) with the following ones:

$$\begin{aligned} x^i_{uv} &\geq 0, && i \in \{1, ..., k\}, u \in V, v \in \bar{N}(u), \\ x^i_{uu} &\geq 0, && i \in \{1, ..., k\}, u \in V, \end{aligned}$$

we obtain the Linear Programming relaxation of MIP^2_C, denoted as LP^2_C.

Extended Formulation

We now introduce the new extended formulation (called MIP_E) for the MSCP with an exponential number of variables and $k + n$ constraints. Let us denote the collection of all stable sets of G as:

$$\mathscr{S} = \{S \subseteq V : (u, v) \notin E \quad \forall u, v \in S\}.$$

This model uses the following binary variables. For each stable set $S \in \mathscr{S}$ and each color $i \in \{1, ..., k\}$,

$$\xi^i_S = \begin{cases} 1 \text{ if stable set } S = V_i, \\ 0 \text{ otherwise.} \end{cases}$$

The new extended formulation reads as follows:

$$Z(MIP_E) = \min \sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}} c^i_S \xi^i_S \tag{3.12}$$

$$\sum_{S \in \mathscr{S}} \xi^i_S \leq 1, \qquad i \in \{1, ..., k\}, \tag{3.13}$$

$$\sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S} : v \in S} \xi^i_S \geq 1, \qquad v \in V, \tag{3.14}$$

$$\xi^i_S \in \{0, 1\}, \qquad i \in \{1, ..., k\}, S \in \mathscr{S}, \tag{3.15}$$

where $c^i_S = i \cdot |S|$. In the objective function (3.12), the cost c^i_S is paid if the stable set S receives color i. Constraints (3.13) impose each color i is assigned to at most one stable set S. Constraints (3.14) impose each vertex v is contained in at least one stable set S that receives a color (covering version). Finally, constraints (3.15) are the integrality constraints for variables ξ. The MIP_E can also be seen as a Dantzig-Wolfe decomposition of (3.2)-(3.4).

Formulation MIP_E is inspired by the classical extended formulation for VCP introduced by Mehrotra and Trick [38] presented in Section 1.6 which is one of the most effective formulation for the VCP. We address the interested reader to Malaguti and Toth [37] and Cornaz, Furini and Malaguti [9] for a discussion on this topic.

We use the covering version of Constraints (3.14) to restrict the domain of the associated dual variables, but the optimal solution of LP_E is not a covering. Since a cost is paid for every vertex in the MSCP, a solution containing a vertex covered more than one time can be replaced by the same solution structure minus this vertex, leading to a valid solution with a smaller objective function.

By replacing constraints (3.15) with the following ones:

$$\xi_S^i \geq 0 \qquad i \in \{1, ..., k\}, S \in \mathscr{S},$$

we obtain the Linear Programming relaxation of MIP_E, denoted as LP_E.

Proposition 9 *The bound provided by LP_E is at least as strong as the bound provided by LP_C, i.e., $z(LP_E) \geq z(LP_C)$; and there are cases in which $z(LP_E) > z(LP_C)$.*

Proof. Let ξ^* be an optimal solution of MIP_E. From this solution, we construct a solution to MIP_C and we prove that this solution is feasible for MIP_C and has the same objective function in MIP_C and in MIP_E. Let us denote this solution as :

$$x_v^i = \sum_{S \in \mathscr{S} : v \in S} \xi_S^i.$$

Replacing these variables into the objective function of MIP_C, we obtain

$$\sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i = \sum_{i \in \{1,...,k\}} \sum_{v \in V} \left(i \cdot \sum_{S \in \mathscr{S} : v \in S} \xi_S^{i*} \right)$$

$$\sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i = \sum_{i \in \{1,...,k\}} \sum_{v \in V} \sum_{S \in \mathscr{S} : v \in S} \left(i \cdot \xi_S^{i*} \right)$$

$$\sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i = \sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}} \left(i \cdot |S| \right) \xi_S^{i*}$$

$$\sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i = \sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}} c_S^i \xi_S^{i*}$$

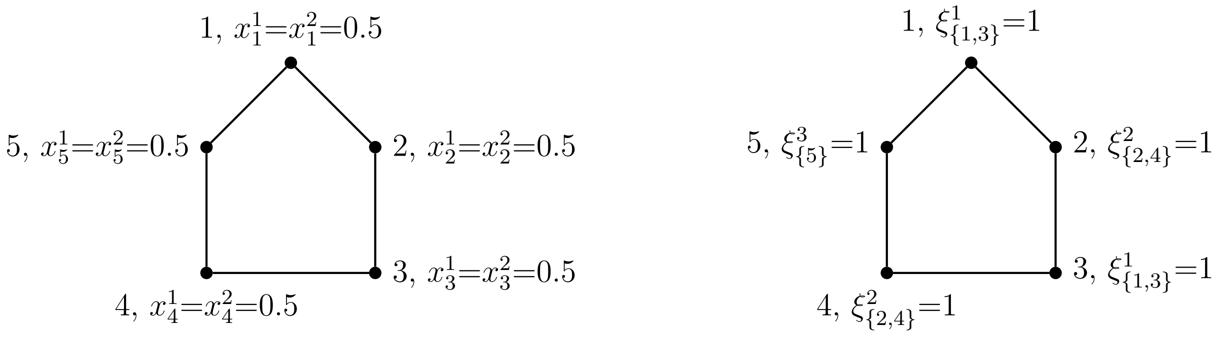

Figure 3.4: Optimal solution of LP_C and LP_E on a G.

which is the objective function of MIP_E. We now replace these variables in Constraints (3.2):

$$x_u^i + x_v^i \leq 1$$

$$\sum_{S \in \mathscr{S}, u \in S} \xi_S^{i*} + \sum_{S \in \mathscr{S}: v \in S} \xi_S^{i*} \leq 1$$

Since vertices u and v in Constraints (3.2) are adjacent, there exists no stable set S such that $u, v \in S$. Thus, the variables in $\sum\limits_{S \in \mathscr{S}, u \in S} \xi_S^{i*}$ and $\sum\limits_{S \in \mathscr{S}: v \in S} \xi_S^{i*}$ are different. Moreover, we have $\sum\limits_{S \in \mathscr{S}, u \in S} \xi_S^{i*} < 1$ since ξ^* is optimal. Constraints (3.2) are not violated. We now replace these variables in Constraints (3.3):

$$\sum_{i \in \{1,\ldots,k\}} x_v^i = 1$$

$$\sum_{i \in \{1,\ldots,k\}} \sum_{S \in \mathscr{S}: v \in S} \xi_S^{i*} = 1$$

Since ξ^* is optimal, we have $\sum\limits_{S \in \mathscr{S}, u \in S} \xi_S^{i*} = 1$, thus Constraints (3.3) are not violated. This new solution of MIP_C we built with optimal variables ξ_S^i is valid and holds the same value.

We now present a case where $z(LP_C) < z(LP_E)$. Let us consider a graph G which is a cycle of size 5 where vertices are numbered consecutively. Following Proposition 8, we have $z(LP_C) = 7.5$. The optimal value of $z(LP_E)$ is 9, obtained with variables $\xi^1_{\{1,3\}} = 1$, $\xi^2_{\{2,4\}} = 1$ and $\xi^3_{\{5\}} = 1$ (see Figure 3.4). This solution is integer, thus $\Sigma(G) = 9$. □

In the next subsections we describe the major components of the algorithm solving MIP_E.

Column generation for MIP_E

Since MIP_E has exponentially many ξ^i_S variables, *Column Generation* (CG) techniques are necessary to efficiently solve LP_E. Let us denote by π_i the dual variables associated with Constraints (3.13) and by μ_v the dual variables associated with Constraints (3.14). The dual problem of the continuous relaxation of MIP_E reads as follows:

$$\max \sum_{v \in V} \mu_v + \sum_{i \in \{1,...,k\}} \pi_i \tag{3.16}$$

$$\sum_{v \in S} \mu_v + \pi_i \leq c^i_S, \qquad S \in \mathscr{S}, i \in \{1,...,k\}, \tag{3.17}$$

$$\mu_v \geq 0, \qquad v \in V, \tag{3.18}$$

$$\pi_i \leq 0, \qquad i \in \{1,...,k\}. \tag{3.19}$$

In order to solve LP_E which is the same value as the optimal solution of its dual (3.16)-(3.19), we do not enumerate all variables ξ^i_S which is highly impracticable. The reduced cost of a variable ξ^i_S is

$$\tilde{c^i_S} = c^i_S - \left(\sum_{v \in S} \mu_v + \pi_i \right) = i \cdot |S| - \sum_{v \in S} \mu_v - \pi_i.$$

which is also the value of the violation of any constraint (3.17). Hence, to solve LP_E by CG, we initialize the model with a subset of variables containing a feasible solution, and, given an optimal primal-dual solution $(\xi^*, (\pi^*, \mu^*))$ to this restricted master problem, we iteratively add variables with negative reduced cost. For a given color i, the goal of the subproblem is to ensure that no negative reduced cost variables exist. To achieve that, the variable with the minimum reduced cost is sought:

$$\max_{S \in \mathscr{S}} \left\{ \sum_{v \in S} \mu_v + \pi_i - i \cdot |S| \right\}.$$

Finding this variable is obtained by solving a MWSS on G, where the weight of a vertex v is defined as $\mu^*_v - i$. Clearly, k pricing problems have to be solved at each iteration of the CG scheme, one for each possible color. When no stable set with negative reduced cost exists, the rounded-up value of the optimal solution value of LP_E provides a valid lower bound for the MSCP. We solve the MWSS by means of the combinatorial Branch-and-Bound algorithm proposed by Held et al. [23].

The weights of the vertices $\mu^*_v - i, v \in V$ can be negative which MWSS does not support. An easy fix is to remove the vertices with negative weights by setting the weight of those vertices to 0 in MWSS. They still might be taken in the solution given by the MWSS algorithm. We remove from the solution the vertices with negative weights.

In order to speed up the convergence of the CG procedure we propose the following idea. In addition, we can limit the number of pricing problems that have to be solved at each CG iteration. As soon as no reduced cost column is found for a color i such that the corresponding constraint (3.13) is < 1 then no color $j > i$ can generate columns with negative reduced costs. Indeed, we would have $\pi_i = 0$ and $\pi_j = 0\ \forall j > i$. Since the weights of the vertices in the maximum weight stable set pricing problems are decreasing for increasing values of j, if there is no reduced cost variable for j, there cannot be for colors with a larger index.

Initialization procedure for MIP_E

As previously stated, MIP_E has an exponential number of variables, thus a CG algorithm is used to solve LP_E. This implies that not all variables ξ^i_S are in MIP_E. The initial set of variables for MIP_E must be a feasible solution.

For the VCP, it is sufficient to initialize the column generation MIP model (see Section 1.6) with stable set variables corresponding to a singleton of each vertex of the graph. If $k < n$, we cannot apply the same technique. Let us suppose for each $v \in V$ and for each $i \in \{1, ..., k\}$, a variable $\xi^i_{\{v\}}$ is created. With this set of variables, all covering vertices constraints (3.14) cannot be satisfied since at least two vertices u and v would be assigned the same color i, which means that the corresponding constraint (3.13) would be violated.

For the MSCP, the initial set of variables used in all our computation results is obtained by a simple heuristic creating a feasible solution. A vertex v is chosen from the graph $G = (V, E)$, and a stable set S is iteratively constructed from V. Once vertices can no longer be added to S, S is added as a variable in MIP_E and the process starts again without the vertices from the previously constructed stable set S (see Algorithm 4).

This also provides MIP_E with an initial upper bound since Algorithm 4 creates a feasible solution for MIP_E.

Algorithm 4: Initialization procedure for MIP_E

```
Data: $G = (V, E)$: graph to color
Result: $MIP_E$ is initialized with a feasible solution for the MSCP
$i = 1$;
while $V$ is not empty do
    $S = \emptyset$;
    for each vertex $w \in V$ do
        if $w$ is not adjacent to any vertices of $S$ then
            $S = S \cup \{w\}$;
            $V = V \backslash \{w\}$;
        end
    end
    Add variable $\xi_S^i$ to $MIP_E$;
    $i = i + 1$;
end
```

Branch-and-Price algorithm for MIP_E

To complete the algorithm dedicated to solving MIP_E, we define a branching rule for MIP_E, thus defining a complete Branch-and-Price algorithm. We use and adapt a branching rule derived from Zykov [60] for the VCP. We use the following proposition from Barnhart et al. [1].

Proposition 10 (Barnhart et al. [1]) *If A is a $0-1$ matrix, and a basic solution to $A\lambda$ is fractional, i.e., at least one of the components of λ is fractional, then there exists two rows r and s of the master problem such that*

$$0 < \sum_{k:y_{rk},y_{sk}} \lambda_k < 1.$$

This proposition can be applied to MIP_E. Constraints (3.13) can be written as

$$\sum_{S\in\mathscr{S}} \xi_S^i = 1, \qquad i \in \{1, ..., k\} \tag{3.20}$$

by adding extra variables. For each $i \in \{1, ..., k\}$, variables ξ_S^i with $S = \emptyset$ and no coefficient in the objective are added to the model. This implies that Constraints (3.20) can be satisfied with theses variables, without altering the solution and Constraints (3.14).

Constraints (3.14) can also be written as

$$\sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S} : v \in S} \xi_S^i = 1, \qquad v \in V, \tag{3.21}$$

since it does not change the structure of the solution.

Thus, with Constraints (3.20) and (3.21), Proposition 10 can be applied to MIP_E. Let us consider a fractional solution ξ. There exists two non-adjacent vertices u and v such that

- $\exists S_1 \in \mathscr{S}$ such that either $u \in S_1, v \notin S_1$ or $u \notin S_1, v \in S_1$ and $0 < \sum_{i \in \{1,...,k\}} \xi_{S_1}^i < 1$,
- $\exists S_2 \in \mathscr{S}$ such that $u, v \in S_2$ and $0 < \sum_{i \in \{1,...,k\}} \xi_{S_2}^i < 1$.

The branching rule is defined as follows. Two children nodes are created:

- In the first child node, vertices u and v are assigned the same color. All variables ξ_S^i such that $v \in S, u \notin S$ or $v \notin S, u \in S$, for all $i \in \{1,...,k\}$ are set to 0. This prevents the solution from assigning different colors for vertices u and v. The pricing subproblem must be modified as well in order to respect this branching decision. In the graph G for this specific subproblem, vertices u and v are merged, i.e., both u and v are removed from G and a new vertex w is added to G with neighborhood $N_G(w) = N_G(u) \cup N_G(v)$.
- In the second child node, vertices u and v are assigned different colors. All variables ξ_S^i such that $v \in S, u \in S$ for all $i \in \{1,...,k\}$ are set to 0. This prevents the solution from assigning the same color to vertices u and v. The pricing subproblem must be modified as well in order to respect this branching decision. In the graph G for this specific subproblem, edge (u, v) is added to G.

An easy way to enforce the branching decisions while using MWSS [23] is computing a reduced graph following the same procedure as in Section 2.4 and giving it to the MWSS algorithm.

Lagrangian bound for MIP_E

Considering an iteration of the CG algorithm, a lower bound on $z(LP_E)$, denoted z_D, can be derived from the current dual solution. This lower bound can potentially be used as a stopping condition of the CG algorithm. The following Proposition describes this lower bound.

Proposition 11 *Let (π^*, μ^*) be the current solution of the restricted dual problem during an iteration of the CG algorithm and c_S^{i*} the smallest reduced cost. A feasible dual solution (π^{**}, μ^{**}) to the master problem can be constructed as follows:*

$$\begin{aligned} \mu_v^{**} &= \mu_v^*, & v \in V, \\ \pi_i^{**} &= \pi_i^* - c_S^{i*}, & i \in \{1, ..., k\}. \end{aligned}$$

Its value is a valid lower bound for $z(LP_E)$.

Proof. Let (π^{**}, μ^{**}) be such a solution. Let us replace those variables in constraint (3.17):

$$\sum_{v \in S} \mu_v^{**} + \pi_i^{**} = \sum_{v \in S} \mu_v^* + \pi_i^* - c_S^{i*}, \qquad S \in \mathscr{S}, i \in \{1, ..., k\}.$$

Since c_S^{i*} is optimal, it is the value of the largest violation for all constraints (3.17), i.e., $c_S^{i*} = \max\limits_{S \in \mathscr{S}} \{c_S^i - \sum\limits_{v \in S} \mu_v^* + \pi_i^*\}$. We have:

$$\sum_{v \in S} \mu_v^* + \pi_i^* - c_S^{i*} = \sum_{v \in S} \mu_v^* + \pi_i^* - \max_{S \in \mathscr{S}} \{c_S^i - \sum_{v \in S} \mu_v^* + \pi_i^*\} \leq c_S^i, \qquad S \in \mathscr{S}, i \in \{1, ..., k\}.$$

This means if we remove this value from $\sum\limits_{v \in S} \mu_v^* + \pi_i^*$ no constraint can be violated, thus (π^{**}, μ^{**}) is a valid feasible solution for LP_E and its value is $z_D = \sum\limits_{v \in V} \mu_v + \sum\limits_{i \in \{1,...,k\}} \pi_i - k \cdot c_S^{i*}$. $\square$

Figure 3.5 shows the slow convergence of our current CG algorithm on instance DSJC125.5 towards the optimal LP value of MIP_E. The vertical axis is the value (LP or Lagrangian bound) and the horizontal axis is the computation time. The curve "LP" represents the value of $z(LP_E)$ during the CG algorithm. We can see that the curve describing the evolution of the LP value is far from linear. From 0 seconds to 100 iterations, the LP value goes from 1400 to approximately 990. From this value, it takes more than 500 iterations to obtain the LP of the MP which is 5 times the

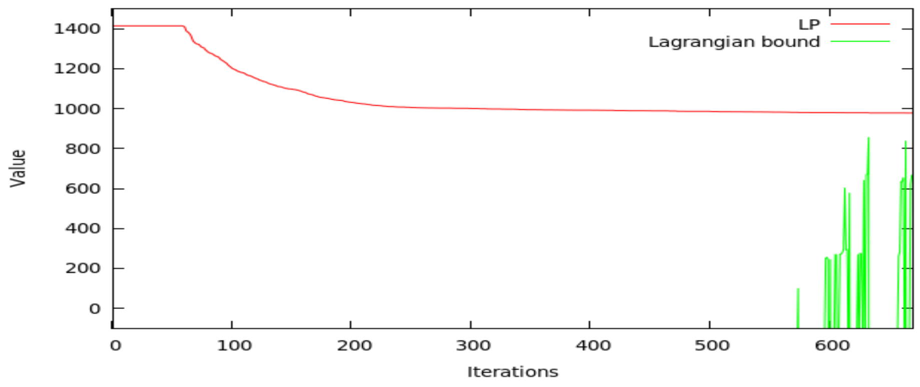

Figure 3.5: LP value and computation time of instance DSJC125.5.col

number of iterations it took to go from 1400 to approximately 990. It shows that when approaching a near optimal LP value, the algorithm struggles to find the optimal LP value even though it is close.

This phenomenon is known as the tailing off effect. As described in [35], a main explanation for the tailing off effect is that the dual variables converge slowly towards their respective optimal value, and, from one iteration to another, take unrelated and random values. The following sections aim at decreasing this effect, which will improve the efficiency of MIP_E.

