Métaheuristiques Guidées par l'Apprentissage pour la Coloration de Graphe

Cyril Grelier

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

université angers

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ D'ANGERS

ÉCOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Cyril GRELIER

Métaheuristiques Guidées par l'Apprentissage pour la Coloration de Graphe

Thèse présentée et soutenue à Angers, le 12-12-2023
Unité de recherche : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)
Thèse N° : 264016

Rapporteurs avant soutenance :

Tristan CAZENAVE Professeur, LAMSADE, Université Paris-Dauphine
Sébastien VEREL Professeur, LISIC, Université du Littoral Côte d'Opale

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Nicolas DURAND	Professeur, OPTIM, ENAC Toulouse
Examinateurs :	Tristan CAZENAVE	Professeur, LAMSADE, Université Paris-Dauphine
	Béatrice DUVAL	Professeure émérite, LERIA, Université d'Angers
	Sébastien VEREL	Professeur, LISIC, Université du Littoral Côte d'Opale
Dir. de thèse :	Jin-Kao HAO	Professeur, LERIA, Université d'Angers
Co-enc. de thèse :	Olivier GOUDET	Maître de conférences, LERIA, Université d'Angers

Invité(s) :

Daniel PORUMBEL Maître de conférences, CEDRIC, CNAM Paris

REMERCIEMENTS

D'abord, je tiens à remercier mon directeur de thèse Jin-Kao Hao, Professeur à l'Université d'Angers, et mon co-encadrant de thèse Olivier Goudet, Maître de Conférences à l'Université d'Angers, pour leur supervision, aide, conseils, soutien ainsi que tout ce qu'ils m'ont appris pendant ces trois ans. Ils m'ont permis d'enrichir mes connaissances théoriques et méthodologiques tout au long de la thèse pour pouvoir continuer à travailler dans la recherche. Je les remercie pour leur gentillesse et leurs encouragements apportés pendant cette thèse.

Je remercie aussi Tristan Cazenave, Professeur à l'Université Paris-Dauphine, et Sébastien Verel, Professeur à l'Université du Littoral Côte d'Opale, d'avoir accepté de rapporter cette thèse ainsi que Béatrice Duval, Professeure émérite à l'Université d'Angers, et Nicolas Durand, Professeur à l'ENAC Toulouse, pour avoir suivi ma thèse lors des comités de suivi individuel en m'offrant un autre point de vue et des conseils sur les travaux en cours, mais aussi Daniel Porumbel, Maître de Conférences au CNAM Paris, pour avoir accepté de rejoindre le jury en tant que membre invité.

Je tiens aussi à remercier les membres du laboratoire, tout d'abord David Lesaint, Professeur à l'Université d'Angers, pour avoir accepté de travailler avec nous pour implémenter des modèles de programmation par contrainte liés aux travaux de la thèse. Adrien Goëffon, Professeur à l'Université d'Angers, pour ses conseils pour analyser l'espace de recherche et ses remarques très pertinentes sur la réduction de sommets. Merci à tous ceux qui ont pu m'aider pour le développement, la théorie, la méthodologie, l'enseignement et bien d'autres sujets pendant ces trois ans. Merci à Catherine Pawlonski, Christine Bardaine, Isabelle Prat et Emmanuelle Baudouin pour leur aide administrative tout au long de la thèse et leur gentillesse. Merci aux doctorants, à mes collègues de bureau et plus largement à tous les collègues du laboratoire qui ont rendu ces années de thèse plus conviviales que ça soit lors de sorties entre les confinements et ensuite, lors de conférences, lors des parties de jeux de société ou simplement lors des pauses déjeuner où leur présence était fort appréciée.

Merci à tous les enseignants des différentes formations que j'ai pu suivre qui m'ont poussé à poursuivre les études et qui m'ont permis de découvrir la recherche.

Je remercie également mes parents et toute ma famille pour leur soutien pendant toutes ces années malgré les quelques détours dans mes études.

Merci à mes amis qui m'accompagnent depuis toutes ces années qui m'ont permis de passer de bons moments pendant cette thèse que ça soit à distance ou en présentiel.

Merci à Suzu, dont les miaulements et ronrons ont été considérés comme des encouragements.

Un grand merci à Séverine pour tout son soutien et ses encouragements pendant toute la thèse.

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE

Contexte

Le problème de coloration de graphe (GCP - *Graph Coloring Problem* ou VCP - *Vertex Coloring Problem*) est un problème très connu dans le domaine de l'optimisation combinatoire. Il est à la fois facile à comprendre et à représenter, mais aussi NP-difficile. Étant donné un graphe composé de sommets et d'arêtes liant ces sommets, l'objectif est de colorer chaque sommet de manière à ce que deux sommets voisins ne partagent pas la même couleur tout en utilisant le moins de couleurs possibles. Le GCP peut modéliser diverses applications telles que la planification d'emplois du temps, l'affectation de fréquences radio et bien d'autres, comme le jeu de Sudoku. Il existe de nombreuses extensions et variantes du problème de coloration de graphe pour répondre à différentes problématiques et applications. Parmi les plus populaires nous pouvons citer le problème de somme coloration minimale (MSCP - *Minimal Sum Coloring Problem*), dont l'objectif est la minimisation de la somme des valeurs des couleurs utilisées pour colorer les sommets, ou encore le problème de coloration équitable (ECP - *equitable coloring problem*), pour lequel le nombre de sommets par couleur doit être le plus proche possible. Nous étudions ici le problème de coloration pondérée (*WVCP - Weighted Vertex Coloring Problem*), pour lequel chaque sommet possède un poids et dont l'objectif est de minimiser la somme des poids des sommets les plus lourds de chaque couleur. Ce problème trouve des applications, entre autres, dans l'ordonnancement de tâches en parallèle ou dans la décomposition de matrices. Dans la littérature, les problèmes de coloration ont été très étudiés avec différentes méthodes exactes et approchées. De plus en plus, des méthodes d'apprentissage sont utilisées pour résoudre ce type de problème d'optimisation combinatoire, afin de guider des procédures de résolution dans leur recherche.

Objectifs

Dans cette thèse, nous nous sommes intéressés à la résolution des problèmes de coloration GCP et WVCP. Ces deux problèmes sont liés et nous montrerons notamment que la résolution du GCP peut aider à trouver des bornes utiles pour la résolution du WVCP. Par ailleurs, contrai-

rement à la plupart des méthodes de la littérature qui s'intéressent à la résolution d'un seul problème de coloration, nous étudierons ici si des méthodes proposées pour le GCP peuvent s'étendre à un autre problème de coloration plus général comme le WVCP. Tout d'abord, nous proposons d'étudier des propriétés théoriques des problèmes GCP et WVCP, qui permettent de réduire le nombre de variables et l'espace de recherche pour des instances difficiles de ces problèmes. Nous présenterons ensuite des méthodes heuristiques de type Monte Carlo Tree Search (MCTS), qui explorent l'arbre des solutions légales en étant guidées par de l'apprentissage, notamment pour le positionnement des sommets de poids les plus lourds dans les différents groupes de couleurs. Pour en améliorer les performances, nous avons hybridé ce MCTS à des algorithmes de recherche locale. Toujours dans le cadre du MCTS, nous avons ensuite cherché à apprendre à sélectionner les bons opérateurs de recherche locale de manière adaptative au cours de la résolution. Une comparaison de différentes techniques d'apprentissage pour le choix de ces opérateurs, allant de simples stratégies probabilistes à des techniques d'apprentissage profond, sera présentée. Enfin, nous proposons d'étendre le travail effectué sur le choix dynamique d'opérateurs au cours de la résolution d'une instance donnée, au cadre des algorithmes mémétiques, qui alternent l'utilisation d'opérateurs de recherche locale avec des opérateurs de croisement au sein d'une population de solutions candidates.

Contributions

Les contributions réalisées durant cette thèse ont été présentées dans 3 articles de conférences internationales, 2 articles de revues internationales (dont 1 en cours de relecture) et 3 communications dans des conférences nationales (voir le Chapitre 8.4 pour la liste complète des références). Nous avons aussi réalisé un dernier travail pour une revue encore non soumis. Les contributions de cette thèse sont les suivantes :

- une étude sur des arbres de recherche de Monte Carlo hybridés avec des algorithmes gloutons et des recherches locales dans le cadre du WVCP. Nous avons étudié tout d'abord la possibilité de remplacer la phase de simulation aléatoire de l'algorithme MCTS par un algorithme glouton. Afin d'améliorer les résultats nous avons proposé ensuite d'appliquer une recherche locale. Ce projet a fait l'objet d'un article à la conférence EVOCOP 2022 [Grelier *et al.*, 2022], suivi par une version étendue pour la revue *SN Computer Science* (en cours de relecture) et une communication nationale pour la conférence ROADEF 2022. Depuis ces publications, nous avons étudié l'application de cet algorithme au GCP. Cette extension n'a pas été publié, mais vous sera présentée.