Computational results

In this section we present the results of computational experiments on two benchmark sets of instances. The first set is composed by random instances proposed by San Segundo [51] with 60 and 70 vertices and density d varying from 0.1 to 0.9. We complement this subset with instances with 30, 40 and 50 vertices varying from 0.1 to 0.9. For each value of d and n, we select five instances (225 instances in total). The second set is a subset of the DIMACS instances already used in the literature to compute upper bounds and lower bounds for MSCP (see [26]).

We use the following procedure to get an upper bound for MIP_E. Let us consider a node of the branching tree of the Branch-and-Price algorithm solving MIP_E. After the CG algorithm is completed, we define another ILP initialized with all the variables and constraints of the current model of MIP_E. Then, we let this new model run for 60 seconds as if the formulation was compact, i.e., it is a simple Branch-and-Bound. The upper bound found by this new model gives us an upper bound on $\Sigma(G)$.

We performed all experiments on a icore 7 computer at 3.4 GHz equipped with 16 GB RAM and using `CPLEX` 12.6 as ILP solver (with default parameters and in single thread mode). We used $k = \Delta(G) + 1$ as an upper bound on the strength of the graph.

To solve the pricing problems, we used the Branch-and-Bound algorithm MWSS proposed by Held, Cook and Sewell [23].

Pricing strategies for MIP_E

In this subsection we discuss two different aspects of the pricing strategy that we use in MIP_E. The first aspect is the choice of the LP algorithm solving LP_E at each step of the CG algorithm and the second aspect is the number of variables that we add at each round of the CG algorithm.

Adding variables to MIP_E

In this subsection, we will discuss the procedure used in our implementation of MIP_E to add variables ξ to the model. Choosing which variables ξ to add in the model at each step of the CG algorithm is a crucial part of optimizing MIP_E.

We solve the k pricing subproblems iteratively, starting with the subproblem associated to color 1. The pricing subproblems are solved to optimality. If a negative reduced cost variable is found, we stop the pricing procedure and we begin the variable addition phase.

Let S^* be the stable set found with a negative reduced cost for the color i^* at an iteration of the CG algorithm. Starting from this potential variable $\xi_{i^*}^{S^*}$, we devise the following procedures aiming at optimizing the use of this potential variable:

- `nostrat`: the naive strategy which only adds variable $\xi_{S^*}^{i^*}$ in MIP_E.
- `stratall`: the strategy which consists of adding for all $i \in \{1, ..., k\}$, non-existing variables $\xi_{S^*}^{i}$ in MIP_E. The aim of this strategy is to give potentially interesting variables to MIP_E for all colors.
- `redcost`: the strategy which consists of adding for all $i \in \{1, ..., k\}$ adding non-existing variables $\xi_{S^*}^{i}$ in MIP_E if their associated reduced cost is negative. This is a more refined version of the `stratall` strategy which limits the added variables to promising ones.

Algorithm	`nostrat`	`stratall`	`redcost`	`completion`
Completed instances	15%	25%	30%	30%

Table 3.1: Comparison between different pricing strategies for solving LP_E

- `completion`: this strategy is focused on finding good UB for MIP_E. Variable $\xi^{i^*}_{S^*}$ is added in MIP_E. Starting with this variables, we apply Algorithm 4 in order to find a feasible solutions including S^* in color i^*. All variables corresponding to this feasible solution are added in MIP_E and UB is updated if the value of the feasible solution is better. This strategy has been used successfully in [21] for the VCP.

We used the DIMACS benchmark to test each of these procedures for adding variables to MIP_E. In Table 3.1 we report the percentage of completed instances for each procedure. If the value of $z(LP_E)$ for a given instance of the DIMACS benchmark is computed, the instance is considered completed in Table 3.1.

According to Table 3.1, computing $z(LP_E)$ with the `nostrat` strategy gives us the lowest completion rate. Since only one variable is added at each round of the pricing, the number of iterations required increases. With the three other strategies, more than one variable are added, which potentially increases the computation time of $z(LP_E)$. The `stratall` strategy gives us the second lowest completion rate. Too many variables are added, especially potentially non-relevant ones, which slows down the computation time of the LP and thus the model. Both `redcost` and `completion` strategies hold the same completion rate.

Table 3.2 describes the results of these two strategies in more detail. Each line is now one DIMACS instance from the DIMACS benchmark. Only instances for which $z(LP_E)$ has been computed by at least one strategy are shown which gives us 29 instances in the table. Each entry in the table is the respective computation time of $z(LP_E)$. In Table 3.2, we can see that both strategies can solve the same instances except for one instance for each (resp. flat_300_20 for `redcost` and jean for `completion`). Finally, `redcost` is faster than `completion` for 15 instances against 11 for `completion`. This, in all further computations, `redcost` will be used for the computation of $z(LP_E)$.

LP algorithm

The commercial solver `CPLEX` 12.6 is used in our implementation. It offers different LP algorithms:

Instance	`redcost`	`completion`
2-Insertions_3	0.12	0.08
3-Insertions_3	1.45	0.50
DSJC125.5	49.41	51.42
DSJC125.9	2.78	3.14
DSJC250.9	71.71	84.25
flat300_20_0	96.76	600.01
huck	29.26	35.73
jean	600.02	77.37
miles1000	42.09	44.42
miles1500	1.53	2.24
miles750	345.95	155.37
mug100_1	209.73	76.59
mug100_25	123.25	351.38
mug88_1	9.66	9.94
mug88_25	2.77	2.98
myciel3	0.00	0.00
myciel4	0.01	0.00
myciel5	0.27	0.22
myciel6	13.13	26.54
queen10_10	8.34	10.40
queen11_11	6.84	66.12
queen12_12	130.79	108.96
queen13_13	412.77	183.13
queen5_5	0.00	0.00
queen6_6	0.06	0.06
queen7_7	0.06	0.22
queen8_12	1.36	0.38
queen8_8	2.76	2.63
queen9_9	2.33	5.17

Table 3.2: Comparison between different pricing strategies for solving LP_E

- `Primal simplex` which `CPLEX` advises to use in case the number of columns exceeds the number of constraints.
- `Dual simplex` which is the standard algorithm advised by `CPLEX`.
- `Network optimizer` which `CPLEX` advises to use in case the major part of the problem is structured as a network.

Algorithm	`Primal simplex`	`Dual simplex`	`Network optimizer`	`Barrier optimizer`	`Sifting optimizer`
Completed instances	30%	27%	25%	21%	27%

Table 3.3: Comparison between different `CPLEX` algorithms for solving LP_E

- `Barrier optimizer` which `CPLEX` advises to use in case the number of constraints exceeds the number of columns.
- `Sifting optimizer` which `CPLEX` advises to use in case the holds a particular structure.

We used the DIMACS benchmark to test each algorithm. Table 3.3 is a summary of the comparison of the results of each of the previously mentioned `CPLEX` algorithm for solving LP_E with a time limit of 600 seconds. This table has the same entries as previously defined.

Table 3.3 shows us that the most relevant algorithm for solving LP_E is `Primal simplex` and not `Dual simplex`. As expected `Network optimizer`, `Barrier optimizer` and `Sifting optimizer` have the lowest completion rates compared to `Primal simplex` and not `Dual simplex`. These three algorithms are not tailored for this benchmark of instances or the MSCP. In the rest of our computational results, the `Primal simplex` will be used.

Computational comparison between MIP_C and MIP_E

We now compare computationally MIP_C and MIP_E. In the tables for each DIMACS instance we report:

- *name*: the name of the instance
- n: the number of vertices
- m: the number of edges
- d: the density
- LB_t: the best lower bound known in the literature for the DIMACS instances ([26])

- UB_t: the best upper bound known in the literature for the DIMACS instances ([26])

For random instances, the column identified by # reports the number of instances with the same value of n and d. All additional information for random instances are averaged over the number of instances for a given entry. A "tl" indicates that the time limit was reached for an instance and a "-" indicates that the value could not be computed (within the time limit). Table 3.4 compares the results of the two formulations MIP_C and MIP_E on the first benchmark of 225 random instances.

For MIP_C, we report:

- UB_C: the value of the best upper bound
- LB_C: the value of the best lower bound
- t_C: the corresponding computation time
- $nodes_C$: the number of nodes

For MIP_E, we report:

- UB_E: the value of the best upper bound
- $nodes_E$: the number of nodes
- t_E^{UB}: the corresponding computation time for UB_E
- LB_E: the value of the linear relaxation of MIP_E
- t_E^{LB}: the corresponding computation time for LB_E

On this benchmark of instances, MIP_E is overall more efficient than MIP_C as showed in Table 3.4. For $n = 50$ and $d = 0.2, 0.3$, $n = 60$ and $d = 0.1, 0.2, 0.3$, $n = 70$ and $d = 0.1$, MIP_C has less fails than MIP_E. MIP_E is less efficient for sparse instances. For all other instances, MIP_E has less or equal fails than MIP_C. MIP_E and MIP_C totalize respectively 50 and 113 fails over 225 instances which is close to half of the benchmark for MIP_C. When MIP_E and MIP_C both solve a subset of instances with same size and density, i.e., 0 fails, MIP_E is faster on instances with $d \geq 0.4$ while MIP_C is faster for instances with $d \leq 0.3$ (except for $n = 30$ and $d = 0.2$). For $n = 70$ and $d = 0.2, 0.3, 0.4, 0.5$, MIP_C and MIP_E have the same number of fails. On those

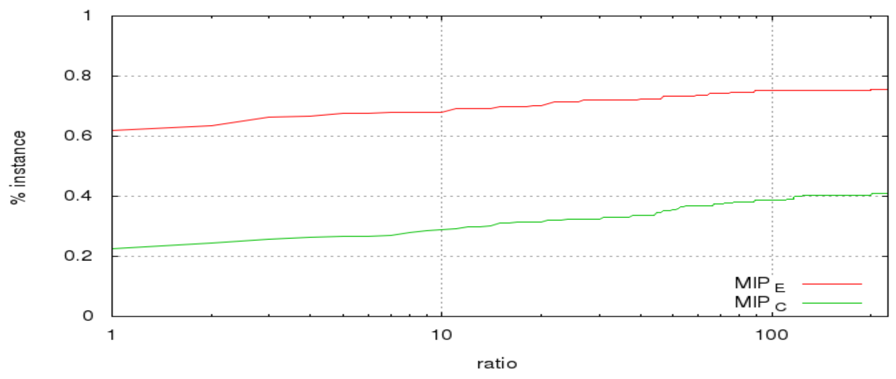

Figure 3.6: Performance profile of formulations MIP_E and MIP_C on a benchmark of random instances.

five set of instances, MIP_E always has the best lower bound and has the best upper bound three times. MIP_C has a better lower bound for two set of instances with low density $d = 0.2, 0.3$. Finally, MIP_E more efficient than MIP_C for this benchmark of instances.

Figure 3.6 presents the performance profile of formulations MIP_E and MIP_C for the random benchmark of instances. For a complete explanation of performances profiles, see the Appendix. This performance profile shows that MIP_E is more efficient than MIP_C no matter the time ratio. For every possible value of the ratio, MIP_E is able to be more efficient on a larger percentage of instances than MIP_C. Finally, MIP_E is able to solve 77% of the benchmark against 60% for MIP_C.

In Table 3.5 we report the results for DIMACS instances. On 93 instances, MIP_E is able to solve 20 instances and MIP_C is able solve 44. This is largely due to the fact that MIP_E is only able to compute LB_E for 32 instances. On those 32 instances, MIP_E is able to solve two thirds. Every time LB_E can be computed LB_E is at least equal to LB_t and for 21 instances, it is better, sometimes by a very large difference (for example DSJC500.9). The gap $UB_E - \lceil LB_E \rceil$ is strictly superior than the gap $UB_C - \lceil LB_C \rceil$ for 7 instances, the opposite is true for 10 instances. Since LB_E is far better than LB_C this means that the weakness of the gap $UB_E - \lceil LB_E \rceil$ comes from the weakness of bound UB_E.

Figure 3.7 presents the performance profile of formulations MIP_E and MIP_C for the DIMACS benchmark of instances. For a complete explanation of performances profiles, see the Appendix. As previously showed, on this benchmark of instances MIP_C is more efficient than MIP_E no matter the time ratio. For every possible value of the ratio,

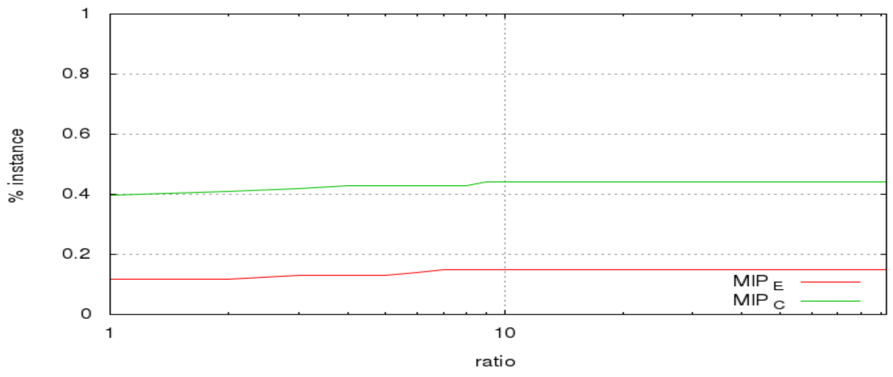

Figure 3.7: Performance profile of formulations MIP_E and MIP_C on a benchmark of DIMACS instances.

name	n	m	d	LP_C^2	LP_C	LP_E
2-Insertions_3	37	72	0.1	61	56	62
3-Insertions_3	56	110	0.1	90	84	92
DSJC125.1	125	736	0.1	219	188	237
DSJC125.9	125	6961	0.9	1585	188	2500
mug100_1	100	166	≈0.0	156	192	202
mug100_25	100	166	≈0.0	155	192	202
mug88_1	88	146	≈0.0	137	132	178
mug88_25	88	146	≈0.0	137	132	178
myciel3	11	20	0.4	20	17	21
myciel4	23	71	0.3	42	35	44
myciel5	47	236	0.2	84	71	88
myciel6	95	755	0.2	167	143	176

MIP_C is able to solve a larger percentage of instances than MIP_E. Finally, MIP_C is able to solve 42% of the benchmark against 18% for MIP_E.

In Table 3.4.2, we present preliminary results for MIP_C^2. This compact model has a large number of variables and constraints and we could compute the linear relaxation only for a few DIMACS instances (12). We present those values and compare them to both LP_C and LP_E. For 10 instances out of 12, $z(LP_C^2) > z(LP_C)$, LP_C^2 is potentially stronger on average than LP_C. As for LP_C, we have $z(LP_C^2) < z(LP_E)$.

MIP_E is clearly the more efficient model for the benchmark of random instances. For the DIMACS instances, MIP_C is the more efficient in terms of solved instances, but MIP_E has the potential to solve more instances which we believe is not the case for MIP_C.

Instance			MIP_E						MIP_C				
n	d	#	UB_E	nodes$_E$	t_E^{UB}	LB_E	t_E^{LB}	fail	UB_C	nodes$_C$	t_C	LB_C	fail
30	0.1	5	47.6	1	0.24	47.6	0.21	0	47.6	3	0.00	47.6	0
30	0.2	5	59.4	2	0.2	59.3	0.13	0	59.4	4	0.08	59.4	0
30	0.3	5	72	4	0.3	71.5	0.10	0	71.8	5	0.43	72.0	0
30	0.4	5	84.2	1	0.15	83.9	0.08	0	84.2	6	0.91	84.2	0
30	0.5	5	101.4	2	0.1	101.1	0.05	0	101.2	7	2.62	101.4	0
30	0.6	5	119.2	11	0.32	118.4	0.04	0	118.6	9	10.32	119.2	0
30	0.7	5	138.2	8	0.21	137.1	0.03	0	137.4	10	59.02	138.2	0
30	0.8	5	162	3	0.05	161.6	0.02	0	161.6	12	32.89	162.0	0
30	0.9	5	216.2	2	0.02	215.8	0.01	0	215.8	16	721.23	215.8	1
40	0.1	5	69	5	2.5	68.7	0.65	0	68.8	3	0.07	69.0	0
40	0.2	5	88.4	3	1.03	88.2	0.42	0	88.4	5	0.51	88.4	0
40	0.3	5	113.6	103	31.85	110.8	0.34	0	111.2	6	9.98	113.6	0
40	0.4	5	133	169	44.49	130.9	0.28	0	131.2	7	72.52	133.0	0
40	0.5	5	159	7	2.89	158.1	0.21	0	158.2	9	361.10	159.0	0
40	0.6	5	180.8	9	0.81	179.8	0.18	0	180.0	10	1072.89	180.8	0
40	0.7	5	214.6	3	0.25	213.7	0.11	0	214.0	12	3497.02	210.4	4
40	0.8	5	268	7	0.35	266.8	0.06	0	267.0	15	3231.07	262.6	4
40	0.9	5	347.4	9	0.1	346.8	0.03	0	346.8	19	907.07	346.4	1
50	0.1	5	90	7	9.59	89.5	2.17	0	89.6	4	0.20	90.0	0
50	0.2	5	124.6	578	1594.08	121.3	1.57	2	121.8	5	14.24	124.0	0
50	0.3	5	160	861	984.58	156.8	1.04	1	157.4	7	501.72	160.0	0
50	0.4	5	191.6	160	844.47	188.4	0.73	1	189.0	9	3506.78	186.3	4
50	0.5	5	224.4	253	101.69	222.1	0.55	0	222.6	10	3401.31	211.6	4
50	0.6	5	269.4	60	18.81	267.1	0.36	0	267.4	12	tl	241.7	5
50	0.7	5	317.8	36	5.86	316.4	0.27	0	316.6	14	tl	283.1	5
50	0.8	5	384.4	3	0.37	384.2	0.16	0	384.2	17	tl	340.9	5
50	0.9	5	516.4	3	0.15	515.9	0.08	0	516.0	24	tl	495.5	5
60	0.1	5	124.4	173	2588.48	118.7	5.63	3	119.2	4	1.35	120.0	0
60	0.2	5	169	252	tl	158.1	3.88	5	158.8	6	475.37	162.2	0
60	0.3	5	205.8	236	2865.18	195.9	3.11	3	196.4	7	tl	189.2	5
60	0.4	5	247.4	737	2540.49	240.8	1.71	3	241.2	9	tl	216.7	5
60	0.5	5	302	673	1792.25	296.7	1.29	2	297.0	11	tl	249.9	5
60	0.6	5	353.6	964	545.54	350.8	0.89	0	351.4	13	tl	291.2	5
60	0.7	5	433.6	14,378	785.5	429.7	0.60	1	430.2	17	tl	353.5	5
60	0.8	5	520.6	3	1.27	519.7	0.38	0	520.2	20	tl	433.4	5
60	0.9	5	674.2	2	0.33	673.9	0.17	0	674.0	25	tl	608.9	5
70	0.1	5	153.2	107	2987.13	143.9	18.14	4	144.4	4	11.59	146.0	0
70	0.2	5	217.6	92	tl	199.2	12.69	5	199.4	7	tl	193.6	5
70	0.3	5	277.8	93	tl	254.0	6.75	5	254.6	8	tl	224.2	5
70	0.4	5	328.4	111	tl	312.0	4.06	5	312.4	10	tl	253.0	5
70	0.5	5	390.2	343	tl	379.1	2.47	5	379.6	13	tl	294.4	5
70	0.6	5	466.6	143	2576.28	458.9	1.60	3	459.4	15	tl	358.5	5
70	0.7	5	545	88	1496.36	539.8	1.21	2	540.4	18	tl	418.6	5
70	0.8	5	683	107	66.73	678.6	0.74	0	679.0	23	tl	542.6	5
70	0.9	5	907.4	9	0.86	905.4	0.27	0	905.6	30	tl	774.8	5

Table 3.4: Comparison between MIP_C and MIP_E on a benchmark of 225 random instances

Instance						MIP_E					MIP_C		
name	n	m	d	LB_t	UB_t	UB_E	nodes$_E$	t_E^{UB}	LB_E	t_E^{LB}	UB_C	t_C	LB_C
2-Insertions_3	37	72	0.1	55	62	62	1	0.10	62	0.10	62	0.23	62
3-Insertions_3	56	110	0.1	84	92	92	1	1.10	92	0.88	92	0.21	92
anna	138	493	0.1	273	276	9591	1	tl	-	0.00	276	0.15	276
C2000.5	2000	999836	0.5	15091	132483	-	1	tl	-	tl	-	tl	-
david	87	406	0.1	234	237	280	1	tl	-	tl	237	0.11	237
DSJC1000.1	1000	49629	0.1	2762	8991	-	1	tl	-	tl	12892	tl	0
DSJC1000.5	1000	249826	0.5	6708	37575	57246	1	tl	-	tl	57246	528.84	0
DSJC1000.9	1000	449449	0.9	26557	103445	148939	1	tl	-	tl	-	tl	-
DSJC125.1	125	736	0.1	247	326	412	1	tl	-	tl	340	tl	281.166
DSJC125.5	125	3891	0.5	549	1012	1065	76	tl	978	50.15	1206	tl	577.101
DSJC125.9	125	6961	0.9	1691	2503	2503	39	56.70	2500	2.34	3154	tl	1465.01
DSJC250.1	250	3218	0.1	570	970	1325	1	tl	-	tl	1147	tl	601.905
DSJC250.5	250	15668	0.5	1287	3210	4587	1	tl	-	tl	4587	tl	1227.67
DSJC250.9	250	27897	0.9	4311	8277	8315	74	tl	8235	72.62	11494	tl	970.468
DSJC500.1	500	12458	0.1	1250	2836	-	1	tl	-	tl	4110	tl	1293.69
DSJC500.5	500	62624	0.5	2923	10886	15878	1	tl	-	tl	-	tl	-
DSJC500.9	500	224874	≈1.0	1.0	29862	39776	11	tl	29783	2188.28	-	tl	-
DSJR500.1c	500	121275	≈1.0	15398	16286	16294	54	tl	16234	489.62	20651	1583.40	914
DSJR500.1	500	3555	≈0.0	2069	2156	-	1	tl	-	tl	2149	tl	2126.99
DSJR500.5	500	58862	0.5	22974	25440	31365	1	tl	-	tl	31365	tl	0
flat1000_50_0	1000	245000	0.5	6601	25500	56217	1	tl	-	tl	-	tl	-
flat1000_60_0	1000	245830	0.5	6640	30100	55741	1	tl	-	tl	-	tl	-
flat1000_76_0	1000	246708	0.5	6632	37164	55116	1	tl	-	tl	55116	1522.54	0
flat300_20_0	300	21375	0.5	1531	3150	3150	1	304.20	3150	297.67	6334	tl	1443.26
flat300_26_0	300	21633	0.5	1548	3966	-	1	tl	-	tl	6078	tl	1458.29
flat300_28_0	300	21695	0.5	1547	4238	6237	1	tl	-	tl	6237	tl	1477.6
fpsol2.i.1	496	11654	0.1	3403	3403	8376	1	tl	-	tl	3403	19.37	3403
fpsol2.i.2	451	8691	0.1	1668	1668	7461	1	tl	-	tl	1668	18.65	1668
fpsol2.i.3	425	8688	0.1	1636	1636	7435	1	tl	-	tl	1636	18.58	1636
games120	120	638	0.1	442	443	511	1	tl	-	tl	443	0.39	443
homer	561	1629	≈0.0	1129	1150	-	1	tl	-	tl	1150	3.26	1150
huck	74	301	0.1	243	243	243	1	111.30	243	61.43	243	0.05	243
inithx.i.1	864	18707	0.1	3676	3676	14521	1	tl	-	tl	3676	77.38	3676
inithx.i.2	645	13979	0.1	2050	2050	11824	1	tl	-	tl	2050	63.34	2050
inithx.i.3	621	13969	0.1	1986	1986	11789	1	tl	-	tl	1986	61.95	1986
jean	80	254	0.1	216	217	3240	1	3299.50	-	0.00	217	0.04	217
latin_square	900	307350	0.8	40950	41444	75848	1	tl	-	tl	75848	290.70	0
le450_15a	450	8168	0.1	2331	2632	-	1	tl	-	tl	3418	tl	2446.76
le450_15b	450	8169	0.1	2348	2632	-	1	tl	-	tl	2876	tl	2471.44
le450_15c	450	16680	0.2	2610	3487	-	1	tl	-	tl	5302	tl	2915.78
le450_15d	450	16750	0.2	2628	3505	-	1	tl	-	tl	5421	tl	2927.2
le450_25a	450	8260	0.1	3003	3153	-	1	tl	-	tl	3258	tl	3114.29
le450_25b	450	8263	0.1	3305	3365	-	1	tl	-	tl	3379	tl	3330.24
le450_25c	450	17343	0.2	3657	4515	-	1	tl	-	tl	5897	tl	4051.23
le450_25d	450	17425	0.2	3698	4544	-	1	tl	-	tl	6096	tl	4067.96
le450_5a	450	5714	0.1	1193	1350	-	1	tl	-	tl	1497	tl	1251.24
le450_5b	450	5734	0.1	1189	1350	-	1	tl	-	tl	1405	tl	1246.69
le450_5c	450	9803	0.1	1278	1350	2924	1	tl	-	tl	1350	tl	1338.07
le450_5d	450	9757	0.1	1282	1350	3225	1	tl	-	tl	1350	tl	1337.49
miles1000	128	3216	0.4	1623	1666	1666	1	26.10	1666	18.11	1666	39.89	1666
miles1500	128	5198	0.6	3239	3354	3354	1	1.40	3354	0.78	3354	12.07	3354
miles250	128	387	≈0.0	318	325	374	1	tl	-	tl	325	0.13	325
miles500	128	1170	0.1	686	705	705	24	1953.00	705	484.37	705	17.80	705
miles750	128	2113	0.3	1145	1173	1173	8	541.00	1173	89.73	1173	77.59	1173
mug100_1	100	166	≈0.0	188	202	207	49	tl	202	107.73	202	0.92	202
mug100_25	100	166	≈0.0	186	202	202	55	2970.70	202	153.85	202	2.11	202
mug88_1	88	146	≈0.0	164	178	178	28	909.60	178	26.13	178	0.75	178
mug88_25	88	146	≈0.0	162	178	178	51	1863.50	178	16.98	178	0.56	178
mulsol.i.1	197	3925	0.2	1957	1957	3223	1	tl	-	tl	1957	1.72	1957
mulsol.i.2	188	3885	0.2	1191	1191	3227	1	tl	-	tl	1191	2.39	1191
mulsol.i.3	184	3916	0.2	1187	1187	17020	1	tl	-	tl	1187	2.49	1187
mulsol.i.4	185	3946	0.2	1189	1189	3268	1	tl	-	tl	1189	2.54	1189
mulsol.i.5	186	3973	0.2	1160	1160	3271	1	tl	-	tl	1160	2.51	1160
myciel3	11	20	0.4	16	21	21	1	0.00	21	0.00	21	0.01	21
myciel4	23	71	0.3	34	45	45	16	0.60	44	0.01	45	0.25	45
myciel5	47	236	0.2	70	93	93	1,416	tl	88	0.32	93	1.02	93
myciel6	95	755	0.2	142	189	210	71	tl	176	17.69	189	10.68	189
myciel7	191	2360	0.1	286	381	438	3	tl	349	3287.95	383	tl	349.083
qg.order30	900	26100	0.1	13950	13950	-	1	tl	-	tl	13950	268.62	13950
qg.order40	1600	62400	≈0.0	32800	32800	42016	1	tl	-	tl	32837	tl	32800
qg.order60	3600	212400	≈0.0	109800	109800	-	1	tl	-	tl	-	tl	-
queen10_10	100	1470	0.3	550	553	593	61	tl	550	8.14	570	tl	550
queen11_11	121	1980	0.3	726	733	726	14	1131.80	726	29.08	749	tl	726
queen12_12	144	2596	0.3	936	943	1128	18	tl	936	146.09	962	tl	936
queen13_13	169	3328	0.2	1183	1191	1523	5	tl	1183	915.64	1258	tl	1183
queen14_14	196	4186	0.2	1470	1482	1872	1	tl	-	tl	1605	tl	1470
queen15_15	225	5180	0.2	1800	1814	2296	1	tl	-	tl	1889	tl	1800
queen16_16	256	6320	0.2	2176	2193	2768	1	tl	-	tl	2329	tl	2176
queen5_5	25	160	0.5	75	75	75	1	0.00	75	0.00	75	0.02	75
queen6_6	36	290	0.5	126	138	138	1	0.10	138	0.04	138	174.90	138
queen7_7	49	476	0.4	196	196	196	1	0.10	196	0.03	196	0.34	196
queen8_12	96	1368	0.3	624	624	625	73	tl	624	4.13	624	2.03	624
queen8_8	64	728	0.4	288	291	291	1	1.70	291	1.49	291	tl	288
queen9_9	81	1056	0.3	405	409	425	93	tl	405	1.03	409	tl	405
school1	385	19095	0.3	2439	2674	-	1	tl	-	tl	5925	tl	2388.56
school1_nsh	352	14612	0.2	2176	2392	-	1	tl	-	tl	5118	tl	2192.79
wap05a	905	43081	0.1	12449	13656	-	1	tl	-	tl	16510	tl	13170.3
wap06a	947	43571	0.1	12454	13773	-	1	tl	-	tl	16675	tl	13255
wap07a	1809	103368	0.1	24800	28617	-	1	tl	-	tl	-	tl	-
wap08a	1870	104176	0.1	25283	28885	-	1	tl	-	tl	-	tl	-
zeroin.i.1	211	4100	0.2	1822	1822	3225	1	tl	-	tl	1822	2.22	1822
zeroin.i.2	211	3541	0.2	1004	1004	-	1	tl	-	tl	1004	2.17	1004
zeroin.i.3	206	3540	0.2	998	998	3146	1	tl	-	tl	998	2.14	998

Table 3.5: Comparison between MIP_C and MIP_E on a benchmark of DIMACS instances

Improvements for MIP_E

Overlapped extended formulation

In the CG model for VCP using stable set variables (see Section 1.6), the set of inclusion-wise maximal stable sets can be used instead of the set of all stable sets. Let us consider a solution of the VCP with vertices having multiple stable sets assigned to them. This is not a problem since the objective function, which is the number of used stable sets, is the same whether or not a given vertex is in one or many stable sets. This allows the model to handle only inclusion-wise maximal stable sets instead of the set of all stable sets which implies having less variables.

Unfortunately, this does not hold for the MSCP. Given a vertex of the graph, the objective function takes into account every stable set this vertex is part of. While this is permitted in constraint (3.14), it is suboptimal to do that.

Consider the following case. Let G be a graph with four vertices and only one edge between vertices 1 and 2. Let us consider that the only two generated stable sets are $S^1_{\{1,2,3\}}$ and $S^2_{\{2,3,4\}}$. The associated solution to choosing these two stable sets give us a value of $3*1+3*2=9$ which is far from the optimal value $\Sigma(G)=5$. Although it is far in terms of value, it is not far structurally, but we are forced to generate stable set $S^2_{\{4\}}$ in order to create the optimal solution of value five.

We propose to introduce the following variables. For each vertex $v \in V$ and each color $i \in \{2,...,k\}$:

$$y_v^i = \begin{cases} 1 \text{ if vertex } v \text{ is suppressed from color } i, \\ 0 \text{ otherwise.} \end{cases}$$

Variables y allows the model to suppress any vertex from a given color it has been assigned to by the ξ variables. Let us reconsider the previous example. With these new variables y, there would be no need to generate stable set S^2_4. Instead, variables y^2_2, y^2_3 would be set to 1 and the optimal solution would be reached. Considering these new variables y, a second version of MIP_E, denoted MIP^1_E can be formulated as follows:

$$z(MIP_E^1) = \min \sum_{i\in\{1,...,k\}} \sum_{S\in\mathscr{S}} c_S^i \xi_S^i - \sum_{v\in V} \sum_{i\in\{2,...,k\}} i \cdot y_v^i \tag{3.22}$$

$$\sum_{S\in\mathscr{S}} \xi_S^i \le 1, \qquad i \in \{1,...,k\}, \tag{3.23}$$

$$\sum_{i\in\{1,...,k\}} \sum_{S\in\mathscr{S}:v\in S} \xi_S^i - \sum_{i\in\{2,...,k\}} y_v^i \ge 1, \qquad v \in V, \tag{3.24}$$

$$y_v^i \le \sum_{S\in\mathscr{S}:v\in S} \xi_S^i, \qquad i \in \{2,...,k\}, v \in V \tag{3.25}$$

$$\xi_S^i \in \{0,1\}, \qquad i \in \{1,...,k\}, S \in \mathscr{S}, \tag{3.26}$$

$$y_v^i \in \{0,1\}, \qquad i \in \{2,...,k\}, v \in V. \tag{3.27}$$

The objective function is as before with the addition of variables y. These variables cost i since the removal of a vertex v from color i must be gained in the new solution. Constraint (3.23) are as previously defined. Constraint (3.24) are now constructed with variables y. If a vertex v is suppressed from a color i, this means that this vertex is no longer covered by a stable set in color i, thus it must be set back in this constraint. Constraint (3.25) allow variables y to be used. For each vertex $v \in V$ and each color $i \in \{2,...,k\}$, a vertex can be suppressed from a color i if and only if it has been covered by a stable set in color i. In any other case, the possibility to remove a vertex v from color i is not available. Constraints (3.26) and (3.27) are respectively the integrality constraints for variables ξ and y.

By replacing constraints (3.26) and (3.27) with the following ones:

$$\xi_S^i \ge 0, \qquad i \in \{1,...,k\}, S \in \mathscr{S},$$
$$y_v^i \ge 0, \qquad i \in \{2,...,k\}, v \in V,$$

we obtain the Linear Programming relaxation of MIP_E^1, denoted as LP_E^1. Let us denote ν_v^i the dual variables associated to constraints (3.25). The dual problem of LP_E^1 reads as follows:

$$\max \sum_{v\in V} \mu_v + \sum_{i\in\{1,...,k\}} \pi_i \tag{3.28}$$

$$\sum_{v\in S}(\mu_v - \nu_v^i) + \pi_i \leq c_S^i, \qquad S \in \mathscr{S}, i \in \{2,...,k\}, \tag{3.29}$$

$$\sum_{v\in S} \mu_v + \pi_i \leq c_S^i, \qquad S \in \mathscr{S}, i = 1, \tag{3.30}$$

$$\nu_v^i - \mu_v \leq -i, \qquad v \in V, i \in \{2,...,k\}, \tag{3.31}$$

$$\mu_v \geq 0, \qquad v \in V, \tag{3.32}$$

$$\pi_i \leq 0, \qquad i \in \{1,...,k\}, \tag{3.33}$$

$$\nu_v^i \leq 0, \qquad v \in V, i \in \{2,...,k\}, \tag{3.34}$$

The reduced cost of a given variable ξ_S^i in MIP_E^1 is

$$\tilde{c_S^i} = c_S^i - \left(\sum_{v\in S}(\mu_v - \nu_v^i) + \pi_i\right) = i\cdot|S| - \sum_{v\in S}(\mu_v + \nu_v^i) - \pi_i,$$

and the problem that must be solved is

$$\max_{S\in\mathscr{S}} \left\{ \sum_{v\in S}(\mu_v - \nu_v^i) + \pi_i - i\cdot|S| \right\},$$

which is a MWSS where the weight of a vertex for a given color i is defined as $\mu_v - \nu_v^i$. The Branch-and-Price algorithm described in Section 3.3.3.3 remains the same for MIP_E^1.

Constraints (3.24) can be generalized as follows:

$$\sum_{i\in\{j,...,k\}} \sum_{S\in\mathscr{S}:v\in S} \xi_S^i - \sum_{i\in\{j+1,...,k\}} y_v^i \geq 1 - \sum_{i\in\{1,...j-1\}} \sum_{S\in\mathscr{S}:v\in S} \xi_S^i, \qquad v \in V, j \in \{1,...,k-1\}. \tag{3.35}$$

These constraints mean that for each vertex and for each color $j \in \{1,...,k-1\}$, vertex v can now be suppressed from a color more than one time. These constraints can potentially increase the value of $z(LP_E)$. The addition of those constraints does not change the Branch-and-Price algorithm, the only difference is the weight of each vertex in the pricing subproblem.

For the random benchmark of instances, MIP_E is more efficient than MIP_E^1. In Table 3.6 we compare MIP_E with the overlapped extended formulation MIP_E^1 described in Section 3.5.2 for the DIMACS benchmark. We removed instances where no version of MIP_E could compute the root node. For MIP_E^1, we report:

- UB_E^1: the value of the best upper bound
- $nodes_E^1$: the number of nodes
- $t_E^{UB^1}$: the corresponding computation time for UB_E^1
- LB_E^1: the value of the linear relaxation of MIP_E^1
- $t_E^{LB^1}$: the corresponding computation time for LB_E^1

MIP_E^1 is able to compute LB_E^1 for 43 instances which is 11 more instances than MIP_E. Moreover, MIP_E^1 is able to solve 2 more instances than MIP_E. The instances solved by these two respective models do not overlap completely as 6 instances are solved by MIP_E^1 and not MIP_E and 4 instances are solved by MIP_E and not MIP_E^1. The respective computing times t_E^{LB} and $t_E^{LB^1}$ do not dominate each other as there exists instances where the gap between these two values is huge in favor of MIP_E and in favor MIP_E^1.

Finally, we present a direct comparison between LP_C, LP_E and LP_E^1 in Table 3.7. Again, we mentioned in the table only a subset of DIMACS instances for which at least one version of MIP_E is able to compute the linear relaxation of the model. The gap between the values of LP_C and both LP_E and LP_E^1 can be very large, up to 15484. On average, the gap between LP_C and LP_E is around 1000. This implies that, in order to close very hard DIMACS instances, we think that MIP_E shows the most potential.