- un travail sur un algorithme mémétique utilisant de l'apprentissage profond pour sélectionner des croisements dans une très large population. Cet algorithme, applicable à la fois au GCP et au WVCP, utilise un réseau de neurones pour guider la sélection des croisements à appliquer sur une population de solutions candidates dans une très large population. Ce travail est publié dans la revue *Knowledge-Based System* [Goudet *et al.*, 2022] et a aussi fait l'objet d'une communication nationale à la conférence ROADEF 2023.
- une étude sur la sélection automatique d'opérateurs de recherche locale dans un arbre de recherche de Monte Carlo appliquée au WVCP. Après avoir analysé les impacts des différents opérateurs de recherche locale sur les performances de l'algorithme MCTS et constaté que les performances variaient en fonction des instances, nous avons cherché à apprendre à sélectionner les bons opérateurs de recherche locale de manière adaptative au cours de la recherche. Ce travail est publié à la conférence EVOCOP 2023 [Grelier *et al.*, 2023] a aussi fait l'objet d'une communication nationale à la conférence ROADEF 2023.
- un travail sur la réduction de sommets, de bornes pour le WVCP et la mise au point de modèles de programmation par contrainte. Dans ce travail, une étude théorique a été menée sur la réduction de sommets et la recherche de bornes pour le WVCP. Nous avons notamment découvert de nouvelles bornes supérieures sur le score et le nombre de couleurs utilisées pour le WVCP. Nous avons ensuite proposé trois modèles de programmation par contrainte basés sur une représentation primale, duale et un couplage de ces deux représentations pour le WVCP. Ce travail a été publié pour la conférence IJCAI 2023 [Goudet *et al.*, 2023].
- un travail sur la sélection automatique d'opérateurs de croisement et de recherche locale dans le cadre d'un algorithme mémétique pour le GCP et le WVCP. Ce travail n'a pas encore été publié.

Plan

Le plan de la thèse est le suivant :

1. Introduction aux Problèmes de Coloration de Graphe Étudiés : présentation des problèmes de coloration de graphe, de leurs applications et des instances de référence de la littérature.
2. État de l'Art pour les Problèmes de Coloration : état de l'art autour des thématiques de la thèse. Entre autres, nous présenterons les méthodes exactes, approchées, d'apprentissage

et diverses hybridations.

3. Réduction : implémentation de règles de réduction pour le WVCP et GCP.
4. Nouvelles Bornes pour le WVCP : découvertes de nouvelles bornes supérieures sur le score et le nombre de couleurs utilisées pour le WVCP.
5. MCTS et Algorithmes Gloutons : présentation d'un arbre de recherche de Monte Carlo combiné à des algorithmes gloutons pour le WVCP et GCP.
6. MCTS et Recherches Locales : présentation d'un arbre de recherche de Monte Carlo et d'une hybridation avec des recherches locales pour le WVCP.
7. MCTS et Hyperheuristiques : présentation d'un arbre de recherche de Monte Carlo avec des méthodes de sélection automatique de recherches locales pour le WVCP.
8. Algorithmes Mémétiques et Hyperheuristiques : présentation d'un algorithme mémétique avec sélection automatique d'opérateurs de croisement et de recherche locale pour le GCP et WVCP.

CHAPITRE 1

INTRODUCTION AUX PROBLÈMES DE COLORATION DE GRAPHE ÉTUDIÉS

Dans ce chapitre, nous présentons d'abord le problème de coloration de graphe standard (GCP) puis nous introduisons la coloration de graphe pondéré (WVCP), en présentant des applications réelles qui peuvent être modélisées avec ces représentations. Nous présentons ensuite les espaces de recherche étudiés ainsi que les instances de référence utilisées dans l'état de l'art.

1.1 Le Problème de Coloration de Graphe Standard

Le problème de coloration de graphe standard, *Graph Coloring Problem* (GCP) ou *Vertex Coloring Problem* (VCP), est un problème d'optimisation combinatoire très connu qui consiste à colorer l'ensemble des sommets d'un graphe avec le moins de couleurs possible, de façon à ce que deux sommets voisins ne partagent pas la même couleur.

Étant donné un graphe $G = (V, E)$ non orienté, où V est l'ensemble des n sommets du graphe et E l'ensemble des m arêtes reliant les sommets, le GCP peut typiquement être résolu en résolvant une succession de problèmes de k-coloration, pour k décroissant. Le problème de k-coloration de graphe consiste à colorer chaque sommet de V avec exactement k couleurs de façon à ce que la contrainte de coloration soit respectée. Il s'agit donc de trouver une solution S correspondant à une partition de l'ensemble V des sommets du graphe en k ensembles de sommets indépendants V_i tel que $S = \{V_1, \ldots, V_k\}$ avec $V_i \cap V_j = \emptyset$, $\forall i, j \in [1, k], i \neq j$ et $V_1 \cup \cdots \cup V_k = V$. Un ensemble de sommets V_i est dit indépendant s'il ne contient pas deux sommets du graphe reliés par une arête. Ce problème de k-coloration peut aussi être vu comme un problème d'affectation. On définira alors $c(v)$ comme la couleur du sommet $v \in V$:

$$\begin{aligned} c : V &\rightarrow [1, k] \\ v &\mapsto c(v), \end{aligned}$$

et on dira que $c(v) = i$ si $v \in V_i$. Pour la k-coloration, de façon à obtenir une solution légale (sans conflits), l'objectif est souvent de minimiser une fonction de *fitness* f correspondant au nombre de conflits dans la solution.

Pour une solution S donnée, on a $f(S) = \sum_{\{u,v\} \in E} \delta_{uv}$ avec :

$$\delta_{uv} = \begin{cases} 1 & \text{si } u \in V_i, v \in V_j \text{ et } i = j \text{ si } i \neq 0, \\ 0 & \text{sinon.} \end{cases} \tag{1.1}$$

Le nombre chromatique $\chi(G)$ correspond au nombre minimum de couleurs k nécessaires pour colorer le graphe G de façon légale.

Trouver $\chi(G)$ et donc résoudre le GCP est NP-difficile [Garey et Johnson, 1979], ce qui signifie qu'à moins que $\mathcal{P} = \mathcal{NP}$, il n'existe pas d'algorithmes permettant de résoudre ce problème de façon optimale en un temps polynomial.

La Figure 1.1 illustre un exemple où $\chi(G) = 3$, il faut donc au minimum 3 couleurs pour colorer ce graphe.

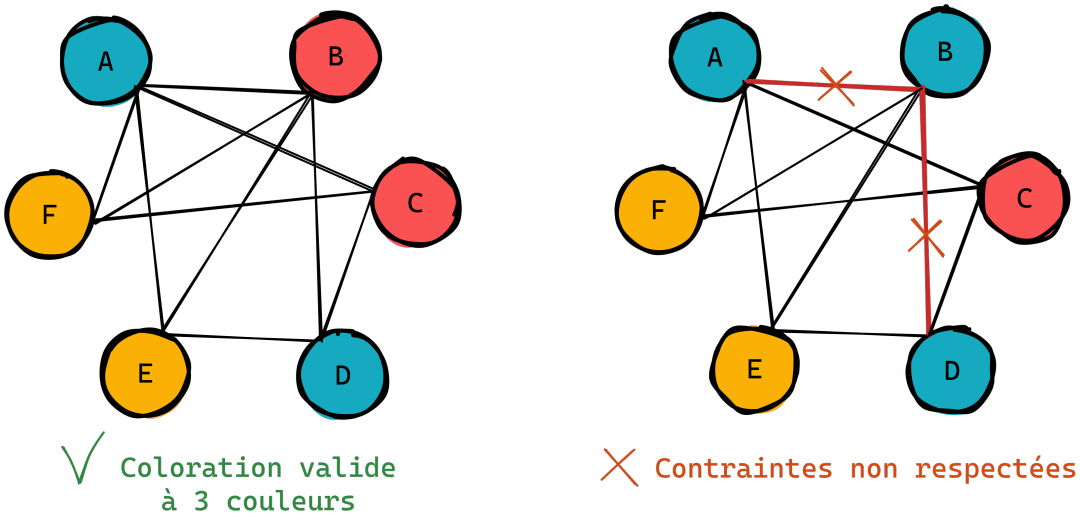


FIGURE 1.1 – À gauche, exemple de coloration valide à 3 couleurs. À droite, exemple de coloration invalide, car B partage la même couleur que ses voisins A et D.

Les applications du GCP sont variées [Lewis, 2015], on pourra citer entre autres :

- la planification d'emplois du temps (affectation de salles pour des étudiants et professeurs, deux cours ne peuvent pas avoir lieu dans la même salle au même moment et un étudiant/professeur ne peut assister qu'à un seul cours à la fois),
- l'affectation de fréquences radio (les fréquences doivent être différentes pour deux stations proches),
- le jeu de Sudoku,
- etc.

La Figure 1.2 donne un exemple d'application du GCP pour le jeu de Sudoku. Pour compléter une grille de Sudoku, le joueur dispose de chiffres de 1 à 9 (ou couleurs), et certaines cases (ou sommets) sont déjà colorées. Une case est liée aux cases de la même ligne, même colonne et du même bloc (il existe un arc entre ces sommets). Un Sudoku facile comme celui présenté sur la Figure 1.2 peut être résolu avec un algorithme simple, car à tout moment de la résolution il existe au moins une case suffisamment contrainte par ses voisines pouvant être colorée de manière optimale.

1				9	2		7	4
9	2	4	1			5	3	
		3	5	6				2
6			4		7			8
		2		5		4	6	7
5			8		6		9	
		6		8		7		
	5	9			3			1
4			6	1		3		9

FIGURE 1.2 – Exemple d'application du GCP avec un Sudoku. Le joueur dispose de 9 chiffres (ou couleurs) pour remplir toutes les cases de cette grille. Les voisins d'une case sont les cases de la même ligne, de la même colonne ou d'un même bloc.

1.2 Le Problème de Coloration Pondérée

Il existe de nombreuses variantes du problème de coloration de graphe dans la littérature, comme le problème de coloration équitable ou de somme coloration d'un graphe [Formanowicz et Tanaś, 2012, Lewis, 2015]. Parmi celles-ci, nous avons étudié, en plus du GCP, le problème de coloration pondérée ou *Weighted Vertex Coloration Problem* (WVCP). Une instance du WVCP est définie par un graphe pondéré $G = (V, E, w)$ où w est une fonction associant à chaque sommet $v \in V$ un poids $w(v)$ strictement positif. L'objectif du WVCP est de minimiser la somme des poids des sommets les plus lourds de chaque groupe de couleurs :

$$f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$$

. Pour ce problème, le nombre de couleurs k à utiliser pour obtenir une solution optimale n'est pas imposé. Cependant, on sait qu'une borne inférieure du nombre k de couleurs à utiliser pour obtenir une solution optimale du WVCP est χ_G et qu'une borne supérieure est $\Delta + 1$ (avec Δ le degré maximum d'un sommet dans le graphe) [Demange *et al.*, 2007]. Nous verrons dans le Chapitre 4 que cette borne supérieure du nombre de couleurs nécessaires pour obtenir une solution optimale pourra être affinée pour certaines classes d'instances.

Le WVCP est une généralisation du GCP et est donc NP-difficile. En effet, le GCP peut être vu comme un cas spécial du WVCP où tous les poids sont égaux à 1, car dans ce cas, minimiser $f(S)$ revient à minimiser le nombre de couleurs utilisées.

Dans certains cas, une solution avec plus de couleurs peut voir son score diminuer et donc s'améliorer. La Figure 1.3 montre un exemple de ce genre de cas où il est préférable d'utiliser une couleur de plus pour réduire le score.

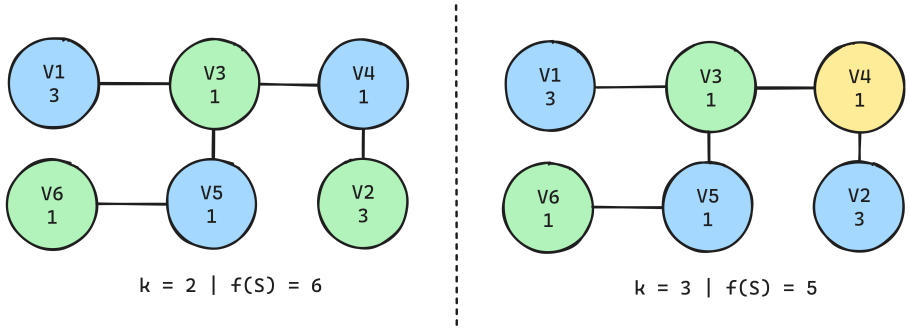


FIGURE 1.3 – Exemple d'un cas du WVCP où utiliser 2 couleurs (nombre chromatique du GCP) apporte au mieux un score de 6, alors qu'une 3e couleur permet d'obtenir le score optimal de 5 pour le WVCP.

Les applications du WVCP sont plus spécifiques que pour le GCP. On pourra citer entre autres [Ribeiro *et al.*, 1989, Prais et Ribeiro, 2000, Boudhar et Finke, 2000, Liu *et al.*, 2006, Halldórsson et Shachnai, 2008, Furini et Malaguti, 2012] avec notamment :

- la décomposition de matrices,
- la planification de tâches par lots dans un environnement parallèle,
- la gestion de mémoire vive,
- la conception de réseaux métropolitains,
- l'affectation du trafic pour des réseaux de télécommunications.

La Figure 1.4 donne un exemple d'application du WVCP pour la planification de ressources dans un environnement parallèle. Ce problème de planification consiste à exécuter un ensemble de tâches en un temps le plus court possible. Les tâches ont différentes durées et ont besoin d'avoir un accès exclusif à certaines ressources (1). Il est possible d'exécuter ces tâches en parallèle par lot pour accélérer le processus. Cependant deux tâches ayant besoin de la même ressource ne peuvent pas être lancées dans le même lot. Le temps d'exécution d'un lot correspond au temps de la plus longue tâche de celui-ci. Ce problème peut être modélisé facilement avec le WVCP. En effet, en reliant deux à deux les tâches ayant besoin des mêmes ressources (2) et en affectant à chaque tâche la durée de celle-ci comme poids (3), on obtient un graphe pondéré. Une solution optimale du WVCP pour ce graphe (4) permettra d'établir un plan optimal de partitionnement des tâches en différents lots (5), pour obtenir le temps d'exécution le plus court possible.

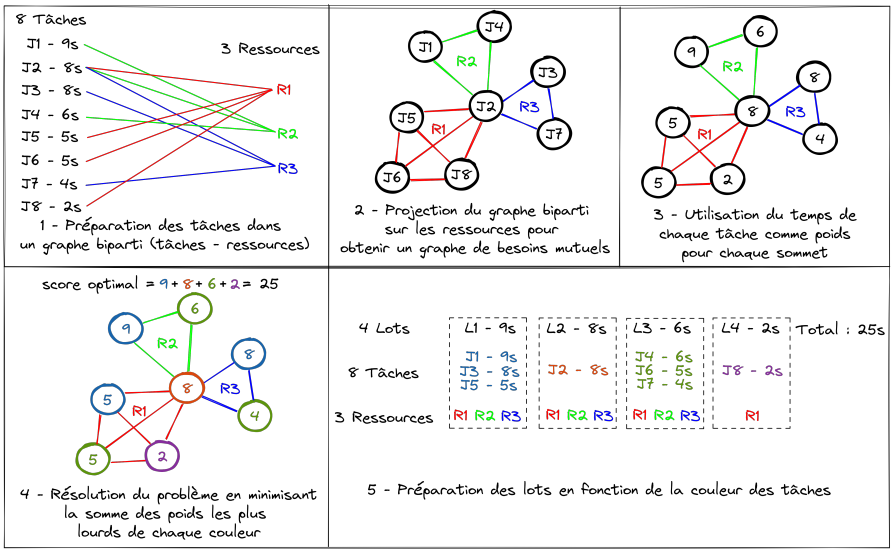


FIGURE 1.4 – Exemple d'application du WVCP pour la planification de ressources.

1.3 Instances

Les problèmes GCP et WVCP ont été très largement étudiés dans la littérature. Pour comparer les différents algorithmes de résolutions pour ces problèmes sur une même base, la littérature dispose d'instances de références (ou *benchmark*). Certaines de ces instances ont été introduites depuis plus de 30 ans grâce à de nombreux articles ou bien sont issues de challenges de coloration comme le challenge DIMACS [Johnson et Trick, 1996]. La liste suivante énumère les différents types d'instances classiques de la littérature. n correspond au nombre de sommets dans le graphe, p à la probabilité de lien entre deux sommets et χ au nombre chromatique du graphe (déjà connu par construction pour certaines instances).

- Les DSJC[n].[p] et C[n].[p] [Johnson *et al.*, 1991, Johnson et Trick, 1996] sont des graphes aléatoires.
- Les DSJR[n].[p] et r[n].[p] [Johnson *et al.*, 1991] sont des graphes géométriques et les DSJR[n].[p].c et r[n].[p].c leurs graphes complémentaires.
- Les flat[n]_[χ]_0 [Culberson, 2001] sont des graphes quasi aléatoires.
- Les instances géométriques GEOM[n] [Johnson et Trick, 1996] ont été générées à partir de points répartis sur une grille de taille 10000x10000 et connectés s'ils sont suffisamment proches les uns des autres.
- Les graphes GPIA proviennent de problèmes de partitionnement de matrices.
- Les fpsol2, inithx, mulsol et zeroin proviennent de problèmes réels sur de l'allocation de registres dans des codes [Lewandowski et Condon, 1993].
- Les FullIns et Insertions viennent du challenge DIMACS [Johnson et Trick, 1996]. Ils proviennent d'une généralisation des graphes de Mycielski avec des sommets ajoutés pour augmenter la taille du graphe sans augmenter sa densité.
- Les graphes Leighton, le[n]_[χ] [Leighton, 1979] sont des graphes construits avec un nombre chromatique connu.
- Les graphes latin_square_10 [Lewandowski et Condon, 1993] viennent du problème de carré latin comme pour les qg.order[x] [Gomes et Shmoys, 2002].
- Les school1 et school1_nsh [Lewandowski et Condon, 1993] sont des graphes issus de problèmes de planification réels pour des écoles avec ou sans salles.
- Les graphes SGB [Johnson et Trick, 1996] viennent de la *Stanford GraphBase* de Donald Knuth et sont divisés en plusieurs catégories :
  - Les graphes issus de livres. Chaque sommet représente un personnage. Deux sommets sont liés si les personnages se rencontrent dans le livre. Pour les livres : Anna

Karenina de Tolstoy (anna), David Copperfield de Dickens (david), l'Iliad d'Homer (homer), Huckleberry Finn de Twain (huck) et Les Misérables d'Hugo (jean).

- Les graphes de rencontres sportives. Chaque équipe est représentée par un sommet. Deux sommets sont liés si les équipes se sont rencontrées.
- Les graphes de villes (miles). Chaque ville des États-Unis est représentée par un sommet. Deux sommets sont liés si les deux villes correspondantes sont suffisamment proches l'une de l'autre. Ces graphes sont géométriques.
- Les graphes du problème des n-reines (queen). L'objectif est de placer n reines sur un échiquier de taille n sans qu'elles puissent s'attaquer. Chaque sommet représente une case du jeu d'échecs. Deux sommets sont liés si les cases sont sur la même ligne, colonne ou diagonale. On sait que le nombre chromatique de ces graphes est n.

- Les graphes de Mycielski (myciel) [Johnson et Trick, 1996] sont générés de manière à avoir un nombre chromatique élevé sans triangles (cliques de taille 2 au maximum).
- Les graphes pxx et rxx [Prais et Ribeiro, 2000] correspondent à des problèmes de décomposition de matrices.
- Les instances wap[x]a [Koster, 2005] sont issues de problèmes de routage optique.
- Les instances R[n].[p](g/gb) sont issues des problèmes de multicoloration.

Une partie des graphes utilisés pour le GCP l'est aussi pour le WVCP. Les graphes dont les sommets ne possèdaient pas de poids à l'origine s'en sont vus attribuer de manière aléatoire par [Sun *et al.*, 2018]. Dans ce travail de thèse, nous avons travaillé avec 188 instances pour le WVCP et 244 instances pour le GCP.