Alternative branching rule for MIP_E

In this subsection, we present another branching rule for MIP_E. For this branching rule, we introduce the following variables. For each vertex $v \in V$ and each color $i \in \{1, ..., k\}$,

$$z_v^i = \begin{cases} 1 \text{ if vertex } v \text{ has been assigned color } i, \\ 0 \text{ otherwise.} \end{cases}$$

The branching rule consists of choosing a vertex v that has not been assigned a color yet and assigning a feasible color i. This is possible with the help of variables z_v^i as we

Instance						MIP_E					MIP_E^1				
name	n	m	d	LB_t	UB_t	UB_E	nodes$_E$	t_E^{UB}	LB_E	t_E^{LB}	UB_E^1	nodes$_E^1$	$t_E^{UB^1}$	LB_E^1	$t_E^{LB^1}$
2-Insertions_3	37	72	0.1	55	62	62	1	0.10	62	0.10	62	4	0.10	62	0.03
3-Insertions_3	56	110	0.1	84	92	92	1	1.10	92	0.88	92	7	0.30	92	0.08
anna	138	493	0.1	273	276	9591	1	tl	-	0.00	276	6	194.90	276	70.86
david	87	406	0.1	234	237	-	1	tl	-	tl	237	1	12.10	237	12.09
DSJC125.1	125	736	0.1	247	326	412	1	tl	-	tl	412	2	tl	313	3067.16
DSJC125.5	125	3891	0.5	549	1012	1065	76	tl	978	50.15	1108	61	tl	978	99.77
DSJC125.9	125	6961	0.9	1691	2503	2503	39	56.70	2500	2.34	2512	193	tl	2500	22.78
DSJC250.9	250	27897	0.9	4311	8277	8315	74	tl	8235	72.62	9650	16	tl	8235	2066.51
DSJC500.9	500	224874	≈1.0	29862	39776	11	1	tl	29783	2188.28	-	-	tl	-	tl
DSJR500.1c	500	121275	≈1.0	15398	16286	16294	54	tl	16234	489.62	16294	63	tl	16234	655.67
flat300_20_0	300	21375	0.5	1531	3150	3150	1	304.20	3150	297.67	-	-	tl	-	tl
games120	120	638	0.1	442	443	511	1	tl	-	tl	511	10	tl	443	1744.62
huck	74	301	0.1	243	243	243	1	111.30	243	61.43	243	44	41.20	243	2.32
jean	80	254	0.1	216	217	3240	1	3299.50	-	0.00	217	31	29.20	217	2.94
miles1000	128	3216	0.4	1623	1666	1666	1	26.10	1666	18.11	1666	1	61.80	1666	61.81
miles1500	128	5198	0.6	3239	3354	3354	1	1.40	3354	0.78	3354	4	10.20	3354	5.21
miles500	128	1170	0.1	686	705	705	24	1953.00	705	484.37	-	-	tl	-	tl
miles750	128	2113	0.3	1145	1173	1173	8	541.00	1173	89.73	1173	19	143.00	1173	54.63
mug100_1	100	166	≈0.0	188	202	207	49	tl	202	107.73	202	61	1438.60	202	82.23
mug100_25	100	166	≈0.0	186	202	202	55	2970.70	202	153.85	202	95	3114.30	202	61.37
mug88_1	88	146	≈0.0	164	178	178	28	909.60	178	26.13	178	124	2481.10	178	22.65
mug88_25	88	146	≈0.0	162	178	178	51	1863.50	178	16.98	178	28	239.30	178	18.69
mulsol.i.1	197	3925	0.2	1957	1957	3223	1	tl	-	tl	1957	6	33.70	1957	20.75
mulsol.i.2	188	3885	0.2	1191	1191	3227	1	tl	-	tl	1194	120	tl	1191	165.88
mulsol.i.3	184	3916	0.2	1187	1187	17020	1	tl	-	tl	1190	171	tl	1187	117.74
mulsol.i.4	185	3946	0.2	1189	1189	3268	1	tl	-	tl	1191	177	tl	1189	147.67
mulsol.i.5	186	3973	0.2	1160	1160	3271	1	tl	-	tl	1161	213	tl	1160	83.32
myciel3	11	20	0.4	16	21	21	1	0.00	21	0.00	21	4	0.00	21	0.00
myciel4	23	71	0.3	34	45	45	16	0.60	44	0.01	45	55	1.40	44	0.01
myciel5	47	236	0.2	70	93	93	1,416	tl	88	0.32	97	4755	tl	88	0.08
myciel6	95	755	0.2	142	189	210	71	tl	176	17.69	215	1067	tl	176	1.79
myciel7	191	2360	0.1	286	381	438	3	tl	349	3287.95	438	76	tl	349	15.68
queen10_10	100	1470	0.3	550	553	593	61	tl	550	8.14	610	66	tl	550	20.76
queen11_11	121	1980	0.3	726	733	726	14	1131.80	726	29.08	928	46	tl	726	70.85
queen12_12	144	2596	0.3	936	943	1128	18	tl	936	146.09	1221	20	tl	936	309.00
queen13_13	169	3328	0.2	1183	1191	1523	5	tl	1183	915.64	1523	5	tl	1183	1949.76
queen5_5	25	160	0.5	75	75	75	1	0.00	75	0.00	75	1	0.00	75	0.00
queen6_6	36	290	0.5	126	138	138	1	0.10	138	0.04	138	6	0.30	138	0.06
queen7_7	49	476	0.4	196	196	196	1	0.10	196	0.03	196	1	0.20	196	0.15
queen8_12	96	1368	0.3	624	624	625	73	tl	624	4.13	627	92	tl	624	16.33
queen8_8	64	728	0.4	288	291	291	1	1.70	291	1.49	291	1	1.90	291	1.85
queen9_9	81	1056	0.3	405	409	425	93	tl	405	1.03	451	83	tl	405	6.63
school1_nsh	352	14612	0.2	2176	2392	0	1	tl	-	tl	2392	2	2869.10	2392	1745.39
zeroin.i.1	211	4100	0.2	1822	1822	3225	1	tl	-	tl	1822	8	74.60	1822	21.10
zeroin.i.2	211	3541	0.2	1004	1004	0	1	tl	-	tl	1008	192	tl	1004	95.87
zeroin.i.3	206	3540	0.2	998	998	3146	1	tl	-	tl	1002	152	tl	998	76.67

Table 3.6: Comparison between MIP_E and MIP_E^1 on a benchmark of DIMACS instances

modify the model MIP_E, now denoted MIP_E^2 as follows:

$$Z(MIP_E^2) = \min \sum_{v \in V} \sum_{i \in \{1,...,k\}} i \cdot z_v^i \tag{3.36}$$

$$\sum_{S \in \mathscr{S}} \xi_S^i \leq 1, \qquad i \in \{1,...,k\}, \tag{3.37}$$

$$\sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}: v \in S} \xi_S^i \geq 1, \qquad v \in V, \tag{3.38}$$

$$z_v^i = \sum_{S \in \mathscr{S}: v \in S} \xi_S^i, \qquad v \in V, i \in \{1,...,k\}, \tag{3.39}$$

$$\xi_S^i \in \{0,1\}, \qquad i \in \{1,...,k\}, S \in \mathscr{S}, \tag{3.40}$$

$$z_v^i \geq 0, \qquad v \in V, i \in \{1,...,k\}. \tag{3.41}$$

name	n	m	d	LP_C	LP_E	LP_E^1
2-Insertions_3	37	72	0.1	56	62	62
3-Insertions_3	56	110	0.1	84	92	92
anna	138	493	0.1	199	-	276
david	87	406	0.1	127	-	237
DSJC125.1	125	736	0.1	188	-	313
DSJC125.5	125	3891	0.5	188	978	978
DSJC125.9	125	6961	0.9	188	2500	2500
DSJC250.9	250	27897	0.9	375	8235	8235
DSJC500.9	500	224874	≈1.0	-	2189	-
DSJR500.1c	500	121275	≈1.0	750	16234	16234
flat300_20_0	300	21375	0.5	450	3150	-
games120	120	638	0.1	180	-	443
huck	74	301	0.1	111	243	243
jean	80	254	0.1	115	-	217
miles1000	128	3216	0.4	192	1666	1666
miles1500	128	5198	0.6	192	3354	3354
miles500	128	1170	0.1	192	705	-
miles750	128	2113	0.3	192	1173	1173
mug100_1	100	166	≈0.0	150	202	202
mug100_25	100	166	≈0.0	150	202	202
mug88_1	88	146	≈0.0	132	178	178
mug88_25	88	146	≈0.0	132	178	178
mulsol.i.1	197	3925	0.2	266	-	1957
mulsol.i.2	188	3885	0.2	275	-	1191
mulsol.i.3	184	3916	0.2	271	-	1187
mulsol.i.4	185	3946	0.2	273	-	1189
mulsol.i.5	186	3973	0.2	274	-	1160
myciel3	11	20	0.4	17	21	21
myciel4	23	71	0.3	35	44	44
myciel5	47	236	0.2	71	88	88
myciel6	95	755	0.2	143	176	176
myciel7	191	2360	0.1	287	349	349
queen10_10	100	1470	0.3	150	550	550
queen11_11	121	1980	0.3	182	726	726
queen12_12	144	2596	0.3	216	936	936
queen13_13	169	3328	0.2	254	1183	1183
queen5_5	25	160	0.5	38	75	75
queen6_6	36	290	0.5	54	138	138
queen7_7	49	476	0.4	74	196	196
queen8_12	96	1368	0.3	144	624	624
queen8_8	64	728	0.4	96	291	291
queen9_9	81	1056	0.3	122	405	405
school1_nsh	352	14612	0.2	528	-	2392
zeroin.i.1	211	4100	0.2	274	-	1822
zeroin.i.2	211	3541	0.2	290	-	1004
zeroin.i.3	206	3540	0.2	285	-	998

Table 3.7: Comparison between the values of the linear relaxations of MIP_C, MIP_E and MIP_E^1 on a benchmark of DIMACS instances

This model is similar to MIP_E except for constraints (3.39), (3.41) and the objective function that is now the same as MIP_C using instead variables z_v^i. Constraint (3.39) enforce that z_v^i represents the color i has been assigned to vertex v with the help of the corresponding variables ξ_S^i. Constraint (3.41) is the non negativity constraint for variables z. Integrality is not needed here since variables z are a linear combination of variables ξ which are integer thanks to constraints (3.40).

By replacing constraints (3.40) with the following ones:

$$\xi_S^i \geq 0 \qquad i \in \{1, ..., k\}, S \in \mathscr{S},$$

we obtain the Linear Programming relaxation of MIP_E^2, denoted as LP_E^2. Let us denote γ_v^i the dual variables associated to constraints (3.39). The dual problem of LP_E^2 reads as follows:

$$\max \sum_{v\in V} \mu_v + \sum_{i\in\{1,...,k\}} \pi_i \tag{3.42}$$

$$\sum_{v\in S}(\mu_v - \gamma_v^i) + \pi_i \le c_S^i, \qquad S\in\mathscr{S}, i\in\{1,...,k\}, \tag{3.43}$$

$$\gamma_v^i - \mu_v \le i, \qquad v\in V, i\in\{1,...,k\}, \tag{3.44}$$

$$\mu_v \ge 0, \qquad v\in V, \tag{3.45}$$

$$\pi_i \le 0, \qquad i\in\{1,...,k\}, \tag{3.46}$$

$$\gamma_v^i \lesseqgtr 0, \qquad v\in V, i\in\{1,...,k\}. \tag{3.47}$$

The reduced cost of a given variable ξ_S^i in MIP_E^2 is

$$\tilde{c_S^i} = c_S^i - \left(\sum_{v\in S}(\mu_v - \gamma_v^i) + \pi_i\right) = i\cdot|S| - \sum_{v\in S}(\mu_v + \gamma_v^i) - \pi_i,$$

and the problem that must be solved is

$$\max_{S\in\mathscr{S}}\left\{\sum_{v\in S}(\mu_v - \gamma_v^i) + \pi_i - i\cdot|S|\right\},$$

which is a MWSS where the weight of a vertex for a given color i is defined as $\mu_v - \gamma_v^i$.

We now replicate the branching rule used in MIP_C for MIP_E^2 thanks to variables z_v^i. Let us consider a fractional solution (ξ^*, z^*). If there exists a vertex v and a color i such that $0 < z_v^i < 1$, the following branching rule is applied:

- In the first child node, vertex v is assigned color i, thus we set $z_v^i = 1$. The pricing subproblem must be modified as well in order to respect this branching decision.

  For pricing subproblem i, vertex v and its neighborhood $N_G(v)$ are removed from the graph G. After a solution is produced by the algorithm, vertex v is added to this solution. It is possible since no vertices $w\in N_G(v)$ are part of the solution, creating no conflict and keeping the solution valid.

  For all other pricing subproblems, vertex v is removed from the graph G.

- In the second child node, vertex v cannot be assigned color i, thus we set $z_v^i = 0$. The pricing subproblem must be modified as well in order to respect this branching decision. For pricing subproblem i, vertex v is removed from the graph G.

If there exists no variable z_v^i such that $0 < z_v^i < 1$, the solution is trivially integer since all vertex have been assigned a color.

This branching rule can potentially lead to a smaller branching tree height than the derived Zykov branching rule presented in Section 3.3.3. The depth of a leaf in this branching tree is at worst $n * h$ against $n * n$ for the derived Zykov branching. Computation results showed that the derived Zykov branching rule is more efficient overall. The respective computation time of MIP_E with this branching rule is in the same order of magnitude than the Zykov branching rule, but for most of the instances, it is slower.

It is interesting to note that variables y_v^i from MIP_E^1 can potentially be used to replicate this branching rule. The branching decision where a vertex u is assigned a color i can be replicated by setting $y_v^j = 0$ for all $j \neq i$.

Valid inequalities for MIP_E

Proposition 6 can be translated into the following family of constraints for MIP_E:

$$\sum_{S \in \mathscr{S}} |S| \xi_S^i \geq \sum_{S \in \mathscr{S}} |S| \xi_S^{i+1}, \qquad i \in \{1, ..., k-1\}.$$

For all possible colors $i \in \{1, ..., k-1\}$ the respective size of the chosen stable set for color i must be greater or equal than the chosen stable set for color $i+1$. Preliminary results showed that the addition of this constraint did not improve the efficiency of MIP_E. The computation time associated to MIP_E injected with this class of constraints is always slower than the computation time of MIP_E and MIP_E^1.

Upper bound on s(G)

In this section, we present an upper bound on $s(G)$, denoted UB_h. This upper bound can be computed from any feasible solution for the MSCP. During the Branch-and-Price algorithm solving MIP_E, every time a new feasible solution is computed, a potentially new upper bound UB_h can be computed. UB_h is based on the following proposition:

Proposition 12 *Let $\Sigma^h(G)$ be the optimal value of the MSCP for graph G using exactly h colours.*

$$\Sigma^h(G) \geq LB_h = \frac{h(h+1)}{2} + n - h$$

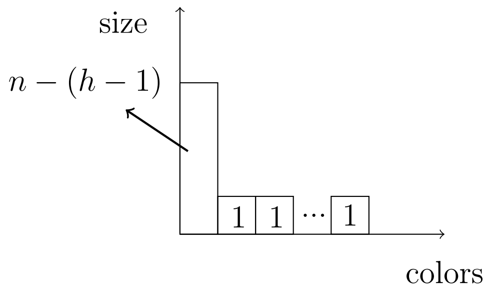

Figure 3.8: Best case for a given color h

Proof. Let h be a given number of colors. The best case we can hope for is the following: The largest number of possible vertices is assigned the first color, and the rest of vertices are assigned each remaining color that needs to be filled. This gives us the following value for the sum coloring:

$$\sum_{i=2}^{h} i + (n-(h+1)) = \frac{h(h+1)}{2} - 1 + n - h + 1 = \frac{h(h+1)}{2} + n - h$$

which is a lower bound for $\Sigma^h(G)$. □

LB_h is a lower bound on the optimal solution of the MSCP with exactly h colors. This means any solution of the MSCP using h colors is larger or equal than LB_h. Let v be the value of a given feasible solution for the MSCP. Thanks to LB_h, v can be bounded. Since there exists $j \in \{1,...,k\}$ such that $LB_j \leq \Sigma(G)$ and $\Sigma(G) \leq v$, there exists $l \in \{1,...,k\}$ such that $LB_l \leq v \leq LB_{l+1}$. This means we have bounded this feasible solution with the a bound on the optimal value of the MSCP with $l+1$ colors. This means either in order to have a solution with a better value than v, we must use at worst j colors. Thus we have $UB_h = l$.

This bound can be further improved by replacing n by $\alpha(G)$ in the formula. Further work on upper bounds on $s(G)$ is developed by Lecat et al. [32].

Boxstep method applied to MIP_E

As discussed in 3.3.3.4, the CG algorithm struggles to converge towards the optimum value. The boxstep method, introduced by du Merle et al. [11], is a method used to tackle this problem.

The aim of the boxstep method is to prevent the LP solution from oscillating too much. When the CG algorithm is tailing off, it can be due to the LP solution structure changing completely from one iteration to another. To prevent that, a box is created around the LP solution. If the LP solution is not inside the box, a penalty is payed in the objective function, with the hope of making this particular LP solution not optimal for this iteration of the CG algorithm, favoring a solution inside the box.

We define the boxstep for MIP_E around variables μ_v, $v \in V$ as the stability centre. Let $\delta_-, \delta_+ \geq 0$ be two parameters defining the size of the box boundaries and $\epsilon_-, \epsilon_+ \geq 0$ be two parameters defining the cost of constructing a solution outside the box. Let us denote by MIP_E^B the model MIP_E with a boxstep. To create this boxstep, parameters $\delta_-, \delta_+, \epsilon_-, \epsilon_+$ are injected into MIP_E^B and new variables y_+^v, $v \in V$ are added as follows:

$$Z(MIP_E^B) = \min \sum_{i\in\{1,...,k\}} \sum_{S\in\mathscr{S}} c_S^i \xi_S^i + \sum_{v\in V}(-\delta_- y_-^v + \delta_+ y_+^v) \tag{3.48}$$

$$\sum_{S\in\mathscr{S}} \xi_S^i \leq 1, \qquad i \in \{1, ..., k\}, \tag{3.49}$$

$$\sum_{i\in\{1,...,k\}} \sum_{S\in\mathscr{S}:v\in S} \xi_S^i - y_-^v + y_+^v \geq 1, \qquad v \in V, \tag{3.50}$$

$$0 \leq y_-^v \leq \epsilon_-, \qquad v \in V, \tag{3.51}$$

$$0 \leq y_+^v \leq \epsilon_+, \qquad v \in V, \tag{3.52}$$

$$\xi_S^i \in \{0, 1\}, \qquad i \in \{1, ..., k\}, S \in \mathscr{S}. \tag{3.53}$$

Constraints (3.51) and (3.52) create the boxstep as previously defined. Their meaning is directly linked to the dual of MIP_E^B. Let z_-^v,z_+^v be the variables respectively associated to constraints (3.51) and (3.52). The dual of MIP_E^B reads as follows:

$$\max \sum_{v\in V}\mu_v + \sum_{i\in\{1,...,k\}}\pi_i - \sum_{v\in V}(\epsilon_- z_-^v + \epsilon_+ z_+^v) \tag{3.54}$$

$$\sum_{v\in S}\mu_v + \pi_i \le c_S^i, \quad S\in\mathscr{S}, i\in\{1,...,k\}, \tag{3.55}$$

$$-\mu_v + z_-^v \le -\delta_-, \qquad v\in V, \tag{3.56}$$

$$\mu_v + z_-^v \le \delta_+, \qquad v\in V, \tag{3.57}$$

$$z_-^v \le 0, \qquad v\in V, \tag{3.58}$$

$$z_+^v \le 0, \qquad v\in V, \tag{3.59}$$

$$\mu_v \ge 0, \qquad v\in V, \tag{3.60}$$

$$\pi_i \le 0, \qquad i\in\{1,...,k\}. \tag{3.61}$$

The penalties linked to the boxstep are in the objective function. Constraints (3.56) and (3.56) compute the coordinates of the dual solution and activate the penalties accordingly. Constraints (3.56) and (3.56) respectively define the boundaries of variables z_-^v and z_+^v.

Preliminary tests showed that this adaptation of the boxstep method did not help improve the efficiency of MIP_E yet. This method holds many parameters and specific tuning of these parameters could be the key to improving MIP_E with a boxstep method.

Dual ascent applied to MIP_E

The dual ascent method [3, 2] is a parametric and Lagrangian relaxation that is known to be very efficient for set-partitioning problems. It provides high quality lower bounds to the problem with a relatively small computation time. Based on the work of Boschetti et al. [3] and Boschetti and Maniezzo [2] on set-partitioning and set-covering, we adapt the dual ascent method to MIP_E. Let us remind the formulation MIP_E:

$$Z(MIP_E) = \min \sum_{i\in\{1,...,k\}}\sum_{S\in\mathscr{S}} c_S^i \xi_S^i \tag{3.62}$$

$$\sum_{S\in\mathscr{S}} \xi_S^i \le 1, \qquad i\in\{1,...,k\}, \tag{3.63}$$

$$\sum_{i\in\{1,...,k\}}\sum_{S\in\mathscr{S}:v\in S} \xi_S^i \ge 1, \qquad v\in V, \tag{3.64}$$

$$\xi_S^i \in \{0,1\}, \qquad i\in\{1,...,k\}, S\in\mathscr{S}, \tag{3.65}$$

and its dual problem:

$$\max \sum_{v\in V} \mu_v + \sum_{i\in\{1,...,k\}} \pi_i \tag{3.66}$$

$$\sum_{v\in S} \mu_v + \pi_i \leq c_S^i, \qquad S \in \mathscr{S}, i \in \{1,...,k\}, \tag{3.67}$$

$$\mu_v \geq 0, \qquad v \in V, \tag{3.68}$$

$$\pi_i \leq 0, \qquad i \in \{1,...,k\}. \tag{3.69}$$

Let us introduce the following variables. For each variable ξ_S^i, $S \in \mathscr{S}$, $i \in \{1,...,k\}$ we associate $|S|+1$ new variables assigned to ξ_S^i:

$$y_{S,i}^v = \begin{cases} 1 \text{ if variable } \xi_S^i \text{ has been chosen to satisfy constraint } v, \\ 0 \text{ otherwise.} \end{cases}$$

$$y_{S,i}^i = \begin{cases} 1 \text{ if variable } \xi_S^i \text{ has been chosen to satisfy constraint } i, \\ 0 \text{ otherwise.} \end{cases}$$

This gives us this new definition for variables ξ_S^i:

$$\xi_S^i = \frac{1}{|S|+1}\left(\sum_{v\in S} y_{i,S,v} + y_{i,S,i}\right), \qquad i \in \{1,...,k\}, S \in \mathscr{S}.$$

Replacing these variables into MIP_E gives us a parametric relaxation of MIP_E denoted MIP_E^P. This is a relaxation of MIP_E since a variable can be chosen for a given constraint out of the $|S|+1$ constraints it has a positive coefficient in, but not for the other $|S|$ constraints which could violate the constraints of the original problem.

MIP_E^P reads as follows:

$$Z(MIP_E^P) = \min \sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}} c_S^i \frac{1}{|S|+1} \left(\sum_{v \in S} y_{i,S,v} + y_{i,S,i} \right) \tag{3.70}$$

$$\sum_{S \in \mathscr{S}} \frac{1}{|S|+1} \left(\sum_{v \in S} y_{i,S,v} + y_{i,S,i} \right) \leq 1, \quad i \in \{1,...,k\}, \tag{3.71}$$

$$\sum_{S \in \mathscr{S}} y_{i,S,i} \leq 1, \quad i \in \{1,...,k\}, \tag{3.72}$$

$$\sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}: v \in S} \frac{1}{|S|+1} \left(\sum_{v \in S} y_{i,S,v} + y_{i,S,i} \right) \geq 1, \quad v \in V, \tag{3.73}$$

$$\sum_{i \in \{1,...,k\}} \sum_{S \in \mathscr{S}: v \in S} y_{i,S,v} \geq 1, \quad v \in V, \tag{3.74}$$

$$y_{S,i}^v \in \{0,1\}, \quad i \in \{1,...,k\}, S \in \mathscr{S}, v \in S, \tag{3.75}$$

$$y_{S,i}^i \in \{0,1\}, \quad i \in \{1,...,k\}, S \in \mathscr{S}. \tag{3.76}$$

Constraints (3.72) and (3.74) are corresponding to the regular constraints (3.63) (3.64) with the new variables y. Following the dual ascent method, we now apply a Lagrangian relaxation on both constraints (3.71) and (3.73). Let λ be a non-negative penalty vector of size $n + k$, we now have the following parametric relaxation of MIP_E by relaxing constraints (3.71) and (3.73) into the objective function:

$$
\begin{aligned}
\min \sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}} c_S^i \frac{1}{|S|+1}\left(\sum_{v\in S} y_{i,S,v} + y_{i,S,i}\right) \\
+\sum_{i\in\{1,\ldots,k\}} \lambda_i \left(\sum_{S\in\mathscr{S}} \frac{1}{|S|+1}\left(\sum_{v\in S} y_{i,S,v} + y_{i,S,i}\right) - 1\right) \\
+\sum_{v\in V} \lambda_v \left(1 - \sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} \frac{1}{|S|+1}\left(\sum_{v\in S} y_{i,S,v} + y_{i,S,i}\right)\right) \\
\sum_{S\in\mathscr{S}} y_{i,S,i} \le 1 \qquad & i\in\{1,\ldots,k\}, \\
\sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} y_{i,S,v} \ge 1, \qquad & v\in V, \\
y_{i,S,i}\in\{0,1\}, \qquad & i\in\{1,\ldots,k\}, S\in\mathscr{S}, \\
y_{i,S,v}\in\{0,1\}, & i\in\{1,\ldots,k\}, S\in\mathscr{S}, v\in S.
\end{aligned}
$$

This can be simplified as follows:

$$
\begin{aligned}
\min \sum_{i\in\{1,\ldots,k\}} \left(\left(\sum_{s\in\mathscr{S}} \bar{c_S^i} y_{i,S,i}\right) - \lambda_i\right) \\
+\sum_{v\in V} \left(\left(\sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} \bar{c_S^i} y_{i,S,v}\right) + \lambda_v\right) \\
\sum_{S\in\mathscr{S}} y_{i,S,i} \le 1 \qquad & i\in\{1,\ldots,k\}, \\
\sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} y_{i,S,v} \ge 1, \qquad & v\in V, \\
y_{i,S,i}\in\{0,1\}, \qquad & i\in\{1,\ldots,k\}, S\in\mathscr{S}, \\
y_{i,S,v}\in\{0,1\}, & i\in\{1,\ldots,k\}, S\in\mathscr{S}, v\in S,
\end{aligned}
$$

with $\bar{c_S^i} = \frac{1}{|S|+1}\left(c_S^i - \sum_{w\in S} \lambda_w + \lambda_i\right)$. The problem is decomposable in $k+n$ subproblems. For each $v\in V$, we have to solve:

$$\min \sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} \bar{c_S^i} y_{i,S,v} + \lambda_v \tag{3.77}$$

$$\sum_{i\in\{1,\ldots,k\}} \sum_{S\in\mathscr{S}:v\in S} y_{i,S,v} \ge 1, \tag{3.78}$$

$$y_{i,S,v} \le 1, \qquad i\in\{1,\ldots,k\}, S\in\mathscr{S}, v\in S. \tag{3.79}$$

Let us denote μ'_v the dual variables associated to constraints (3.79) and $\rho_{i,S,v}$ the dual variables associated to constraints (3.79). The associated dual problem of (3.77)-(3.79) is the following :

$$\max\ \mu'_v + \sum_{i\in\{1,...,k\}} \sum_{S\in\mathscr{S}:v\in S} \rho_{i,S,v} \tag{3.80}$$

$$\mu'_v + \rho_{i,S,v} \le \bar{c}^i_S, \qquad i \in \{1,...,k\}, S \in \mathscr{S}, v \in S, \tag{3.81}$$

$$\mu'_v \ge 0, \tag{3.82}$$

$$\rho_{i,S,v} \le 0 \qquad i \in \{1,...,k\}, S \in \mathscr{S}, v \in S. \tag{3.83}$$

For each $i \in \{1, ..., k\}$, we have to solve:

$$\min \sum_{s\in\mathscr{S}} \bar{c}^i_S y_{i,S,i} - \lambda_i \tag{3.84}$$

$$\sum_{S\in\mathscr{S}} y_{i,S,i} \le 1, \tag{3.85}$$

$$y_{i,S,i} \in \{0, 1\}, \qquad S \in \mathscr{S}. \tag{3.86}$$

A feasible dual solution (μ, π) to (3.66)-(3.69) can be constructed with the optimal solutions of the $n + k$ subproblems. This feasible dual solution is described by the following proposition.

Proposition 13 *Let μ and π respectively be vector of size n and k. A feasible solution to (3.66)-(3.69) can be defined as follows. For $v \in V$,*

$$\mu_v = \begin{cases} \lambda_v & \text{if there exists a variable such that } \bar{c}^i_S < 0, \\ \min\limits_{i\in\{1,...,k\},S\in\mathscr{S}:v\in S} \{\bar{c}^i_S\} + \lambda_v & \text{otherwise.} \end{cases}$$

We also define dual variables $\rho_{i,S,v}$ of constraints (3.79):

$$\rho_{i,S,v} = \begin{cases} \bar{c}^i_S & \text{if } \bar{c}^i_S < 0, \\ 0 & \text{otherwise.} \end{cases}$$

For $i \in \{1, ..., k\}$:

$$\pi_i = \begin{cases} -\lambda_i + \min\limits_{S\in\mathscr{S}} \left\{ \sum\limits_{v\in S} \rho_{i,S,v} \right\} & \text{if } \forall S \in \mathscr{S}, \bar{c}^i_S > 0, \\ \min\limits_{S\in\mathscr{S}}\{\bar{c}^i_S\} - \lambda_i + \min\limits_{S\in\mathscr{S}} \left\{ \sum\limits_{v\in S} \rho_{i,S,v} \right\} & \text{otherwise.} \end{cases}$$

Proof. Let us consider dual constraint (3.67) for column λ_S^i. The following inequality holds:

$$\min_{j\in\{1,...,k\},T\in\mathscr{S}:v\in S}\{\bar{c_T^j}\} \leq \bar{c_S^i}$$

From the definition of μ_v, we obtain:

$$\mu_v \leq \bar{c_S^i}, \qquad v \in S, S \in \mathscr{S}, i \in \{1,...,k\}$$

and by adding those inequalities, we derive

$$\sum_{v\in S}\mu_v \leq \sum_{v\in S}\bar{c_S^i}, \qquad S \in \mathscr{S}, i \in \{1,...,k\}.$$

If no variable is such that $\bar{c_S^i} < 0$,

$$\sum_{v\in S}\mu_v + \pi_i \leq \sum_{v\in S}\left(\min_{j\in\{1,...,k\},T\in\mathscr{S}:v\in T}\{\bar{c_T^j}\} + \lambda_v\right) + \min_{U\in\mathscr{S}}\{\bar{c_U^i}\} - \lambda_i + \min_{W\in\mathscr{S}}\left\{\sum_{w\in W}\rho_{i,W,w}\right\}$$

According to the definition of $\rho_{i,W,w}$, we have

$$\sum_{v\in S}\left(\min_{j\in\{1,...,k\},T\in\mathscr{S}:v\in T}\{\bar{c_T^j}\} + \lambda_v\right) + \min_{W\in\mathscr{S}}\left\{\sum_{w\in W}\rho_{i,W,w}\right\} \leq \sum_{v\in S}(\bar{c_S^i} + \lambda_v).$$

Thus,

$$\begin{aligned}
&\sum_{v\in S}\mu_v + \pi_i \leq \sum_{v\in S}(\bar{c_S^i} + \lambda_v) + \bar{c_S^i} - \lambda_i \\
&\sum_{v\in S}\mu_v + \pi_i \leq \sum_{j=1}^{|S|}\bar{c_S^i} + \bar{c_S^i} + \sum_{v\in S}\lambda_v - \lambda_i \\
&\sum_{v\in S}\mu_v + \pi_i \leq \sum_{j=1}^{|S|+1}\bar{c_S^i} + \sum_{v\in S}\lambda_v - \lambda_i \\
&\sum_{v\in S}\mu_v + \pi_i \leq \sum_{j=1}^{|S|+1}\frac{1}{|S|+1}\left(c_S^i - \sum_{w\in S}\lambda_w + \lambda_i\right) + \sum_{v\in S}\lambda_v - \lambda_i \\
&\sum_{v\in S}\mu_v + \pi_i \leq c_S^i - \sum_{w\in S}\lambda_w + \lambda_i + \sum_{v\in S}\lambda_v - \lambda_i \\
&\sum_{v\in S}\mu_v + \pi_i \leq c_S^i
\end{aligned}$$

The same method can be applied to the case where there exists a variable such that $\bar{c_S^i} < 0$. No constraint is violated, (μ, π) is a feasible solution for the dual of MIP_E. □

To solve this formulation, a subgradient algorithm can be used to optimize the lagrangian penalty vector. This algorithm requires a lot of tuning, preliminary computational tests showed us that this method could not reduce the computing time of computing LB_E while providing a quality lower bound. With our current configuration, the computation time of the dual ascent algorithm is larger than the computation time of LP_E while computing a smaller lower bound than LP_E.

Conclusion

In this chapter, we have studied the MSCP and proposed three ILP models to solve it. We compared those formulations computationally. MIP_E is the most efficient model for a random benchmark of instances and shows the most potential for the DIMACS benchmark of instances. We improved this model by proposing a slightly modified version of this model allowing us to create reduce the number of variables in the model.

This new model MIP_E has not shown all of its potential. The main drawback is that the associated CG algorithm is hard to compute. We have tried several methods to tackle this problem, unsuccessfully. An interesting stream of research would be to use tight upper bounds of $s(G)$ to initialize our algorithm. This would lead to a large decrease in terms of variables. Another weakness of the model is the heuristic procedure finding new feasible solutions for the MSCP. These feasible solutions were not good enough, a more refined procedure could improve the efficiency of MIP_E and take more advantage of the strong value of $z(LP_E)$. One of the main drawbacks of MIP_E is the slow convergence of the column generation algorithm. Tackling this issue is crucial if we want to improve MIP_E, a stream of research would be to do a complete review of stabilization procedures.

This work has been published in [18].

Appendix

Computational results for DIMACS instances

				DSATUR					Branch-and-Price algorithm						
Instance				[51]		Reimplementation [16]			MMT-BP [36]			Exactcolors [23]			
instance	n	m	d	UB	time	UB	time	nodes	LB	UB	time	LB	UB	time	nodes
1-FullIns_4	93	593	0.1	5	0.2	5	0.0	3,013	4	5	tl	4	5	tl	39,851
1-FullIns_5	282	3247	0.1	6	tl	6	0.0	4,611	4	6	tl	4	6	tl	2,893
1-Insertions_4	67	232	0.1	5	240.0	5	tl	3,285,368,292	3	5	tl	4	5	tl	48,673
1-Insertions_5	202	1227	0.1	6	tl	6	tl	1,941,304,976	3	6	tl	4	6	tl	535
1-Insertions_6	607	6337	≈0.0	7	tl	7	tl	984,504,060	3	7	tl	-	-	tl	-
2-FullIns_3	52	201	0.2	5	0.0	5	0.0	6	5	5	2.9	5	5	0.0	1
2-FullIns_4	212	1621	0.1	6	tl	6	0.0	69	5	6	tl	5	6	tl	3,057
2-FullIns_5	852	12201	≈0.0	7	tl	7	0.0	40,482	5	7	tl	5	7	tl	289
2-Insertions_4	149	541	≈0.0	5	tl	5	tl	2,282,050,125	3	5	tl	3	5	tl	2,149
2-Insertions_5	597	3936	≈0.0	6	tl	6	tl	1,028,588,270	3	6	tl	3	6	tl	25
3-FullIns_3	80	346	0.1	6	tl	6	0.0	7	6	6	2.9	6	6	0.0	1
3-FullIns_4	405	3524	≈0.0	7	tl	7	0.0	129	6	7	tl	6	7	tl	989
3-FullIns_5	2030	33751	≈0.0	8	tl	8	tl	665,789,658	6	8	tl	6	8	tl	115
3-Insertions_3	56	110	0.1	4	0.6	4	1.5	3,434,261	3	4	tl	3	4	tl	85,789
3-Insertions_4	281	1046	≈0.0	5	tl	5	tl	1,608,405,161	3	5	tl	3	5	tl	211
3-Insertions_5	1406	9695	≈0.0	6	tl	6	tl	497,689,431	2	6	tl	-	-	tl	-
4-FullIns_3	114	541	0.1	7	tl	7	0.0	8	7	7	3.4	7	7	0.0	1
4-FullIns_4	690	6650	≈0.0	8	tl	8	0.0	128	7	8	tl	7	8	tl	1,137
4-FullIns_5	4146	77305	≈0.0	9	tl	9	tl	423,272,354	7	9	tl	7	9	tl	33
4-Insertions_3	79	156	0.1	4	351.0	4	tl	3,229,375,685	3	4	tl	3	4	tl	32,149
4-Insertions_4	475	1795	≈0.0	5	tl	5	tl	1,250,788,090	3	5	tl	3	5	tl	31
5-FullIns_3	154	792	0.1	8	tl	8	0.0	9	8	8	4.6	8	8	0.0	1
5-FullIns_4	1085	11395	≈0.0	9	tl	9	0.0	153	8	9	tl	8	9	tl	737
abb313GPIA	1557	53356	≈0.0	10	tl	10	tl	785,420,931	8	9	tl	-	-	tl	-
anna	138	493	0.1	11	0.0	11	0.0	1	11	11	3.6	11	11	0.0	1
ash331GPIA	662	4181	≈0.0	4	0.0	4	0.0	1,131	4	4	45.9	4	6	tl	45
ash608GPIA	1216	7844	≈0.0	4	0.1	4	0.0	2,427	4	4	tl	-	-	tl	-
ash958GPIA	1916	12506	≈0.0	4	0.4	5	tl	234,359,946	3	4	tl	-	-	tl	-
C2000.5	2000	999836	0.5	-	-	216	tl	1	-	-	-	-	-	tl	-
C4000.5	4000	4000268	0.5	-	-	-	tl	-	-	-	-	-	-	tl	-
david	87	406	0.1	11	0.0	11	0.0	1	11	11	0.2	11	11	0.0	1
DSJC1000.1	1000	49629	0.1	-	-	25	tl	311,540,866	5	20	tl	-	-	tl	-
DSJC1000.5	1000	249826	0.5	-	-	114	tl	144,353,376	13	84	tl	-	-	tl	-
DSJC1000.9	1000	449449	0.9	-	-	-	tl	-	51	224	tl	-	-	tl	-
DSJC125.1	125	736	0.1	5	0.0	5	0.0	290	5	5	42.0	5	6	tl	2,097
DSJC125.5	125	3891	0.5	19	tl	19	tl	1,007,429,681	17	17	tl	17	23	tl	4,487
DSJC125.9	125	6961	0.9	46	tl	45	tl	701,674,554	44	44	tl	44	50	tl	32,661
DSJC250.1	250	3218	0.1	9	tl	9	tl	1,017,076,072	6	8	tl	-	-	tl	2,152
DSJC250.5	250	15668	0.5	34	tl	35	tl	569,582,279	20	28	tl	26	37	tl	535
DSJC250.9	250	27897	0.9	82	tl	-	tl	-	71	72	tl	71	90	tl	2,647
DSJC500.1	500	12458	0.1	14	tl	14	tl	603,420,521	5	12	tl	-	-	tl	-
DSJC500.5	500	62624	0.5	62	tl	62	tl	321,102,532	16	48	tl	-	-	tl	-
DSJR500.1c	500	121275	≈1.0	85	tl	-	tl	-	85	85	288.5	85	88	tl	1,421
DSJR500.1	500	3555	≈0.0	12	0.0	12	0.1	117,132	12	12	35.3	12	14	tl	2,321
DSJR500.5	500	58862	0.5	130	tl	134	tl	1,026,643,225	122	122	342.2	122	132	tl	139
flat1000_50_0	1000	245000	0.5	-	-	112	tl	161,099,104	-	-	-	-	-	tl	tl
flat1000_60_0	1000	245830	0.5	-	-	112	tl	142,816,890	-	-	-	-	-	tl	tl
flat1000_76_0	1000	246708	0.5	-	-	111	tl	125,707,629	-	-	-	-	-	tl	tl
fpsol2.i.1	496	11654	0.1	65	0.0	65	0.0	1	65	65	10.6	65	65	0.5	1
fpsol2.i.2	451	8691	0.1	30	0.0	30	0.0	1	30	30	11.2	30	30	0.3	1
fpsol2.i.3	425	8688	0.1	30	0.0	30	0.0	1	30	30	10.0	30	30	0.4	1
games120	120	638	0.1	9	0.0	9	0.0	1	9	9	0.2	9	9	0.0	1
huck	74	301	0.1	11	0.0	11	0.0	1	11	11	0.2	11	11	0.0	1
inithx.i.1	864	18707	0.1	54	0.0	54	0.0	1	54	54	21.0	54	54	1.6	1
inithx.i.2	645	13979	0.1	31	0.0	31	0.0	1	31	31	9.2	31	31	1.4	1
inithx.i.3	621	13969	0.1	31	0.0	31	0.0	1	31	31	9.9	31	31	0.5	1
jean	80	254	0.1	10	0.0	10	0.0	1	10	10	0.2	10	10	0.0	1
latin_square	900	307350	0.8	130	tl	153	tl	1	90	102	tl	90	129	tl	53