Le Tableau 1.1 liste une partie des instances utilisées pour le GCP et le WVCP. Pour chaque instance sont indiqués le nombre de sommets ($|V|$), le nombre d'arêtes ($|E|$) et la densité du graphe ($d = \frac{2m}{n(n-1)}$). Pour le WVCP sont aussi indiqués, pour chaque instance, le poids minimum et maximum (w_{min} et w_{max}), le nombre de poids différents ($|W|$) et le ratio entre le nombre de poids différents et le nombre de sommets ($|W|/|V|$). Les colonnes vides pour certaines instances signifient qu'elles n'existent que pour le GCP. Les deux dernières colonnes, BKS WVCP et BKS GCP correspondent au meilleur score connu dans l'état de l'art (*Best Known Score*) pour chaque problème. Une étoile indique une preuve d'optimalité. Sans preuve d'optimalité, le BKS correspond à la borne supérieure sur le score. Ces bornes supérieures et preuves d'optimalité sont issues de très nombreuses recherches, depuis une trentaine d'années pour le GCP, depuis cinq à plus de vingt ans pour le WVCP (articles que nous citons dans le chapitre suivant sur l'état de l'art). Trouver de nouveaux meilleurs scores pour ces instances, en particulier pour le GCP, est donc une tâche très difficile.

instance	$\|V\|$	$\|E\|$	d	w_max	w_mean	w_min	$\|W\|$	$\|W\|/\|V\|$	BKS WVCP	BKS GCP
C2000.5	2000	999836	0.5	19	10.1	1	19	0.01	2144	145
C2000.9	2000	1799532	0.9	19	10.3	1	19	0.01	5477	408
C4000.5	4000	4000268	0.5							259
DSJC1000.1	1000	49629	0.1	19	9.9	1	19	0.02	300	20
DSJC1000.5	1000	249826	0.5	19	10	1	19	0.02	1185	82
DSJC1000.9	1000	449449	0.9	19	9.5	1	19	0.02	2836	222
DSJC500.1	500	12458	0.1	19	9.9	1	19	0.04	184	12
DSJC500.5	500	62624	0.5	19	10.4	1	19	0.04	685	47
DSJC500.9	500	112437	0.9	19	10	1	19	0.04	1662	126
DSJR500.1c	500	121275	0.97							85*
DSJR500.5	500	58862	0.47							122*
flat1000_60_0	1000	245830	0.49	19	10.2	1	19	0.02	1162	60*
flat1000_76_0	1000	246708	0.49	19	9.9	1	19	0.02	1165	76*
GEOM120a	120	1434	0.2	10	5.5	1	10	0.08	105*	16*
GEOM120b	120	1491	0.21	3	2	1	3	0.03	35*	16*
GEOM120	120	773	0.11	10	5.7	1	10	0.08	72*	11*
latin_square_10	900	307350	0.76	19	10.3	1	19	0.02	1480	97
le450_15a	450	8168	0.08	19	10	1	19	0.04	212	15*
le450_15b	450	8169	0.08	19	10.2	1	19	0.04	216	15*
le450_15c	450	16680	0.17	19	9.6	1	19	0.04	275	15*
le450_15d	450	16750	0.17	19	9.4	1	19	0.04	272	15*
le450_25a	450	8260	0.08	19	10.3	1	19	0.04	306	25*
le450_25b	450	8263	0.08	19	9.7	1	19	0.04	307	25*
le450_25c	450	17343	0.17	19	10	1	19	0.04	342	25*
le450_25d	450	17425	0.17	19	9.6	1	19	0.04	330	25*
miles1000	128	3216	0.4	19	9.6	1	19	0.15	431*	42*
miles1500	128	5198	0.64	19	10.3	1	19	0.15	797*	73*
mulsol.i.5	186	3973	0.23	19	9.9	1	19	0.1	367*	31*
myciel7	191	2360	0.13							8*
myciel7gb	191	2360	0.13	20	10.6	1	20	0.1	109	
myciel7g	191	2360	0.13	5	3	1	5	0.03	29	
p41	116	900	0.13	568	119.7	6	62	0.53	2688*	13*
p42	138	1186	0.13	568	119.8	8	62	0.45	2466*	14*
queen10_10	100	1470	0.3	19	10.3	1	19	0.19	162	11*
r1000.1c	1000	485090	0.97							98
r1000.1	1000	14378	0.03							20*
r1000.5	1000	238267	0.48							234
r29	281	3553	0.09	703	447.7	201	213	0.76	8693*	17*
r30	301	4122	0.09	704	448.8	201	228	0.76	9816*	19*
R75_1g	70	251	0.1	5	3.1	1	5	0.07	18*	4*
R75_5g	75	1407	0.51	5	3.1	1	5	0.07	51*	13
R75_9g	75	2513	0.91	5	2.9	1	5	0.07	110*	33*
wap01a	2368	110871	0.04	19	10.2	1	19	0.01	545	41*
wap02a	2464	111742	0.04	19	10	1	19	0.01	538	40*
wap03a	4730	286722	0.03	19	10	1	19	0.004	562	43
wap04a	5231	294902	0.02	19	10	1	19	0.004	563	41
wap05a	905	43081	0.11	19	9.7	1	19	0.02	542	50*
wap06a	947	43571	0.1	19	10	1	19	0.02	516	40*
wap07a	1809	103368	0.06	19	10.1	1	19	0.01	555	41
wap08a	1870	104176	0.06	19	10	1	19	0.01	529	40*

TABLE 1.1 – Liste d'une partie des instances de référence du GCP et du WVCP.

Une des catégories d'instances les plus difficiles est celle des graphes aléatoires DSJC[n].[p] et C[n].[p] dont très peu d'instances ont été résolues de façon optimale, tout comme pour l'instance latin_square_10 qui est très difficile. On notera par ailleurs que des indices statistiques très convaincants ont été apportés par [Gondran et Moalic, 2019] pour montrer qu'utiliser 47 couleurs pour l'instance très connue et très difficile DSJC500.5 serait optimal. Les instances flat[n]_[χ]_0 sont construites avec un nombre chromatique donné, celui-ci est donc connu, mais certaines restent extrêmement difficiles comme flat1000_76_0 qui n'est résolue au mieux qu'à 81 couleurs dans l'état de l'art, alors que cette instance a été proposée depuis plus de 30 ans. Les wap sont des instances assez difficiles, car très grandes, mais elles possèdent parfois des zones à forte densité entourées de zones peu denses, ce qui a permis de prouver l'optimalité pour certaines d'entres elles pour le GCP.

1.4 Espaces de Recherche pour les Problèmes de Coloration

L'espace de recherche pour lister de manière exhaustive toutes les solutions de coloration possibles pour un graphe est très grand. Par exemple, pour un graphe de 100 sommets, le nombre de couleurs maximums utilisables est de 100. Une recherche exhaustive naïve avec toutes les combinaisons de couleurs possibles pour chaque sommet du graphe reviendrait donc à explorer 100^{100}, soit 10^{200} solutions possibles. Cependant, les problèmes GCP et WVCP partagent la propriété d'être invariants par permutation des couleurs, c'est-à-dire que l'ordre des couleurs n'a pas d'importance et qu'une permutation des indices de couleurs dans une solution n'a pas d'impact sur le score. Nous pouvons donc réduire cet espace de recherche en ne considérant que les différentes partitions possibles de l'ensemble des sommets du graphe et donc plutôt considérer l'espace de recherche suivant :

$$\Omega = \{\{V_1, \ldots, V_k\} | \cup_{i=1}^{k} V_i = V, V_i \cap V_j = \emptyset, i \neq j, 1 \leq i, j \leq k, 1 \leq k \leq |V|\}. \quad (1.2)$$

La taille de cet espace de recherche peut être calculée de façon exacte avec le nombre de Bell $B_n = \sum_{k=0}^{n} \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$, ce qui laisse tout de même dans ce cas $4.76 * 10^{115}$ partitions possibles à évaluer pour notre graphe à 100 sommets. Pour donner un ordre d'idée, le nombre d'atomes dans l'univers observable est estimé à 10^{82}. Explorer toutes les solutions de cette manière est donc difficilement réalisable, même pour des graphes de taille relativement réduite. Il peut donc être intéressant de définir des espaces de recherche plus restreints, mais

surtout de les explorer de manière plus performante qu'une simple énumération.

Trois espaces de recherche sont généralement utilisés dans la littérature :

- l'espace des solutions légales ou valides pour lesquelles aucun voisin ne partage la même couleur et tous les sommets sont colorés (sans limites sur le nombre de couleurs) :

$$\Omega_l = \{\{V_1, \ldots, V_k\} | \cup_{i=1}^{k} V_i = V, V_i \cap V_j = \emptyset, \forall v_1, v_2 \in V_i \wedge (v_1, v_2) \notin E\} \quad (1.3)$$

  avec $1 \leq k \leq |V|$ et $i \neq j, 1 \leq i, j \leq k$.
- l'espace des solutions partiellement légales pour lesquelles aucun voisin ne partage la même couleur, mais les sommets ne sont pas tous systématiquement colorés :

$$\Omega_p = \{\{V_1, \ldots, V_k, U\} | \cup_{i=1}^{k} V_i \cup U = V, V_i \cap V_j = \emptyset, \forall v_1, v_2 \in V_i, (v_1, v_2) \notin E\}, \quad (1.4)$$

  avec U l'ensemble des sommets non colorés d'une solution. Cet espace de recherche est très utilisé par les algorithmes d'exploration d'arbres et les méthodes exactes notamment.
- l'espace des solutions illégales dans lesquelles tous les sommets sont colorés, mais des voisins peuvent partager la même couleur et au plus k-couleurs sont utilisées :

$$\Omega_k = \{\{V_1, \ldots, V_k\} | \cup_{i=1}^{k} V_i = V, V_i \cap V_j = \emptyset, i \neq j, 1 \leq i, j \leq k\} \quad (1.5)$$

  Cet espace est principalement exploré par les méthodes approchées dont l'objectif est de trouver une solution légale avec un nombre réduit de couleurs. Nous définirons aussi un ensemble C de sommets en conflit pour les voisins partageant la même couleur :

$$c(v_1) = c(v_2) \wedge (v_1, v_2) \in E \implies v_1 \in C, v_2 \in C$$

  .

1.4.1 Exploration d'un Espace de Recherche

Pour explorer un espace de recherche, certaines méthodes, qui vous seront présentées dans le chapitre suivant sur l'état de l'art, utilisent des opérateurs de voisinage. Partant d'une solution, ces opérateurs permettent de générer des solutions voisines en modifiant la solution courante.

Voici deux opérateurs très utilisés dans la littérature pour le GCP et le WVCP :

one-move est probablement l'opérateur le plus populaire. Il fait passer un sommet v d'un groupe de couleurs V_i à un autre groupe de couleurs V_j. Si au plus k couleurs sont utilisées, le voisinage d'une solution S qui peut être atteint avec cet opérateur sera noté :

$$N_{o-m}(S) = \{S \oplus < v, V_i, V_j > | v \in V, 1 \leq i \leq k, 1 \leq j \leq |V|, i \neq j\}.$$

Selon l'espace de recherche exploré, ce déplacement pourra autoriser ou non à déplacer un sommet vers une couleur déjà utilisée par un de ses voisins. Il travaille généralement dans les espaces Ω_l ou Ω_k respectivement définis par les Équations 1.3 et 1.5.

grenade est aussi un opérateur très utilisé. Il consiste à affecter une couleur à un sommet en retirant les voisins du sommet de la couleur, de façon à conserver la solution légale. Nous noterons $N_{gre}(S)$ son voisinage. Ce voisinage est utilisé dans le cas d'un espace de recherche partiellement légal Ω_p (Équation 1.4). Avec cet opérateur, une solution voisine $S' = \{V'_1, \ldots, V'_k, U'\}$ peut être obtenue à partir d'une solution $S = \{V_1, \ldots, V_k, U\}$ en colorant un sommet $v \in U$ avec une couleur c. Il se produit alors les changements suivants :

- l'ensemble des sommets non colorés est réduit du sommet v : $U' = U \setminus \{v\}$
- v reçoit la couleur c : $V'_c = V_c \cup \{v\}$
- les voisins de v dans V_c perdent leur couleur : $V'_c = V'_c \setminus N(v)$, $U' = U' \cup N(v)$

Pour résumer, $U' = (U \setminus \{v\}) \cup (N(v) \cap V_c)$ et $V'_c = (V_c \cup \{v\}) \setminus N(v))$.

Dans le chapitre suivant, nous détaillerons des recherches locales proposées dans la littérature pour le GCP et le WVCP qui utilisent ces opérateurs de voisinage.

1.4.2 Distance entre Solutions

Pour identifier à quel point une solution est éloignée d'une autre dans l'espace de recherche, il est nécessaire de définir un critère de distance.

Une distance pouvant être utilisée pour différencier une ou plusieurs solutions de coloration de façon pertinente est la distance de partitionnement. [Porumbel *et al.*, 2011] l'introduit de la manière suivante. Pour deux solutions S_1 et S_2 on a

$$|V| = Similarité(S_1, S_2) + Distance(S_1, S_2),$$

où le terme $Distance(S_1, S_2)$ correspond au nombre minimum de sommets à déplacer d'une partition à l'autre dans S_1 pour obtenir les mêmes partitions que S_2. À l'inverse, le terme

$Similarité(S_1, S_2)$ correspond au nombre de sommets qui n'ont pas besoin d'être déplacés. La fonction *Distance* peut aussi être vue comme le nombre d'applications de l'opérateur de voisinage *one-move* nécessaires pour passer de S_1 à S_2.

On calculera donc la distance entre deux solutions S_1 et S_2 comme :

$$Distance(S_1, S_2) = |V| - Similarité(S_1, S_2) = |V| - MAX_\sigma(\sum_{i=1}^{k} T_{i,\sigma(i)})$$

où T est la matrice de similarité entre les partitions de S_1 et S_2, $T_{i,j} = |S_1^i \cap S_2^j|$, et σ est une correspondance une à une des partitions de S_1 vers celles de S_2.

Nous remarquerons que la distance entre deux solutions n'est pas dépendante de l'ordre des partitions. Ainsi, ce calcul de distance est invariant par permutation des couleurs, car inverser deux partitions dans une solution ne change pas la distance entre les solutions.

Cette distance est aussi utilisée dans l'article [Porumbel *et al.*, 2010b] pour cartographier l'espace de recherche du GCP en utilisant une recherche locale avec l'opérateur de voisinage *one-move*. Ils montrent alors que les minimums locaux de très basse *fitness* pour la k-coloration sont généralement regroupés dans des clusters de solutions qui peuvent être représentés par des sphères de rayon $10\%|V|$.

1.4.3 Voisinages du GCP et du WVCP

Pour mieux différencier les problèmes et estimer à quel point l'espace de recherche est constitué de plateaux, c'est-à-dire de zones de l'espace de recherche où toutes les solutions voisines ont le même score, nous avons analysé le taux de solutions voisines neutres (solutions avec le même score) avec des méthodes utilisant l'opérateur de voisinage *one-move*. Les méthodes utilisées sont des recherches locales, *TabuCol* pour le GCP présenté dans le chapitre suivant et *TabuWeight* pour le WVCP présenté dans le Chapitre 6.

Pour une instance aléatoire très difficile, DSJC500.5, nous avons récupéré le taux de solutions voisines améliorantes et neutres pour les deux méthodes au cours d'une recherche. La Figure 1.5 présente l'évolution de ces taux, en bleu nous retrouvons le taux de voisins neutres, en orange le taux de voisins améliorants. Pour le GCP, le score indiqué en abscisse correspond au nombre de conflits dans la solution pour un objectif de 47 couleurs, pour le WVCP, il correspond à la somme des poids des sommets les plus lourds de chaque groupe de couleurs.

Dans le cas du GCP, le taux de voisins neutres est de 0.5% environ en moyenne, sachant qu'il commence en début de recherche à environ 9% puis tend vers des valeurs inférieures à

0.5% de manière décroissante lorsqu'il atteint des zones de l'espace de recherche avec de bons scores. Pour le taux de voisins améliorants, la tendance est la même avec un taux d'améliorants à environ 7% en début de recherche et qui tend rapidement vers 0% lorsque le score est minimisé.

Dans le cas du WVCP, le taux de voisins neutres est de 5% environ en moyenne et celui-ci varie très peu au cours de la recherche. Il est compris entre 4 et 6%, peu importe le score. Pour les voisins améliorants, le taux est extrêmement faible et ne dépasse pas les 1% tout au long de la recherche et ne semble pas corrélé au score.

Pour résumer, l'exploration avec un même opérateur de voisinage est très différente pour le GCP et le WVCP. Les taux de voisins neutres et de voisins améliorants dans le cas du GCP diminuent lorsque le score décroît ce qui semble indiquer la présence de bassins d'attraction plus simple à explorer, mais peut être durs à quitter. En revanche, pour le WVCP, les taux de voisins neutres sont très élevés et les taux d'améliorants sont très faibles, les deux ne semblent pas dépendre du score, ce qui semble indiquer un espace de recherche constitué de beaucoup plus de plateaux sur lesquels il est possible d'effectuer un grand nombre de mouvements sans jamais améliorer le score.

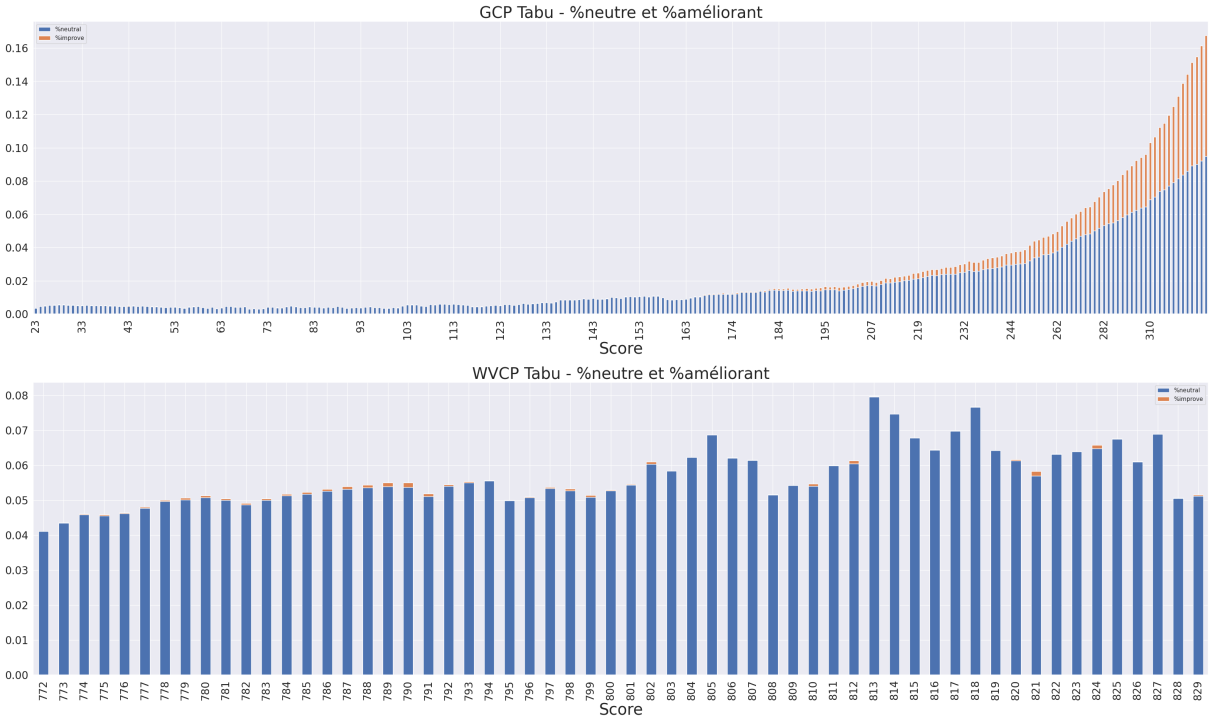


FIGURE 1.5 – Évolution du taux de voisins neutres (bleu) et du taux de voisins améliorants (orange) au cours d'une recherche locale pour le GCP et le WVCP. L'objectif étant de minimiser le score, le début de la recherche se situe à droite et le score diminue en allant vers la gauche.