Table 3.8: Exact VCP algorithms on DIMACS instances

				DSATUR					Branch-and-Price algorithm						
Instance				[51]		Reimplementation [16]			MMT-BP [36]			Exactcolors [23]			
instance	n	m	d	UB	time	UB	time	nodes	LB	UB	time	LB	UB	time	nodes
le450_15a	450	8168	0.1	16	tl	17	tl	2,067,792,547	15	15	0.4	15	17	tl	229
le450_15b	450	8169	0.1	16	tl	16	tl	972,319,400	15	15	0.2	15	17	tl	241
le450_15c	450	16680	0.2	22	tl	22	tl	639,434,972	15	15	3.1	-	-	tl	tl
le450_15d	450	16750	0.2	23	tl	23	tl	635,710,737	15	15	3.8	-	-	tl	tl
le450_25a	450	8260	0.1	25	0.0	25	0.0	1	25	25	0.1	25	25	2.6	1
le450_25b	450	8263	0.1	25	0.0	25	0.0	1	25	25	0.1	25	25	2.8	1
le450_25c	450	17343	0.2	-	-	27	tl	677,939,432	25	25	1356.6	25	28	tl	187
le450_5a	450	5714	0.1	-	-	9	tl	727,964,669	5	5	0.3	-	-	tl	tl
le450_5b	450	5734	0.1	-	-	9	tl	747,642,318	5	5	0.2	-	-	tl	tl
le450_5c	450	9803	0.1	5	0.0	5	0.0	2,892	5	5	0.1	-	-	tl	tl
le450_5d	450	9757	0.1	5	98.1	5	51.6	20,397,241	5	5	0.2	-	-	tl	tl
miles1000	128	3216	0.4	42	0.0	42	0.0	45	42	42	0.2	42	42	0.1	1
miles1500	128	5198	0.6	73	0.0	73	0.0	1	7	73	0.1	73	73	0.2	1
miles250	128	387	≈0.0	8	0.0	8	0.0	1	8	8	5.0	8	8	0.0	1
miles500	128	1170	0.1	20	0.0	20	0.0	1	20	20	3.7	20	20	0.0	1
miles750	128	2113	0.3	31	0.0	31	0.0	44	31	31	0.2	31	31	0.1	1
mug100_1	100	166	≈0.0	4	0.0	4	tl	5,348,610,817	4	4	14.4	4	4	1.5	1
mug100_25	100	166	≈0.0	4	0.0	4	tl	5,230,617,855	4	4	12.0	4	4	1.4	1
mug88_1	88	146	≈0.0	4	324.0	4	460.7	1,694,387,669	4	4	9.6	-	-	tl	tl
mug88_25	88	146	≈0.0	4	191.0	4	290.6	1,062,827,981	4	4	10.6	-	-	tl	tl
mulsol.i.1	197	3925	0.2	49	0.0	49	0.0	1	49	49	0.2	49	49	0.1	1
mulsol.i.2	188	3885	0.2	31	0.0	31	0.0	1	31	31	4.7	31	31	0.1	1
mulsol.i.3	184	3916	0.2	31	0.0	31	0.0	1	31	31	0.2	31	31	0.0	1
mulsol.i.4	185	3946	0.2	31	0.0	31	0.0	1	31	31	0.2	31	31	0.0	1
mulsol.i.5	186	3973	0.2	31	0.0	31	0.0	1	31	31	6.0	31	31	0.0	1
myciel6	95	755	0.2	7	tl	7	tl	2,106,152,516	4	7	tl	5	7	tl	24,537
myciel7	191	2360	0.1	8	tl	8	tl	1,573,252,182	5	8	tl	5	8	tl	4,371
qg.order30	900	26100	0.1	30	0.0	30	0.0	2,491	30	30	0.2	-	-	tl	tl
qg.order40	1600	62400	≈0.0	40	0.2	40	0.2	58,971	40	40	2.9	-	-	tl	tl
qg.order60	3600	212400	≈0.0	61	tl	62	tl	377,150,626	60	60	3.8	-	-	tl	tl
queen10_10	100	1470	0.3	12	466.0	11	tl	1,676,596,671	11	11	686.9	11	14	tl	3,777
queen11_11	121	1980	0.3	13	tl	13	tl	1,538,642,053	11	11	tl	11	14	tl	1,675
queen12_12	144	2596	0.3	14	tl	14	tl	1,430,010,102	12	13	tl	12	16	tl	573
queen13_13	169	3328	0.2	15	tl	15	tl	1,340,956,423	13	14	tl	13	17	tl	335
queen14_14	196	4186	0.2	16	tl	16	tl	1,187,949,544	14	15	tl	14	19	tl	91
queen15_15	225	5180	0.2	18	tl	18	tl	1,167,846,294	15	16	tl	-	-	tl	tl
queen16_16	256	6320	0.2	19	tl	19	tl	1,047,030,161	16	17	tl	16	21	tl	115
queen8_12	96	1368	0.3	12	0.0	12	0.0	375	12	12	0.2	12	12	11.3	261
queen8_8	64	728	0.4	9	0.0	9	0.5	668,699	9	9	3.6	9	13	tl	30,057
queen9_9	81	1056	0.3	10	3.0	10	244.5	300,489,064	10	10	36.6	10	12	tl	12,653
r1000.1c	1000	485090	≈1.0	-	-	-	-	-	-	-	-	-	-	-	-
school1	385	19095	0.3	14	0.1	14	4.5	383	14	14	0.4	14	14	1290.1	777
school1_nsh	352	14612	0.2	14	0.4	14	1.7	1,001	14	14	17.0	14	14	1060.9	697
wap01a	2368	110871	≈0.0	46	tl	51	tl	327,214,469	40	43	tl	-	-	tl	tl
wap02a	2464	111742	≈0.0	46	tl	45	tl	507,088,624	40	42	tl	-	-	tl	tl
wap03a	4730	286722	≈0.0	-	-	53	tl	353,433,910	40	47	tl	-	-	tl	tl
wap04a	5231	294902	≈0.0	-	-	48	tl	215,930,024	40	44	tl	-	-	tl	tl
wap05a	905	43081	0.1	50	0.0	50	0.0	1	50	50	293.2	50	50	6.0	1
wap06a	947	43571	0.1	47	tl	45	tl	565,243,140	40	40	175.0	-	-	tl	tl
wap07a	1809	103368	0.1	44	tl	45	tl	341,848,594	40	42	tl	-	-	tl	tl
wap08a	1870	104176	0.1	45	tl	46	tl	365,009,521	40	42	tl	-	-	tl	tl
will199GPIA	701	6772	0	7	tl	7	0.0	693	7	7	80.7	7	7	3.8	1
zeroin.i.1	211	4100	0.2	49	0.0	49	0.0	1	49	49	3.8	49	49	0.1	1
zeroin.i.2	211	3541	0.2	30	0.0	30	0.0	1	30	30	4.4	30	30	0.0	1
zeroin.i.3	206	3540	0.2	30	0.0	30	0.0	1	30	30	4.5	30	30	0.0	1

Table 3.9: Exact VCP algorithms on DIMACS instances

As discussed in Section 2.2, Table 3.8 and 3.9 report the results of DSATUR [51], our reimplementation of DSATUR [16], MMT-BP [36] and Exactcolors [23] for DIMACS instances. Following the conclusions of [51], for hard DIMACS instances, the Branch-and-Price algorithms remains the best approach overall. However, DSATUR is competitive for a subset of instances.

Instance				Value							Time						
name	n	m	d	χ_{G_A}	χ_{G_A}	ϑ^*	ω	χ_α	χ^*_H	ω^h	χ_{G_A}	χ_{G_A}	ϑ^*	ω	χ_α	χ^*_H	ω^h
1-FullIns_4	93	593	0.1	4	4	4	3	2	2	3	0.0	21.5	27.7	0.0	0.0	0.0	0.0
1-FullIns_5	282	3247	0.1	4	-	-	3	2	2	3	0.2	tl	tl	0.0	0.0	0.0	0.0
1-Insertions_4	67	232	0.1	3	3	3	2	2	2	2	0.1	4.9	4.3	0.0	0.0	0.0	0.0
1-Insertions_5	202	1227	0.1	3	-	-	2	2	2	2	6.0	tl	tl	0.0	0.0	0.0	0.0
1-Insertions_6	607	6337	≈0.0	4	-	-	2	2	2	2	1427.6	tl	tl	0.0	0.1	0.1	0.0
2-FullIns_3	52	201	0.2	5	4	5	4	2	2	4	0.0	0.5	1.0	0.0	0.0	0.0	0.0
2-FullIns_4	212	1621	0.1	5	-	-	4	2	2	4	0.1	tl	tl	0.0	0.0	0.0	0.0
2-FullIns_5	852	12201	≈0.0	5	-	-	4	2	2	4	1.9	tl	tl	0.0	0.0	0.2	0.0
2-Insertions_4	149	541	≈0.0	3	3	3	2	2	2	2	2.1	520.3	498.9	0.0	0.0	0.0	0.0
2-Insertions_5	597	3936	≈0.0	3	-	-	2	2	2	2	381.6	tl	tl	0.0	0.0	0.1	0.0
3-FullIns_3	80	346	0.1	6	5	6	5	2	2	5	0.0	2.9	11.8	0.0	0.0	0.0	0.0
3-FullIns_4	405	3524	≈0.0	6	-	-	5	2	2	5	0.1	tl	tl	0.0	0.5	0.0	0.0
3-FullIns_5	2030	33751	≈0.0	6	-	-	5	-	2	5	7.0	tl	tl	0.0	tl	2.3	0.0
3-Insertions_3	56	110	0.1	3	3	3	2	2	2	2	0.1	1.8	1.7	0.0	0.0	0.0	0.0
3-Insertions_4	281	1046	≈0.0	3	-	-	2	2	2	2	25.3	tl	tl	0.0	0.0	0.0	0.0
3-Insertions_5	1406	9695	≈0.0	-	-	-	2	2	2	2	tl	tl	tl	0.0	4.9	0.7	0.0
4-FullIns_3	114	541	0.1	7	6	7	6	2	2	6	0.0	15.8	97.2	0.0	0.0	0.0	0.0
4-FullIns_4	690	6650	≈0.0	7	-	-	6	2	2	6	0.2	tl	tl	0.0	0.3	0.1	0.0
4-FullIns_5	4146	77305	≈0.0	7	-	-	6	-	2	6	17.5	tl	tl	0.1	tl	21.1	0.0
4-Insertions_3	79	156	0.1	3	3	3	2	2	2	2	0.2	5.4	11.6	0.0	0.0	0.0	0.0
4-Insertions_4	475	1795	≈0.0	3	-	-	2	2	2	2	193.7	tl	tl	0.0	0.0	0.0	0.0
5-FullIns_3	154	792	0.1	8	7	8	7	2	2	7	0.0	45.6	597.9	0.0	0.0	0.0	0.0
5-FullIns_4	1085	11395	≈0.0	8	-	-	7	-	2	7	0.5	tl	tl	0.0	tl	0.3	0.0
abb313GPIA	1557	53356	≈0.0	-	-	-	8	4	3	2	tl	tl	tl	0.0	0.1	1.0	0.0
anna	138	493	0.1	11	11	11	11	1	3	6	0.0	4.7	303.0	0.0	0.0	0.0	0.0
ash331GPIA	662	4181	≈0.0	4	-	-	3	-	2	2	6.0	tl	tl	0.0	tl	0.1	0.0
ash608GPIA	1216	7844	≈0.0	-	-	-	3	-	2	2	tl	tl	tl	0.0	tl	0.4	0.0
ash958GPIA	1916	12506	≈0.0	-	-	-	3	-	2	2	tl	tl	tl	0.0	tl	1.9	0.0
C2000.5	2000	999836	0.5	-	-	-	-	-	23	2	tl	tl	tl	tl	tl	2.4	0.0
C4000.5	4000	4000268	0.5	-	-	-	-	-	32	2	tl	tl	tl	tl	tl	19.8	0.0
david	87	406	0.1	11	11	11	11	2	3	7	0.0	0.7	18.4	0.0	0.0	0.0	0.0
DSJC1000.1	1000	49629	0.1	-	-	-	6	-	6	2	tl	tl	tl	0.0	tl	0.3	0.0
DSJC1000.5	1000	249826	0.5	-	-	-	15	66	17	2	tl	tl	tl	497.6	609.9	0.3	0.0
DSJC1000.9	1000	449449	0.9	-	-	-	-	166	48	4	tl	tl	tl	tl	0.0	0.3	0.0
DSJC125.1	125	736	0.1	5	-	5	4	3	3	2	1.7	tl	219.1	0.0	10.9	0.0	0.0
DSJC125.5	125	3891	0.5	16	14	12	10	12	6	2	0.6	108.2	27.8	0.0	0.0	0.0	0.0
DSJC125.9	125	6961	0.9	43	43	38	34	31	16	10	0.2	0.2	0.6	10.2	0.0	0.0	0.0
DSJC250.1	250	3218	0.1	7	-	-	4	-	3	2	451.2	tl	tl	0.0	tl	0.0	0.0
DSJC250.5	250	15668	0.5	26	-	17	12	20	8	2	10.0	tl	1546.8	0.0	0.0	0.0	0.0
DSJC250.9	250	27897	0.9	71	71	56	-	50	23	7	3.6	5.6	21.7	tl	0.0	0.0	0.0
DSJC500.1	500	12458	0.1	-	-	-	5	-	4	2	tl	tl	tl	0.0	tl	0.0	0.0
DSJC500.5	500	62624	0.5	43	-	-	13	38	12	3	285.5	tl	tl	4.7	4.6	0.0	0.0
DSJR500.1c	500	121275	≈1.0	85	85	84	12	38	25	2	10.3	2.0	50.5	0.0	0.0	0.1	0.0
DSJR500.1	500	3555	0.0	12	-	-	-	-	4	2	0.8	tl	tl	tl	tl	0.0	0.0
DSJR500.5	500	58862	0.5	122	-	-	122	71	8	72	39.5	tl	tl	1022.8	0.0	0.0	0.0
flat1000_50_0	1000	245000	0.5	-	-	-	15	50	16	2	tl	tl	tl	290.7	20.6	0.3	0.0
flat1000_60_0	1000	245830	0.5	-	-	-	15	58	16	2	tl	tl	tl	339.9	119.8	0.3	0.0
flat1000_76_0	1000	246708	0.5	-	-	-	15	66	16	2	tl	tl	tl	703.3	559.0	0.3	0.0
fpsol2.i.1	496	11654	0.1	65	-	-	65	1	3	2	0.4	tl	tl	0.0	0.0	0.0	0.0
fpsol2.i.2	451	8691	0.1	30	-	-	30	1	2	2	0.3	tl	tl	0.0	0.0	0.0	0.0
fpsol2.i.3	425	8688	0.1	30	-	-	30	1	2	2	0.4	tl	tl	0.0	0.0	0.0	0.0
games120	120	638	0.1	9	9	9	9	5	3	2	0.0	5.0	120.1	0.0	0.0	0.0	0.0
huck	74	301	0.1	11	11	11	11	2	3	4	0.0	0.3	7.3	0.0	0.0	0.0	0.0
inithx.i.1	864	18707	0.1	54	-	-	54	1	2	2	1.5	tl	tl	0.0	0.1	0.1	0.0
inithx.i.2	645	13979	0.1	31	-	-	31	1	2	2	1.3	tl	tl	0.0	0.0	0.1	0.0
inithx.i.3	621	13969	0.1	31	-	-	31	1	2	2	0.4	tl	tl	0.0	0.0	0.1	0.0
jean	80	254	0.1	10	10	-	10	2	3	3	0.0	0.5	tl	0.0	0.0	0.0	0.0
latin_square	900	307350	0.8	90	-	10	-	90	17	90	48.7	tl	12.7	tl	0.0	0.2	0.0