1.5 Conclusion

Le GCP et le WVCP sont deux problèmes proches en raison de leur nature, ils utilisent généralement les mêmes instances à la différence des poids sur les sommets. La présence des poids qui interviennent dans le calcul du score du WVCP est une différence importante, car elle redéfinit totalement l'espace de recherche. Les sommets les plus importants du problème deviennent uniquement les plus lourds de chaque couleur. Ainsi, déplacer les sommets les moins lourds ne change pas le score, ce qui conduit à un taux de voisins neutres très élevé dans le voisinage du WVCP. L'état de l'art, dans le chapitre suivant, présentera les stratégies mises en place, parfois très différentes, pour naviguer dans les multiples espaces de recherche associés à ces deux problèmes.

CHAPITRE 2

ÉTAT DE L'ART POUR LES PROBLÈMES DE COLORATION

Le problème de coloration de graphe est un problème NP-complet étudié depuis le XIXe siècle en mathématiques et depuis les années 1970 comme un problème algorithmique, en particulier dans le domaine de l'optimisation combinatoire. Depuis, de nombreux chercheurs ont travaillé sur ce problème pour proposer de nouveaux théorèmes, chercher de nouvelles bornes et proposer de nouveaux algorithmes.

L'optimisation combinatoire réunit un ensemble de méthodes permettant de résoudre des problèmes d'optimisation dans des espaces discrets. Ces méthodes peuvent être exactes ou approchées. Les méthodes exactes permettent d'obtenir la solution optimale d'un problème, tout en prouvant l'optimalité. Par la nature NP-difficile du problème, les méthodes exactes ne peuvent généralement résoudre dans un temps raisonnable que des instances de petite taille ou des instances ayant des propriétés particulières. Pour des graphes plus grands, les méthodes approchées permettent d'obtenir en un temps raisonnable une solution possiblement optimale ou proche de l'être sans pouvoir, en général, le prouver.

Depuis plus d'un siècle, une autre branche des mathématiques, de l'informatique et des sciences cognitives s'est développée pour créer le domaine de l'intelligence artificielle. Ce domaine très vaste regroupe de nombreuses méthodes, dont les méthodes d'optimisation combinatoire mais aussi l'apprentissage automatique. L'apprentissage automatique (ou *Machine Learning* en anglais) est une discipline de l'intelligence artificielle dont l'objectif est de conférer aux machines la capacité d'acquérir des connaissances à partir de données en utilisant des modèles mathématiques. L'apprentissage automatique est de plus en plus utilisé en soutien des méthodes d'optimisation combinatoire pour résoudre des problèmes difficiles.

Cette section présente des méthodes exactes, approchées, d'apprentissage et diverses hybridations utilisées dans l'état de l'art pour résoudre des problèmes de coloration de graphe.

2.1 Méthodes Exactes

Les méthodes exactes permettent de résoudre un problème d'optimisation combinatoire en prouvant l'optimalité de la solution. Elles explorent de manière exhaustive l'espace de recherche en coupant une partie de celui-ci, soit lors de l'exploration, soit dans la formulation du problème.

2.1.1 Méthodes Exactes pour le GCP

[Malaguti et Toth, 2010] liste de nombreuses méthodes exactes utilisées pour résoudre le GCP. Parmi celles-ci, on retrouve le *backtracking* [Kubale et Jackowski, 1985], les méthodes de *branch and bound* ou de *branch and price* [Held *et al.*, 2012]. Celles-ci représentent le problème sous forme d'un arbre et explorent des solutions partiellement légales en divisant le problème en sous-problèmes jusqu'à atteindre des solutions complètes ou jusqu'à des points de coupure si la branche explorée ne peut pas mener à des solutions de meilleure qualité que la meilleure solution déjà trouvée. Ces méthodes sont utilisées pour la programmation linéaire en nombres entiers (ILP) [Malaguti *et al.*, 2011]. Le problème est formulé comme un modèle d'optimisation d'une fonction linéaire avec des contraintes linéaires sur les variables du modèle. Pour les problèmes de grande taille, une méthode de génération de colonnes a été testée par [Mehrotra et Trick, 1996]. La programmation par contraintes (CP - *Constraint Programming*) a aussi été utilisée [Régin, 2003]. Selon ce paradigme, le problème est alors représenté avec un modèle composé de variables possédant des domaines de valeurs et de contraintes limitant ces domaines. Des méthodes hybrides de CP avec de la génération de colonnes ou des méthodes SAT ont également été proposées [Junker *et al.*, 1999, Gualandi et Malucelli, 2012, Hébrard et Katsirelos, 2020]. Une transformation des problèmes de coloration en problèmes SAT, qui utilise des clauses logiques évaluées jusqu'à obtenir une solution satisfiable, a aussi été récemment employée avec un certain succès [Ignatiev *et al.*, 2017, Heule *et al.*, 2022].

Par ailleurs, certaines de ces méthodes convertissent le problème de coloration en un problème de recherche de clique maximum sur le graphe complémentaire afin d'améliorer la borne inférieure du nombre de couleurs ou en représentant le graphe sous différentes formulations : avec un encodage binaire [Lee, 2002] ou bien avec des "super nœuds" [Burke *et al.*, 2010b]. Un "super nœud" étant un sous-graphe du graphe d'origine dans lequel chaque paire de sommets a le même voisinage en dehors du sous-graphe.

2.1.2 Méthodes Exactes pour le WVCP

Le WVCP a aussi été abordé avec plusieurs méthodes exactes. [Ribeiro *et al.*, 1989] et [Furini et Malaguti, 2012] ont utilisé des méthodes de génération de colonnes ou du *branch and price*. [Malaguti *et al.*, 2009] a proposé 3 modèles ILP avec différentes ruptures de symétries en deux phases : une alternance entre de la génération de colonnes et de l'optimisation de colonnes avec l'aide des modèles ILP. Plus récemment, [Cornaz *et al.*, 2017] avec l'algorithme MWSS a proposé une reformulation du problème sous forme d'une recherche d'ensembles indépendants de poids maximum et le résout avec un solveur MIP (*Mixed-Integer Programming*) avec de bons résultats.

Au cours de cette thèse, nous avons aussi contribué aux méthodes exactes avec l'article [Goudet *et al.*, 2023] qui propose trois modèles CP pour le WVCP. Le premier propose un encodage primal du problème, travaillant sur le graphe d'origine, le second propose un encodage dual, adaptation de [Cornaz *et al.*, 2017] pour la CP et le troisième utilise les propriétés des deux encodages liés avec d'autres contraintes pour profiter des forces des deux premiers modèles. Ce travail sur les modèles CP n'est cependant pas présenté plus en détail dans ce manuscrit.

Toutes ces méthodes obtiennent généralement de bons résultats sur la grande majorité des instances de petite taille (jusqu'à une centaine de sommets) et sur quelques instances avec des configurations particulières. On peut penser par exemple aux graphes de la famille des *wap* qui sont des graphes de grande taille avec une grande hétérogénéité des degrés, ce qui rend leur résolution exacte généralement plus facile.

2.2 Méthodes Approchées

Depuis plus de 30 ans, de très nombreuses heuristiques ont été proposées pour résoudre les problèmes de coloration de graphe. Nous référons le lecteur à ces revues bibliographiques [Malaguti et Toth, 2010, Galinier *et al.*, 2013]. Dans cette section, sans être complètement exhaustifs, nous présenterons les trois grandes familles d'heuristiques qui ont été introduites dans la littérature pour résoudre ces problèmes : les approches constructives, les recherches locales et les algorithmes à base de population. Nous détaillerons plus spécifiquement certaines de ces méthodes qui mettent en jeu des mécanismes que nous avons repris dans le cadre de ce travail de thèse.

2.2.1 Approches Constructives

Une première famille d'heuristiques très simple pour obtenir une solution en un temps court est celle des heuristiques constructives, comme l'algorithme glouton (ou *greedy algorithm*) qui peut être appliqué aussi bien pour résoudre le GCP que le WVCP.

Cet algorithme consiste à parcourir les sommets du graphe un par un en leur affectant une couleur respectant les contraintes de coloration sans modifier les décisions précédentes.

Algorithmes Constructifs pour le GCP

Au fil du temps, plusieurs variantes d'un tel algorithme constructif ont été proposées pour résoudre le GCP. Parmi celles-ci, on peut citer notamment :

Welsh-Powell [Welsh et Powell, 1967] propose un algorithme glouton, qui applique la première couleur disponible à chaque sommet après les avoir triés par degré. Celui-ci a influencé les algorithmes gloutons que nous avons développés dans le Chapitre 5. Mais aussi, les algorithmes constructifs de la littérature *DSatur* et *RLF*, présentés ci-après, qui utilisent des règles plus dynamiques pour la sélection des prochains nœuds à colorer.

DSatur (*Degree of Saturation*) de [Brélaz, 1979] utilise le degré de saturation des sommets pour choisir le prochain sommet à colorer. Le degré de saturation d'un sommet est le nombre de couleurs différentes utilisées par ses voisins.

RLF (*Recursive Largest First*) de [Leighton, 1979] cherche à maximiser la taille des ensembles indépendants dans le but d'ouvrir le moins de groupes de couleurs possible pour colorer le graphe.

Algorithmes Constructifs pour le WVCP

L'une des premières métaheuristiques pour le WVCP est R-GRAPS (*Reactive Greedy Randomized Adaptive Search Procedure*) [Prais et Ribeiro, 2000]. R-GRASP est un algorithme qui itère en deux étapes. Premièrement, il initialise une solution avec un algorithme glouton. Deuxièmement, il améliore la solution avec une procédure de recherche locale (ces recherches locales seront détaillées dans la section suivante). À chaque itération, il supprime aléatoirement la couleur d'une partie des sommets et recommence à la première étape. Le choix des sommets à recolorer est géré par un paramètre adaptatif qui évolue en fonction de la qualité de la solution.

Plus récemment, [Sun *et al.*, 2018, Wang *et al.*, 2020, Nogueira *et al.*, 2021] ont utilisé des variantes de l'algorithme de *Welsh-Powell* pour le WVCP afin d'initialiser leurs solutions avant d'appliquer une recherche locale. Les sommets sont triés par poids avant d'être colorés un par un, pour trouver une solution légale avec un score du WVCP aussi bas que possible.

Ces méthodes constructives permettent d'obtenir une première solution rapidement, mais généralement de qualité médiocre. Elles servent donc souvent de point de départ pour d'autres méthodes. Après l'utilisation d'un algorithme glouton, une recherche locale (LS) ou un algorithme évolutionnaire, tel qu'un algorithme mémétique (MA) est généralement utilisé pour améliorer la qualité de la solution. Ces méthodes seront présentées dans les deux prochaines sections.

2.2.2 Recherches Locales

Un algorithme de recherche locale (LS - *Local Search*) explore l'espace de recherche en partant d'une solution et lui applique un opérateur de voisinage donné à chaque itération. À chaque étape, l'application d'un opérateur de voisinage op (cf. Section 1.4.1) sur une solution S permet d'atteindre une nouvelle solution S' qui appartient au voisinage $N_{op}(S)$ de la solution S. On espère ainsi atteindre des solutions de meilleure qualité au fil de la recherche.

L'Algorithme 1 présente une recherche locale générique pour un problème de coloration donné qui utilise un opérateur de voisinage op. Parmi le voisinage $N_{op}(S)$ d'une solution S dans un espace de recherche donné (cf. Section 1.4), on sélectionne généralement un voisin parmi ceux qui minimisent le plus la fonction objectif f et qui respectent éventuellement d'autres critères (par exemple ceux qui n'ont pas été explorés lors des dernières itérations comme détaillé plus bas pour l'algorithme *TabuCol*). Si la solution est meilleure que la meilleure solution trouvée jusqu'ici, celle-ci est conservée et sera retournée à la fin de la recherche.

Algorithme 1 Algorithme de recherche locale générique

Entrée: $G = (V, E)$ un graphe, S une solution et un opérateur op.
Sortie: S^* la meilleure solution trouvée

1: Initialiser S aléatoirement ou bien avec un algorithme constructif (cf Section 2.2.1).
2: $S^* \leftarrow S$
3: **tant que** la condition d'arrêt n'est pas atteinte **faire**
4: $\quad S \leftarrow S'$ où S' est une solution voisine dans $N_{op}(S)$
5: $\quad$ **si** $f(S) < f(S^*)$ **alors**
6: $\quad\quad S^* \leftarrow S$
7: **retourne** S^*

Recherches Locales pour le GCP

De nombreuses recherches locales ont été proposées pour le GCP. Nous ne les détaillerons pas toutes dans cette section, seulement les deux plus connues et les plus efficaces à savoir *TabuCol* et *PartialCol*, que nous avons utilisées dans nos travaux (cf. Chapitres 5 et 8).

TabuCol (*Tabu Coloring*) de [Hertz et Werra, 1987] est une recherche locale qui utilise une liste tabou pour éviter de rester bloquée dans un minimum local. Cet algorithme part d'une solution illégale avec k couleurs puis explore l'espace illégal des solutions avec le voisinage *one-move*. Son objectif est de vider l'ensemble C des sommets en conflit, le score $f(S)$ à minimiser est donc le nombre de sommets dans C. Lorsqu'un sommet est déplacé dans une couleur, il devient tabou (c'est-à-dire qu'il ne peut plus être choisi) pendant plusieurs itérations afin de quitter un minimum local. Le nombre d'itérations, pendant lesquelles un sommet est dit tabou, est calculé comme étant $nb_{iter} = 0.6 \times |C| + random(0..9)$[Dorne et Hao, 1999, Galinier et Hao, 1999].

Cette recherche locale *TabuCol* a inspiré une nouvelle recherche locale *TabuWeight* que nous présenterons dans le Chapitre 7 pour résoudre le WVCP. Dans les calculs présentés dans cette thèse, lorsqu'il s'agira de la procédure *TabuCol*, nous utiliserons toujours sa version optimisée en termes de performance algorithmique qui a été récemment proposée par [Moalic et Gondran, 2018] pour le GCP.

PartialCol de [Blöchliger et Zufferey, 2008] explore l'espace partiellement légal avec l'opérateur *grenade* détaillé plus haut. Son objectif est de vider l'ensemble U des sommets non colorés, son score $f(S)$ à minimiser est donc le nombre de sommets non colorés. La liste tabou fonctionne de manière similaire à *TabuCol*, mais au lieu d'utiliser $|C|$ pour calculer le nombre d'itérations au cours desquelles un mouvement sera tabou, $|U|$ est utilisé.

D'autres méthodes basées sur des recherches locales ont été utilisées dans la littérature pour résoudre le GCP tel que les LNS (*Large Neighborhood Search*) de [Trick et Yildiz, 2007], une recherche locale avec deux voisinages larges, ou bien les VSS (*Variable Space Search*) [Hertz *et al.*, 2008] qui utilisent plusieurs recherches locales pour naviguer dans des espaces de recherches différents, en passant de l'un à l'autre avec diverses transformations.

Recherches Locales pour le WVCP

Le problème WVCP a aussi été traité avec ces méthodes. Plusieurs de ces procédures seront aussi utilisées comme opérateurs de recherche locale dans les Chapitres 7 et 8.

AFISA de [Sun *et al.*, 2018] est une recherche locale itérée utilisant une liste tabou. Cette procédure de recherche locale navigue entre l'espace légal Ω_l et illégal Ω des solutions en proposant un coefficient ϕ ajustant automatiquement l'importance des pénalités face au score de la solution. Dans le cas du WVCP, pour une solution S légale, on rappelle que son score est $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$ (cf. Chapitre précédent). *AFISA* autorise temporairement des conflits dans la solution S courante et cherche à minimiser une fonction de *fitness* étendue correspondant à $f'(S) = f(S) + \phi \cdot |C|$, avec $|C|$ le nombre de conflits dans S. Le coefficient ϕ augmente en fonction du nombre d'itérations sans réduction du nombre de conflits dans la solution et diminue dans le cas contraire. Ce coefficient adaptatif permet donc de gérer un compromis entre le score WVCP de la solution et son nombre de conflits. *AFISA* utilise l'opérateur de voisinage *one-move* pour explorer l'espace de recherche. L'algorithme intègre en plus deux mécanismes de perturbation différents, une recherche locale où chaque déplacement *one-move* différent ne peut être appliqué qu'une seule fois, et une autre sans liste tabou.

RedLS de [Wang *et al.*, 2020] est une recherche locale qui utilise une stratégie dite de *configuration checking* [Cai *et al.*, 2011]. Comme *AFISA*, *RedLS* explore les espaces de recherche illégaux et légaux. Cet algorithme applique plusieurs stratégies d'amélioration et de perturbation pour explorer l'espace de recherche. Lorsqu'une solution légale est trouvée, *RedLS* perturbe la solution en déplaçant tous les sommets les plus lourds d'un groupe de couleurs à un autre groupe, avant de minimiser le nombre de conflits pour récupérer une nouvelle solution légale. Il utilise différentes variantes de l'opérateur de voisinage *one-move* pour réduire le nombre de conflits tout en maintenant le score WVCP aussi bas que possible. Chaque arête en conflit a un poids qui est augmenté à chaque fois qu'elle n'est pas résolue, pour donner la priorité à sa résolution pour les prochaines itérations. Par ailleurs, dans leur article, les auteurs proposent aussi une règle de réduction de sommets pour le WVCP que nous évoquerons plus en détail dans le Chapitre 3. Nous nous sommes aussi inspirés de leur recherche locale pour le WVCP pour en proposer une nouvelle version dédiée pour le GCP, nommée *TabuEdges* et qui sera présentée dans le Chapitre 8.

ILS-TS de [Nogueira *et al.*, 2021] est une recherche locale itérée explorant un espace de recherche partiellement légal en utilisant un opérateur de voisinage *grenade*. À partir d'une solution complète, l'algorithme *ILS-TS* effectue itérativement 2 étapes : (i) il supprime les sommets les plus lourds de 1 à 3 groupes de couleurs V_i et les place dans l'ensemble des sommets non colorés U pour perturber la solution ; (ii) il améliore la solution (c'est-à-dire minimise le score $f(S)$) en appliquant différentes variantes des opérateurs *one-move* et *grenade* jusqu'à ce que l'ensemble des sommets non colorés U devient vide. Lors des déplacements des sommets dans la solution, une liste du nombre de couleurs disponibles n'incrémentant pas le score pour chaque sommet est maintenue à jour. Cette liste permet ensuite lors de l'utilisation de l'opérateur *grenade* de ne choisir que les voisins qui au plus n'ajouteront qu'un seul sommet dans U, car les autres sommets auront au moins un groupe de couleur disponible dans lequel ils n'augmenteront pas le score. Les différents voisinages permettent de choisir en priorité les déplacements qui vident le plus de sommets de U ou améliorent le plus le score.