Table 3.10: Comparison between Lower Bounds on $\chi(G)$ for DIMACS instances

Instance				Value							Time						
name	n	m	d	χ_{G_A}	χ_{G_A}	ϑ^*	ω	χ_α	χ_H^*	ω^h	χ_{G_A}	χ_{G_A}	ϑ^*	ω	χ_α	χ_H^*	ω^h
le450_15a	450	8168	0.1	15	-	-	15	-	5	2	41.8	tl	tl	0.0	tl	0.0	0.0
le450_15b	450	8169	0.1	15	-	-	15	-	6	3	38.3	tl	tl	0.0	tl	0.0	0.0
le450_15c	450	16680	0.2	15	-	-	15	-	7	4	621.2	tl	tl	0.0	tl	0.0	0.0
le450_15d	450	16750	0.2	15	-	-	15	-	6	2	417.8	tl	tl	0.0	tl	0.0	0.0
le450_25a	450	8260	0.1	25	-	-	25	-	8	3	2.4	tl	tl	0.0	tl	0.0	0.0
le450_25b	450	8263	0.1	25	-	-	25	-	8	2	2.5	tl	tl	0.0	tl	0.0	0.0
le450_25c	450	17343	0.2	25	-	-	25	-	9	2	42.2	tl	tl	0.0	tl	0.0	0.0
le450_5a	450	5714	0.1	5	-	-	5	-	3	2	1415.6	tl	tl	0.0	tl	0.0	0.0
le450_5b	450	5734	0.1	-	-	-	5	-	3	2	tl	tl	tl	0.0	tl	0.0	0.0
le450_5c	450	9803	0.1	5	-	-	5	-	3	2	562.0	tl	tl	0.0	tl	0.0	0.0
le450_5d	450	9757	0.1	5	-	-	5	-	3	2	737.6	tl	tl	0.0	tl	0.0	0.0
miles1000	128	3216	0.4	42	42	42	42	16	7	29	0.1	1.0	51.2	0.0	0.0	0.0	0.0
miles1500	128	5198	0.6	73	71	73	73	25	8	51	0.1	0.4	13.7	0.0	0.0	0.0	0.0
miles250	128	387	≈0.0	8	8	8	8	2	4	3	0.0	3.7	199.8	0.0	0.0	0.0	0.0
miles500	128	1170	0.1	20	19	20	20	7	5	12	0.0	6.6	141.1	0.0	0.0	0.0	0.0
miles750	128	2113	0.3	31	31	31	31	10	6	11	0.0	2.9	95.4	0.0	0.0	0.0	0.0
mug100_1	100	166	≈0.0	4	3	3	3	3	2	2	1.6	2.1	64.3	0.0	11.1	0.0	0.0
mug100_25	100	166	≈0.0	4	3	3	3	3	2	2	1.2	1.2	64.0	0.0	25.8	0.0	0.0
mug88_1	88	146	≈0.0	4	3	3	3	3	2	2	0.6	1.9	30.9	0.0	0.0	0.0	0.0
mug88_25	88	146	≈0.0	4	3	3	3	3	2	2	0.8	1.7	30.8	0.0	0.0	0.0	0.0
mulsol.i.1	197	3925	0.2	49	49	49	49	1	4	2	0.1	8.1	1666.2	0.0	0.0	0.0	0.0
mulsol.i.2	188	3885	0.2	31	31	31	31	2	2	2	0.1	7.3	1190.7	0.0	0.0	0.0	0.0
mulsol.i.3	184	3916	0.2	31	31	31	31	2	2	2	0.0	4.9	1018.0	0.0	0.0	0.0	0.0
mulsol.i.4	185	3946	0.2	31	31	31	31	2	2	2	0.0	4.9	1046.6	0.0	0.0	0.0	0.0
mulsol.i.5	186	3973	0.2	31	31	31	31	2	2	2	0.0	6.8	1077.4	0.0	0.0	0.0	0.0
myciel6	95	755	0.2	4	4	3	2	2	2	2	0.1	18.5	24.4	0.0	0.0	0.0	0.0
myciel7	191	2360	0.1	5	4	3	2	2	2	2	1.1	1454.3	1790.9	0.0	0.0	0.0	0.0
qg.order30	900	26100	0.1	30	-	-	30	30	30	30	182.0	tl	tl	0.0	0.0	0.2	0.0
qg.order40	1600	62400	≈0.0	-	-	-	40	40	39	40	tl	tl	tl	0.0	0.0	1.0	0.0
qg.order60	3600	212400	≈0.0	-	-	-	60	60	59	60	tl	tl	tl	0.1	0.1	14.0	0.0
queen10_10	100	1470	0.3	10	10	10	10	10	8	4	0.6	30.3	18.8	0.0	0.0	0.0	0.0
queen11_11	121	1980	0.3	11	11	11	11	11	9	4	0.9	95.0	61.9	0.0	0.0	0.0	0.0
queen12_12	144	2596	0.3	12	12	12	12	12	10	4	1.7	99.6	202.4	0.0	0.0	0.0	0.0
queen13_13	169	3328	0.2	13	13	13	13	13	10	4	3.1	1151.5	504.7	0.0	0.0	0.0	0.0
queen14_14	196	4186	0.2	14	-	14	14	14	11	4	4.9	tl	1351.3	0.0	0.0	0.0	0.0
queen15_15	225	5180	0.2	15	-	-	15	15	12	4	8.5	tl	tl	0.0	0.0	0.0	0.0
queen16_16	256	6320	0.2	16	-	-	16	16	13	4	14.3	tl	tl	0.0	0.0	0.0	0.0
queen8_12	96	1368	0.3	12	12	12	12	12	8	2	0.1	3.9	15.8	0.0	0.0	0.0	0.0
queen8_8	64	728	0.4	9	8	8	8	8	6	4	0.0	2.6	1.3	0.0	0.0	0.0	0.0
queen9_9	81	1056	0.3	9	9	9	9	9	7	4	0.3	6.5	5.5	0.0	0.0	0.0	0.0
r1000.1c	1000	485090	≈1.0	96	95	-	-	41	25	8	1239.8	403.1	tl	tl	0.0	0.3	0.0
school1	385	19095	0.3	14	-	-	14	9	6	2	304.6	tl	tl	0.2	0.2	0.0	0.0
school1_nsh	352	14612	0.2	14	-	-	14	9	6	2	76.5	tl	tl	0.1	0.0	0.0	0.0
wap01a	2368	110871	≈0.0	-	-	-	41	-	8	2	tl	tl	tl	0.0	tl	3.8	0.0
wap02a	2464	111742	≈0.0	-	-	-	40	-	8	2	tl	tl	tl	0.1	tl	4.3	0.0
wap03a	4730	286722	≈0.0	-	-	-	40	-	7	2	tl	tl	tl	0.6	tl	31.2	0.0
wap04a	5231	294902	≈0.0	-	-	-	40	-	7	2	tl	tl	tl	0.2	tl	42.4	0.0
wap05a	905	43081	0.1	50	-	-	50	-	6	2	5.5	tl	tl	0.0	tl	0.2	0.0
wap06a	947	43571	0.1	40	-	-	40	11	6	2	618.8	tl	tl	0.0	23.5	0.2	0.0
wap07a	1809	103368	0.1	-	-	-	40	-	7	2	tl	tl	tl	0.0	tl	1.6	0.0
wap08a	1870	104176	0.1	-	-	-	40	-	7	2	tl	tl	tl	0.0	tl	1.8	0.0
will199GPIA	701	6772	≈0.0	7	-	-	6	-	2	2	3.6	tl	tl	0.0	tl	0.1	0.0
zeroin.i.1	211	4100	0.2	49	49	-	49	1	3	11	0.1	12.5	tl	0.0	0.0	0.0	0.0
zeroin.i.2	211	3541	0.2	30	30	-	30	1	2	3	0.0	13.8	tl	0.0	0.0	0.0	0.0
zeroin.i.3	206	3540	0.2	30	30	-	30	1	2	3	0.0	12.6	tl	0.0	0.0	0.0	0.0

Table 3.11: Comparison between Lower Bounds on $\chi(G)$ for DIMACS instances

Tables 3.10 and 3.11 report the results of the lower bounds presented in Section 2.3 on DIMACS instances. Following Section 2.3.3 analysis, the same conclusions can be drawn. The bounds χ_{G_A} and χ_{G_A} are the strongest bounds.

Comparison between DSATUR$_{\chi_{G_A}}$ and DSATUR

In this section, we compare DSATUR$_{\chi_{G_A}}$ to DSATUR. Table 3.12 reports the results on the new test bed of instances. The table is divided in three parts: in the first part we present the instance features, in the second part we present the results obtained by DSATUR, and in the third part we present the results obtained by DSATUR$_{\chi_{G_A}}$. The same time limit of 1200 seconds is imposed. The following additional average information are reported for each algorithm:

- *#bounds*: the average number of times χ_{G_A} is computed.
- *#bounds**: the average number of times χ_{G_A} is able to prune a node

Table 3.12 clearly shows that DSATUR$_{\chi_{G_A}}$ drastically reduces the number of explored nodes for all sizes and densities. For $n = 60$, DSATUR remains the fastest algorithm. For instances with $d = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6$ and any size, DSATUR also remains the fastest algorithm with the least number of fails. This is expected since the computing time of χ_{G_A} is not fast enough to be efficient as explained in Section 2.4. For very dense instances with $d = 0.8, 0.9$, DSATUR$_{\chi_{G_A}}$ is the fastest algorithms with the least number of fails. For $d = 0.7$, DSATUR is the fastest algorithm for $n = 70, 75$. For all other sizes, DSATUR$_{\chi_{G_A}}$ is the fastest algorithm with the least number of fails. From $n = 85$, DSATUR systematically has more fails and a slower computing time than DSATUR$_{\chi_{G_A}}$. From $n = 95$, DSATUR$_{\chi_{G_A}}$ struggles with instances with density $d = 0.7$, ending up solving none of them from $n = 105$ to $n = 130$. This behavior keeps getting worse since starting from $n = 120$, DSATUR cannot solve any instance for any density. Until $n = 140$, DSATUR$_{\chi_{G_A}}$ is able to solve any instance with density $d = 0.9$.

Again, in Table 3.12, we can see that only a fraction of the calculated bounds (*#bounds*) are able to prune the nodes (*#bounds**). This behavior has also been observed in DSATUR$_{\chi_f^*}$. Thus, in the following section we discuss again the proposed strategies for $\phi_{u,g}(\tilde{C})$ and their impact on the performance of DSATUR$_{\chi_{G_A}}$.

Instance			DSATUR Reimplementation[16]				DSATUR$_{\chi G_A}$				
n	d	#	UB	time	nodes	fails	UB	time	nodes	#bounds* \ #bounds	fails
60	0.1	50	4.00	0.00	11	0	4.00	0.81	1	1.02\1.12	0
60	0.2	50	5.54	0.00	1,141	0	5.34	216.82	159	72.73\158.66	6
60	0.3	50	7.00	0.00	1,054	0	7.00	70.69	39	20.62\39.13	3
60	0.4	50	8.86	0.02	41,490	0	8.82	301.84	330	182.31\330.46	11
60	0.5	50	10.70	0.10	140,806	0	10.69	167.97	311	187.81\310.71	2
60	0.6	50	12.90	0.16	200,882	0	12.90	21.37	65	40.96\64.54	0
60	0.7	50	15.58	0.09	120,228	0	15.58	1.13	7	4.76\7.44	0
60	0.8	50	19.14	0.02	35,216	0	19.14	0.07	2	1.30\1.52	0
60	0.9	50	25.66	0.00	2,957	0	25.66	0.00	1	1.00\1.00	0
70	0.1	50	4.00	0.00	4	0	4.00	1.28	1	1.00\1.00	0
70	0.2	50	6.00	0.00	970	0	6.00	412.82	66	30.55\65.70	17
70	0.3	50	7.84	0.06	83,812	0	7.20	306.84	76	37.40\75.80	40
70	0.4	50	9.68	0.94	1,163,336	0	9.07	213.92	72	42.20\72.20	35
70	0.5	50	11.82	3.42	3,959,720	0	11.61	313.71	238	147.96\237.74	27
70	0.6	50	14.10	3.20	3,518,339	0	14.06	146.44	217	141.19\216.66	3
70	0.7	48	17.21	5.64	6,144,280	0	17.21	10.26	37	24.31\37.13	0
70	0.8	50	21.38	1.18	1,306,601	0	21.38	0.29	3	2.24\3.06	0
70	0.9	50	28.52	0.04	43,187	0	28.52	0.01	1	1.00\1.00	0
75	0.1	49	4.00	0.00	4	0	4.00	2.10	1	1.00\1.00	0
75	0.2	49	6.02	0.00	1,393	0	6.00	356.54	33	15.03\32.84	18
75	0.3	49	7.98	0.05	73,354	0	7.75	578.97	107	58.00\106.75	45
75	0.4	49	10.04	3.91	4,636,774	0	10.00	697.49	175	99.00\175.00	46
75	0.5	49	12.06	13.69	14,912,458	0	11.96	408.06	169	108.50\168.61	21
75	0.6	49	14.88	26.14	27,291,675	0	14.81	259.57	293	192.42\293.45	18
75	0.7	49	17.98	28.78	29,945,679	0	17.98	51.15	141	95.04\141.22	0
75	0.8	49	22.35	7.82	8,072,599	0	22.35	1.03	8	5.29\8.02	0
75	0.9	49	29.94	1.82	1,931,718	0	29.94	0.01	1	1.02\1.06	0
80	0.1	49	4.33	0.00	641	0	4.00	3.30	1	1.00\1.00	16
80	0.2	49	6.29	0.04	50,285	0	6.00	419.81	22	9.90\21.67	28
80	0.3	49	8.24	2.24	2,767,076	0	8.00	794.01	90	48.25\89.75	45
80	0.7	50	18.96	213.54	211,412,127	2	18.96	131.39	319	212.10\318.75	2
80	0.8	50	23.48	72.93	71,749,271	0	23.48	1.60	13	8.32\13.46	0
80	0.9	50	31.34	0.70	655,041	0	31.34	0.02	2	1.34\1.64	0
85	0.7	50	19.31	395.63	373,397,208	15	19.42	207.80	385	259.79\385.28	7
85	0.8	50	24.40	134.99	123,858,470	10	24.42	7.72	53	30.88\52.78	0
85	0.9	50	32.80	22.91	20,330,498	0	32.80	0.01	1	1.00\1.00	0
90	0.7	50	20.25	499.72	453,007,895	46	20.26	400.69	667	429.22\666.96	27
90	0.8	50	25.13	454.77	394,727,070	35	25.40	30.69	193	108.16\192.18	0
90	0.9	50	34.14	22.46	18,789,790	1	34.14	0.03	3	1.34\2.60	0
95	0.7	50	-	-	-	50	21.00	552.58	495	317.38\494.75	42
95	0.8	50	26.50	860.13	751,042,053	48	26.56	84.94	432	253.74\430.76	0
95	0.9	50	35.70	90.99	71,461,097	10	35.70	0.21	17	4.26\16.54	0
100	0.7	50	-	-	-	50	22.00	763.28	823	510.00\822.00	49
100	0.8	50	-	-	-	50	27.49	251.67	1,060	659.59\1058.37	1
100	0.9	50	37.00	274.54	202,716,576	22	37.06	0.64	44	11.10\43.58	0
105	0.7	50	-	-	-	50	-	-	-	-	50
105	0.8	50	-	-	-	50	28.42	401.30	1,454	901.60\1452.38	5
105	0.9	50	38.11	450.11	321,842,577	41	37.96	1.82	101	24.24\100.46	0
110	0.7	50	-	-	-	50	-	-	-	-	50
110	0.8	50	-	-	-	50	29.22	558.31	1,903	1153.97\1901.25	18
110	0.9	50	39.33	674.60	472,118,968	47	39.28	2.69	127	30.68\125.88	0
120	0.7	50	-	-	-	50	-	-	-	-	50
120	0.8	50	-	-	-	50	32.00	853.05	2,393	1383.00\2389.00	48
120	0.9	50	-	-	-	50	41.90	9.44	292	82.46\289.46	11
130	0.7	50	-	-	-	50	-	-	-	-	50
130	0.8	50	-	-	-	50	44.56	19.90	494	148.00\490.78	41
130	0.9	50	-	-	-	50	44.46	28.18	858	167.10\848.12	0
140	0.9	50	-	-	-	50	47.00	68.07	893	305.34\888.26	0

Table 3.12: DSATUR and DSATUR$_{\chi G_A}$ for random instances from [51]

Computing lower bounds only at specific nodes of the branching scheme

Given two input parameter u and g, the function $\phi_{u,g}$ is defined as follow:

$$\phi_{u,g}(\tilde{C}) = (u^* \geq u \text{ AND } \tilde{g} \geq g)$$

where u^* corresponds to the number of times the incumbent solution UB has been updated and, $\tilde{g} = UB - \tilde{k}$ (a measure of the difference between the node lower and upper bound). If $u = 0$ and $g = 0$, the corresponding function $\phi_{0,0}(\tilde{C})$ always returns true, and it corresponds to the case in which the bound is computed at each node of the tree.

The results of DSATUR$_{\chi_{G_A}}$ with different functions $\phi_{u,g}$ are reported in Table 3.13. We test different values of the parameters u and g. For parameter u we test $1, 2, 3$ while for the parameter g we test $2, 3, 4$.

The goal of parameter u is to let the algorithm produce a good upper bound in order to make χ_{G_A} as efficient as possible. From $n = 60$ to $n = 75$, the bounding procedure is not activated for $u = 1, 2, 3$ as the initial upper bound is the optimal solution. It is the best version of DSATUR$_{\chi_{G_A}}$ for these sizes of instances. From $n = 80$ and $u = 1, 2, 3$, DSATUR$_{\chi_{G_A}}$ has the most number of fails and the slowest computing time.

From $n = 90$ to $n = 140$, the regular version of DSATUR$_{\chi_{G_A}}$ is the best one. Looking at the full results, the best version of DSATUR$_{\chi_{G_A}}$ is with $g = 3$ since it totalizes the smallest number of fails (569). It rarely wins for a given size and density, but overall it is the most efficient.

This means that the key to improving DSATUR$_{\chi_{G_A}}$ has no link to parameter g of u. Changing the value of g does not improve the #bounds* $\setminus$ #bounds ratio of DSATUR$_{\chi_{G_A}}$. Each computed bound in DSATUR$_{\chi_{G_A}}$ matters for the efficiency of the algorithm. Changing the value of u does not improve #bounds* $\setminus$ #bounds ratio of DSATUR$_{\chi_{G_A}}$ either. Both the quality of the initial upper bound and the finding speed of the upper bounds from the leaves of the branching tree are good enough for DSATUR$_{\chi_{G_A}}$. Giving DSATUR$_{\chi_{G_A}}$ the possibility to compute a better upper bound does not improve its efficiency. The best version of DSATUR$_{\chi_{G_A}}$ is the regular version using VSR$_{\chi_{G_A}}$ with $\phi_{u,g} = true$.

Instance			$\phi_{0,0}$		$\phi_{0,2}$		$\phi_{0,3}$		$\phi_{0,4}$		$\phi_{1,1}$		$\phi_{2,1}$		$\phi_{3,1}$	
n	d	#	time	fails	time	fails	time	fails	time	fails	time	fails	time	fails	time	fails
60	0.1	50	0.81	0	0.52	0	0.50	0	0.51	0	0.48	0	0.53	0	0.48	0
60	0.2	50	216.82	6	15.70	0	1.96	0	1.96	0	1.86	0	2.06	0	1.87	0
60	0.3	50	70.69	3	16.04	0	2.45	0	2.45	0	2.33	0	2.60	0	2.34	0
60	0.4	50	301.84	11	91.38	0	7.92	0	1.83	0	1.74	0	1.95	0	1.75	0
60	0.5	50	167.97	2	56.68	0	9.26	0	1.47	0	1.09	0	1.22	0	1.10	0
60	0.6	50	21.37	0	11.76	0	4.14	0	1.23	0	0.61	0	0.69	0	0.62	0
60	0.7	50	1.13	0	0.86	0	0.51	0	0.31	0	0.19	0	0.21	0	0.19	0
60	0.8	50	0.07	0	0.06	0	0.04	0	0.03	0	0.02	0	0.03	0	0.02	0
60	0.9	50	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
70	0.1	50	1.28	0	1.21	0	1.17	0	1.16	0	1.10	0	1.22	0	1.10	0
70	0.2	50	412.82	17	49.41	0	9.03	0	8.98	0	8.65	0	9.56	0	8.67	0
70	0.3	50	306.84	40	307.09	13	24.51	0	6.39	0	6.13	0	6.84	0	6.15	0
70	0.4	50	213.92	35	318.56	20	107.96	0	6.21	0	5.30	0	5.89	0	5.30	0
70	0.5	50	313.71	27	273.96	16	186.60	1	23.08	0	5.84	0	6.37	0	5.83	0
70	0.6	50	146.44	3	101.20	1	46.36	0	12.19	0	4.03	0	4.38	0	4.03	0
70	0.7	48	10.26	0	8.11	0	5.53	0	4.89	0	5.30	0	5.71	0	5.29	0
70	0.8	50	0.29	0	0.25	0	0.23	0	0.25	0	0.49	0	0.53	0	0.49	0
70	0.9	50	0.01	0	0.00	0	0.01	0	0.01	0	0.00	0	0.00	0	0.00	0
75	0.1	49	2.10	0	1.93	0	1.87	0	1.87	0	1.76	0	1.96	0	1.76	0
75	0.2	49	356.54	18	76.51	0	14.15	0	14.15	0	13.51	0	14.98	0	13.53	0
75	0.3	49	578.97	45	435.93	14	27.98	0	9.20	0	8.80	0	9.82	0	8.79	0
75	0.4	49	697.49	46	527.81	26	167.07	4	24.17	0	10.43	0	11.45	0	10.41	0
75	0.5	49	408.06	21	250.99	14	155.97	2	28.72	1	15.83	0	17.11	0	15.74	0
75	0.6	49	259.57	18	314.56	10	186.98	1	66.90	0	24.76	0	26.45	0	24.68	0
75	0.7	49	51.15	0	39.48	0	28.15	0	26.41	0	24.97	0	26.80	0	25.15	0
75	0.8	49	1.03	0	0.95	0	1.00	0	1.52	0	3.72	0	4.00	0	3.75	0
75	0.9	49	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0
80	0.1	49	3.30	16	13.89	0	5.60	0	5.59	0	5.34	0	5.86	0	5.34	0
80	0.2	49	419.81	28	103.12	10	25.93	0	25.97	0	24.93	0	27.32	0	24.91	0
80	0.3	49	794.01	45	466.95	22	148.62	2	16.72	0	14.79	0	16.43	0	14.77	0
80	0.7	50	131.39	2	114.17	2	110.28	2	143.87	2	199.09	2	191.77	3	198.93	2
80	0.8	50	1.60	0	1.63	0	3.34	0	8.84	0	48.25	0	51.20	0	47.64	0
80	0.9	50	0.02	0	0.02	0	0.02	0	0.03	0	0.05	0	0.05	0	0.05	0
85	0.7	50	207.80	7	223.57	6	234.36	8	261.05	11	390.67	12	397.64	13	391.76	12
85	0.8	50	7.72	0	9.62	0	38.88	0	42.54	2	80.79	7	83.24	7	77.77	7
85	0.9	50	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0
90	0.7	50	400.69	27	376.00	28	522.77	29	579.56	37	577.15	40	642.50	44	678.65	43
90	0.8	50	30.69	0	28.05	2	87.40	2	171.54	7	254.51	23	222.91	28	228.30	28
90	0.9	50	0.03	0	0.03	0	0.02	0	0.02	0	1.49	0	1.64	0	1.54	0
95	0.7	50	552.58	42	615.93	42	292.06	47	822.77	47	612.59	48	1086.65	49	1038.25	49
95	0.8	50	84.94	0	110.99	1	205.59	10	382.32	24	462.75	38	268.38	44	372.32	43
95	0.9	50	0.21	0	0.19	0	0.17	0	0.25	0	69.81	5	49.23	6	69.50	5
100	0.7	50	763.28	49	955.51	49	-	50	-	50	-	50	-	50	-	50
100	0.8	50	251.67	1	290.58	10	324.88	26	464.75	40	559.17	42	-	50	-	50
100	0.9	50	0.64	0	0.58	0	0.56	0	2.29	0	71.68	11	92.56	13	80.29	14
105	0.7	50	-	50	-	50	-	50	-	50	-	50	-	50	-	50
105	0.8	50	401.30	5	401.25	17	299.02	36	364.58	44	466.33	44	847.68	48	-	50
105	0.9	50	1.82	0	1.67	0	1.72	0	13.85	0	261.60	22	298.41	27	256.53	28
110	0.7	50	-	50	-	50	-	50	-	50	-	50	-	50	-	50
110	0.8	50	558.31	18	519.42	28	516.65	47	958.76	49	307.77	47	-	50	-	50
110	0.9	50	2.69	0	2.43	0	5.88	0	33.81	0	258.57	32	97.59	37	290.69	34
120	0.7	50	-	50	-	50	-	50	-	50	-	50	-	50	-	50
120	0.8	50	853.05	48	871.38	48	-	50	-	50	758.43	49	-	50	-	50
120	0.9	50	9.44	11	10.40	0	25.37	1	88.07	8	250.75	24	161.87	39	72.60	44
130	0.7	50	-	50	-	50	-	50	-	50	-	50	-	50	-	50
130	0.8	50	19.90	41	-	50	-	50	-	50	-	50	-	50	-	50
130	0.9	50	28.18	0	28.71	0	37.37	0	97.53	3	276.27	31	301.64	46	469.14	48
140	0.9	50	68.07	0	67.21	0	76.63	1	148.57	3	172.87	32	264.36	44	189.70	46

Table 3.13: DSATUR$_{\chi_{G_A}}$ and DSATUR$_{\chi_{G_A}}$ with different functions $\phi_{u,g}$

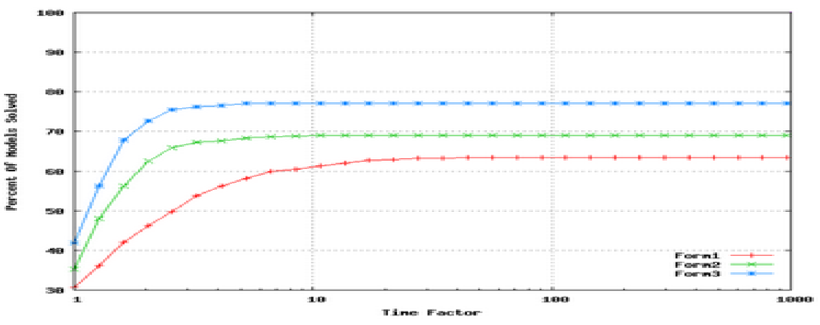

Fig. 1. Performance profile

Figure 3.9: Performance profile

Performance profile

Let $t_{p,s}$, $o_{p,s}$ be the computing time and the objective function value of instance $p \in P$, where P is the set of instances, given by formulation $s \in S = \{Form1, Form2, Form3\}$. Let b_p equal to $\min_{s\in S}\{o_{p,s}\}$. The performance ratio $\rho_s(\tau)$ of a given $\tau \geq 1$ is defined as follows:

$$\rho_s(\tau) = \frac{1}{|P|}\sum_{p\in P, r_{p,s}\leq\tau} 1 \ , \qquad r_{p,s} = \begin{cases} \frac{t_{p,s}}{\min_{s\in S}\{t_{p,s}\}} & \text{if } |\frac{o_{p,s}-b_p}{b_p}| \leq \delta \ , \\ \rho_M & \text{otherwise} \ , \end{cases}$$

where ρ_M is a large enough number such that $\rho_M > r_{p,s}$ for $p \in P$ and $s \in S$. When $\delta = 0$, the ratio $\rho_s(\tau)$ represents the percentage of instances for which a given formulation s returns the best known solutions given time $\tau \min_{s\in S}\{t_{p,s}$ for each instance. Due to the numerical precision of CPLEX, we set $\delta = 10^{-6}$. In the Figure 3.9, the horizontal axis represents the "Time Factor τ" using the logarithmic scale. The vertical axis represents the performance ratio $\rho_s(\tau)$. When $\tau = 1$, the performance ratio gives the percentage of instances that are solved fastest and best by each formulation. By "best" we mean computing the best known objective function value.

Conclusion

In this manuscript, we have tackled coloring problems such as the Vertex Coloring Problem (VCP) and the Minimum Sum Coloring Problem (MSCP). We have used exact methods to solve these problems, namely a Branch-and-Bound based algorithm and Integer Linear Programming (ILP) models.

In the first chapter, we have studied the VCP and proposed an improvement to the Branch-and-Bound algorithm DSATUR.

The idea for improving DSATUR was to inject lower bound into each node of the branching tree as efficiently as possible. Our goal was to enable the algorithm to use these lower bounds to cut potential subtrees of the branching tree, leading to a more efficient algorithm.

We started by studying existing lower bounds for the VCP. We presented them and also introduced a new lower bound. Then, we compared them computationally and chose three promising candidates to inject into DSATUR. Then, we presented a reduced graph associated to a partial coloring. This reduced graph allowed us to devise our new algorithm DSATUR$_{LB}$. In order to make DSATUR$_{LB}$ as efficient as possible, we also proposed new Vertex Selection Rules (VSR) tailored for DSATUR$_{LB}$. We computationally tested these new VSR, and selected for each version of DSATUR$_{LB}$ the most efficient one. Finally we computationally compared DSATUR$_{LB}$ to DSATUR. DSATUR$_{LB}$ is more efficient, as it solves overall more instances with the fastest computing time.

In the second chapter, we have studied the MSCP and propose three ILP formulations designed to solve it.

We started by proposing structural properties of any optimal solution of the MSCP. Then we introduced three ILP formulations designed to solve the MSCP, two compact formulations and one non-compact formulation. Then we define the main components of the algorithms solving these formulations. We introduced the Branch-and-Price algorithm for the non-compact formulation, presenting the column generation algorithm, the branching rule and the initialization procedure.

Then, we compared the first compact formulation and the non-compact formulation on two benchmark of instances. The non-compact formulation is clearly more efficient on the benchmark of random instances in terms of solved instances and computing time. For the DIMACS benchmark, the compact formulation is overall more efficient but the non-compact formulation has more potential. To exploit this potential we have tackle the main difficulties of this non-compact formulation in order to make it more efficient.

We defined an alternative non-compact formulation with a new set of variables. The goal of these variables was to enable the formulation to reduce the number of total columns. It allowed us to compute more efficiently the root lower bound of the formulation. We also devised a new branching rule aimed at reducing the size of the branching tree. The number of available colors is a crucial part of the algorithm, as it defines the number of variables and the number of pricing subproblems we have to solve at each iteration of the column generation algorithm. We developed a procedure producing a potentially new upper bound on this number of colors every time a feasible solution is computed in the algorithm. We also tried to tackle the slow convergence of the column generation algorithm using adaptations of existing methods such as the Boxstep method and the Dual Ascent.

Coloring problems are among the hardest NP-hard problems in the literature. For the VCP, there exists instances with approximately 200 vertices that cannot be solved with an exact method. For the MSCP, most of the DIMACS benchmark cannot be solved either with our exact methods. The most successful algorithms for solving the VCP are based on ILP models. More precisely, formulations using Branch-and-Price algorithms are the most successful so far for the VCP. These formulations have been consistently tested on many problems over the last few years. A crucial part of making these formulations efficient is the implementation. These methods require as much theoretical study as it requires implementation accuracy.

Recently, Benders decomposition formulations have been studied more thoroughly, and made some impressive results. To the best of our knowledge, no Benders decomposition formulation for VCP related problems have been developed. Formulating the

VCP with a Benders decomposition might be the key to improving experimental results on the VCP.

Bibliography

[1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[2] M. A. Boschetti and V. Maniezzo. A set covering based matheuristic for a real-world city logistics problem. *ITOR*, 22(1):169–195, 2015.

[3] M. A. Boschetti, A. Mingozzi, and S. Ricciardelli. A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5(4):735–747, 2008.

[4] D. Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[5] M. B. Campêlo, V. A. Campos, and R. C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7):1097–1111, 2008.

[6] P. E. Coll, J. Marenco, I. Méndez-Díaz, and P. Zabala. Facets of the graph coloring polytope. *Annals OR*, 116(1-4):79–90, 2002.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, (1971).

[8] D. Cornaz, F. Furini, and E. Malaguti. Solving vertex coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 217:151–162, 2017.

[9] D. Cornaz, F. Furini, and E. Malaguti. Solving vertex coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 217:151–162, 2017.

[10] D. Cornaz and V. Jost. A one-to-one correspondence between colorings and stable sets. *Oper. Res. Lett.*, 36(6):673–676, 2008.

[11] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.

[12] J. Edmonds. Covers and packings in a family of sets. *Bulletin of the American Mathematical Society*, 68(5):494–499, 1962.

[13] F. Furini, V. Gabrel, and I. Ternier. Bornes inférieures sur un algorithme exact basé sur DSATUR résolvant la coloration. Congrès annuel de la Société française de Recherche Opérationnelle et d'Aide à la Décision, 2014.

[14] F. Furini, V. Gabrel, and I. Ternier. Lower Bounding Techniques for DSATUR-based Branch and Bound. International Conference on Operations Research (OR2015), 2015.

[15] F. Furini, V. Gabrel, and I. Ternier. Lower Bounding Techniques for DSATUR-based Branch and Bound. *Electronic Notes in Discrete Mathematics*, 52:149–156, 2016.

[16] F. Furini, V. Gabrel, and I.-C. Ternier. Dsatur reimplementation, 2016.

[17] F. Furini, V. Gabrel, and I.-C. Ternier. An Improved DSATUR-Based Branch-and-Bound Algorithm for the Vertex Coloring Problem. *Networks*, 69(1):124–141, 2017.

[18] F. Furini, S. Martin, E. Malaguti, and I. Ternier. ILP Models and Column Generation for the Minimum Sum Coloring Problem. International Network Optimization Conference 2017 (INOC2017), 2017.

[19] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, (2002).

[20] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[21] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[22] P. Hansen, M. Labbé, and D. Schindl. Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, 6(2):135–147, 2009.

[23] S. Held, W. J. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Program. Comput.*, 4(4):363–381, 2012.

[24] A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. In *Proceedings of the 9th metaheuristics international conference*, volume 1101, pages 161–170, (2011).

[25] A. J. Hoffman and L. Howes. On eigenvalues and colorings of graphs, ii. *Annals of the New York Academy of Sciences*, 175(1):238–242, 1970.

[26] Y. Jin, J.-P. Hamiez, and J.-K. Hao. Algorithms for the minimum sum coloring problem: a review. *Artificial Intelligence Review*, 47(3):367–394, 2017.

[27] Y. Jin and J.-K. Hao. Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Information Sciences*, 352:15–34, 2016.

[28] Y. Jin, J.-K. Hao, and J.-P. Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327, 2014.

[29] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, (1972).

[30] C. Koulamas. Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. *Naval Research Logistics (NRL)*, 44(1), 1997.

[31] E. Kubicka and A. J. Schwenk. An introduction to chromatic sums. In *Computer Trends in the 1990s - Proceedings of the 1989 ACM 17th Annual Computer Science Conference, Louisville, Kentucky, USA, February 21-23, 1989*, pages 39–45, 1989.

[32] C. Lecat, C. Lucet, and C.-M. Li. Minimum sum coloring problem: Upper bounds for the chromatic strength. *Discrete Applied Mathematics*, 2017.

[33] Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer. Greedy Algorithms for the Minimum Sum Coloring Problem. In *Logistique et transports*, pages LT–027, Sousse, Tunisia, Mar. (2009).

[34] L. Lovász. An algorithmic theory of numbers, graphs and convexity, 1986.

[35] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[36] E. Malaguti, M. Monaci, and P. Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[37] E. Malaguti and P. Toth. A survey on vertex coloring problems. *ITOR*, 17(1):1–34, 2010.

[38] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[39] I. Méndez-Díaz, G. L. Nasini, and D. E. Severin. A dsatur-based algorithm for the equitable coloring problem. *Computers & OR*, 57:41–50, 2015.

[40] I. Méndez-Díaz and P. Zabala. A polyhedral approach for graph coloring^1. *Electronic Notes in Discrete Mathematics*, 7:178–181, 2001.

[41] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

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

[43] J. Mitchem and P. Morriss. On the cost-chromatic number of graphs. *Discrete Mathematics*, 171(1-3):201–211, 1997.

[44] D. R. Morrison, J. J. Sauppe, E. C. Sewell, and S. H. Jacobson. A wide branching strategy for the graph coloring problem. *INFORMS Journal on Computing*, 26(4):704–717, 2014.

[45] D. R. Morrison, E. C. Sewell, and S. H. Jacobson. Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS Journal on Computing*, 28(1):67–82, 2016.

[46] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[47] G. Palubeckis. On the recursive largest first algorithm for graph colouring. *Int. J. Comput. Math.*, 85(2):191–200, 2008.

[48] G. Palubeckis. Facet-inducing web and antiweb inequalities for the graph coloring polytope. *Discrete Applied Mathematics*, 158(18):2075–2080, 2010.

[49] M. R. Salavatipour. On sum coloring of graphs. *Discrete Applied Mathematics*, 127(3):477–488, 2003.

[50] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[51] P. S. Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers & OR*, 39(7):1724–1733, 2012.

[52] P. S. Segundo, D. Rodríguez-Losada, and A. Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & OR*, 38(2):571–581, 2011.

[53] A. Sen, H. Deng, and S. Guha. On a graph partition problem with application to VLSI layout. *Inf. Process. Lett.*, 43(2):87–94, 1992.

[54] E. C. Sewell. A. improved algorithm for exact graph coloring. In *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13*, pages 359–376, (1993).

[55] C. Thomassen, P. Erdös, Y. Alavi, P. J. Malde, and A. J. Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357, 1989.

[56] A. Verma, A. Buchanan, and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[57] Q. Wu and J.-K. Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[58] Q. Wu and J.-K. Hao. Improved lower bounds for sum coloring via clique decomposition. *arXiv preprint arXiv:1303.6761*, 2013.

[59] Q. Wu and J.-K. Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[60] A. A. Zykov. On some properties of linear complexes. *Matematicheskii sbornik*, 66(2):163–188, 1949.

Résumé

Dans un graphe non orienté, le Problème de Coloration de Graphes (PCG) consiste à assigner à chaque sommet du graphe une couleur de telle sorte qu'aucune paire de sommets adjacents n'aient la même couleur et le nombre total de couleurs est minimisé. DSATUR est un algorithme exact efficace pour résoudre le PCG. Un de ses défauts est qu'une borne inférieure est calculée une seule fois au noeud racine de l'algorithme de branchement, et n'est jamais mise à jour. Notre nouvelle version de DSATUR surpasse l'état de l'art pour un ensemble d'instances aléatoires à haute densité, augmentant significativement la taille des instances résolues.

Nous étudions trois formulations PLNE pour le Problème de la Somme Chromatique Minimale (PSCM). Chaque couleur est représentée par un entier naturel. Le PSCM cherche à minimiser la somme des cardinalités des sous-ensembles des sommets recevant la même couleur, pondérés par l'entier correspondant à la couleur, de telle sorte que toute paire de sommets adjacents reçoive des couleurs différentes. Nous nous concentrons sur l'étude d'une formulation étendue et proposons un algorithme de Branch-and-Price.

Mots Clés

Coloration de graphe, DSATUR, Séparation et Evaluation, Programmation Linéaire en Nombres Entiers, Génération de colonnes, Algorithme de génération de colonnes et branchement.

Abstract

Given an undirected graph, the Vertex Coloring Problem (VCP) consists of assigning a color to each vertex of the graph such that two adjacent vertices do not share the same color and the total number of colors is minimized. DSATUR is an effective exact algorithm for the VCP. We introduce new lower bounding techniques enabling the computing of a lower bound at each node of the branching scheme. Our new DSATUR outperforms the state of the art for random VCP instances with high density, significantly increasing the size of solvable instances. Similar results can be achieved for a subset of high density DIMACS instances.

We study three ILP formulations for the Minimum Sum Coloring Problem (MSCP). The problem is an extension of the classical Vertex Coloring Problem in which each color is represented by a positive natural number. The MSCP asks to minimize the sum of the cardinality of subsets of vertices receiving the same color, weighted by the index of the color, while ensuring that vertices linked by an edge receive different colors. We focus on studying an extended formulation and devise a complete Branch-and-Price algorithm.

Keywords

Graph Coloring, DSATUR, Branch and Bound, Integer Linear Programming, Column Generation, Branch-and-Price.