Limites des Recherches Locales Ces algorithmes de recherche locale peuvent être bloqués dans des minima locaux s'ils utilisent uniquement les opérateurs de voisinage *one-move* ou *grenade*, notamment dans le cas du WVCP qui comporte de nombreuses zones de plateaux (cf. Section 1.4.3), pour lesquelles le score ne change pas sur de larges zones de l'espace de recherche. C'est pourquoi nous avons vu que différents mécanismes ont été introduits dans la littérature pour diversifier la recherche, comme la liste tabou. L'algorithme *ILS* (*Iterated Local Search*) de [Chiarandini *et al.*, 2002] a introduit la notion de recherche locale itérée. Le principe est d'utiliser des perturbations lorsque la recherche locale est bloquée dans un minimum local pour explorer de nouvelles zones de l'espace de recherche. Ce principe se retrouve aussi dans *ILS-TS* par exemple pour le WVCP.

Pour des instances très difficiles de graphes, ces méthodes de perturbation ne permettent généralement pas de diversifier suffisamment la recherche. C'est pourquoi des algorithmes à base de population que nous présentons maintenant ont été introduits dans la littérature.

2.2.3 Algorithmes Mémétiques

Une des familles d'algorithmes à base de population la plus efficace pour résoudre des problèmes de coloration est celle des algorithmes mémétiques. Ce sont des métaheuristiques hybrides qui utilisent un algorithme de recherche locale au sein d'un algorithme à base de population. Ils sont une variante de l'algorithme génétique original [Holland, 1992] (aussi applicable à la coloration [Glass et Prügel-Bennett, 2003]) dans laquelle la mutation aléatoire est

remplacée par une procédure de recherche locale (cf. sous-section précédente).

À chaque génération, un ou plusieurs nouveaux individus (solutions) sont créés par un opérateur de croisement appliqué à plusieurs individus de la population, avant d'être améliorés par une procédure de recherche locale. Dans un algorithme mémétique, l'opérateur de croisement permet de créer de nouveaux points de départ pour la procédure de recherche locale, qui sont censés être plus prometteurs et meilleurs qu'une initialisation purement aléatoire.

Algorithme 2 Algorithme mémétique générique

Entrée: $G = (V, E)$ un graphe
Sortie: S^* la meilleure solution trouvée

$P \leftarrow$ une population d'individus initialisés aléatoirement ou avec un algorithme constructif.
$S^* \leftarrow$ le meilleur individu de P
tant que la condition d'arrêt n'est pas atteinte **faire**
    $(S_1, S_2) \leftarrow$ SÉLECTION(P)
    $S' \leftarrow$ CROISEMENT(G, S_1, S_2)
    $S' \leftarrow$ RECHERCHE LOCALE(G, S')
    si $f(S') < f(S^*)$ **alors**
        $S^* \leftarrow S'$
    $P \leftarrow$ INSERTION(P, S')
retourne S^*

L'Algorithme 2 présente un algorithme mémétique générique. Il commence par générer une population de solutions, de façon aléatoire, ou bien avec un algorithme constructif (cf. Section 2.2.1). Puis, à chaque itération, il sélectionne des individus de la population pour les croiser et obtenir un nouvel individu héritant de caractéristiques de ses parents. L'individu sera alors amélioré par une recherche locale et ajouté ou non à la population en maintenant, généralement, une taille de population constante. Plusieurs stratégies existent : remplacement des parents, individus les plus proches retirés, individu le moins jeune ou avec un score le plus faible retiré, etc. Si le nouvel individu est trop proche des solutions dans la population et de moins bonne qualité, il peut aussi ne pas être intégré dans la population.

Opérateurs de Croisement

En plus des procédures de recherche locale, des opérateurs de croisement (ou *crossover*) sont utilisés à chaque génération de l'algorithme mémétique pour combiner un ou plusieurs individus de la population afin d'en générer de nouveaux. Ces nouveaux individus descendants (ou *offspring*) seront ensuite améliorés par la procédure de recherche locale.

Un opérateur de croisement très populaire pour les problèmes de coloration est l'opérateur *GPX* introduit dans [Galinier et Hao, 1999] :

GPX (*Greedy Partition Crossover*, Algorithme 3) est un opérateur de croisement qui permet de diversifier la population en créant de nouvelles solutions à partir de deux parents. La solution créée hérite des groupes de couleurs les plus grands de ses deux parents, ce qui permet de garder une bonne qualité de solution tout en construisant un nouvel individu différent de ses parents.

[Moalic et Gondran, 2018] ont proposé différentes variantes de cet opérateur GPX en brisant la balance de 50% de couleurs entre les parents en favorisant l'un des parents. Cette méthode sera reprise dans le Chapitre 8.

Algorithme 3 Croisement GPX de [Galinier et Hao, 1999]

Entrée: $G = (V, E)$ un graphe, ; p_1 et p_2 deux individus de la population (colorations) ; k un nombre de couleurs.
Sortie: $child$ le croisement de p_1 et p_2

$child \leftarrow$ un individu vide
pour $c \in 1..k$ **faire**
    $p \leftarrow$ alternance entre p_1 et p_2
    $c* \leftarrow$ la couleur avec le plus de sommets dans p
    pour tout $v \in V^p_{c*}$ **faire**
        $child \leftarrow$ applique $c*$ à v pour $child$
        retire v de p_1 et p_2
appliquer une couleur aléatoire à $U(child)$
retourne $child$

Algorithmes Mémétiques pour le GCP

Le GCP a été abordé avec différents algorithmes mémétiques dans la littérature. Parmi les plus performants et connus, nous pourrons citer HEA (*Hybrid Evolutionary Algorithm*) [Galinier et Hao, 1999] qui a introduit l'opérateur de croisement *GPX* et HEAD (*HEA in Duet*) [Moalic et Gondran, 2018], qui utilise uniquement deux individus et les croise avec l'opérateur *GPX* avant de les améliorer avec une version optimisée de *TabuCol*. Ce *framework* HEAD sera repris dans le Chapitre 8 et présenté plus en détail.

Une autre algorithme mémétique, CISM (*Crossover by Independent Sets and Mutations*), est présenté dans [Dorne et Hao, 1998] où un croisement nommé UIS (*Union Of Independent Sets*) est introduit et conjointement utilisé avec *TabuCol*. Par ailleurs, [Lü et Hao, 2010] a aussi proposé un algorithme mémétique, nommé MACOL, qui met en jeu un croisement avec plusieurs

parents et un critère d'acceptation des solutions dépendant de la distance entre les solutions. La principale différence de ce nouveau croisement avec le croisement *GPX*, en dehors du nombre de parents pouvant être supérieur à deux, est le fait que le parent et le groupe de couleurs sont choisis de manière adaptive en maximisant la taille des groupes choisis. Le critère d'acceptation des solutions est défini de la manière suivante : soit $D_{i,P} = \min\{d_{ij}|S_j \in P, j \neq i\}$ la distance minimale entre la solution S_i et les solutions de la population P. Le score de chaque solution de la population lors de l'insertion est calculé avec la formule suivante :

$$h_{i,P} = f(S_i) + e^{\beta/D_{i,P}}$$

Où $f(S_i)$ est le score de la solution et β est un paramètre. Sachant que la valeur maximum de $D_{i,P}$ est $|V|$, $\beta = \lambda|V|$ où λ est réglé à 0.08 dans l'article d'origine. Ainsi plus λ est petit, plus les solutions sont acceptées facilement vis-à-vis de la distance. [Porumbel *et al.*, 2010a] propose également un algorithme mémétique contrôlant la diversité de la population avec un critère de distance et un croisement avec plusieurs parents. Ce croisement se base sur plusieurs critères, nombre de conflits dans les groupes de couleurs, nombre de sommets, somme des degrés et sélectionne les groupes de couleurs en fonction de ces critères. Le critère de distance consiste à utiliser un seuil de $10\%|V|$ à partir duquel les solutions sont considérées comme trop proches et ne sont pas acceptées dans la population, ou remplacent la solution la plus proche si le score est amélioré. Ce seuil est défini suite à une étude de la cartographie de l'espace de recherche dans [Porumbel *et al.*, 2010b] cité dans le chapitre précédent (cf. Section 1.4.2).

Algorithmes Mémétiques pour le WVCP

À notre connaissance, il existe à ce jour un seul algorithme mémétique pour le WVCP, l'algorithme *DLMCOL* que nous avons récemment proposé dans l'article [Goudet *et al.*, 2022]. Cet algorithme combine une recherche locale inspirée de l'algorithme *AFISA* avec un croisement de type *GPX*. Il présente la particularité d'être constitué d'une très grande population (plus de 20000 individus) et de calculer l'ensemble des recherches locales en parallèle sur une carte GPU (*Graphic Processing Units*). Il utilise des réseaux de neurones invariants par permutation des couleurs pour prédire quel croisement permettra d'obtenir les meilleures solutions après application d'une recherche locale. Cet algorithme n'est pas présenté dans le manuscrit, mais l'architecture générale du réseau de neurones sera reprise dans les Chapitres 7 et 8.

2.2.4 Autres Algorithmes à Population

Il existe de nombreux autres algorithmes à base de population pour la résolution de problèmes de coloration. Parmi les plus notables, AntCol de [Dowsland et Thompson, 2008] propose d'utiliser un algorithme de colonies de fourmis pour guider la recherche locale. [Titiloye et Crispin, 2011] ont développé un algorithme de recuit quantique (QACOL) à base de population qui met en œuvre un recuit simulé pour améliorer des solutions candidates. Il intègre des interactions entre les solutions avec un système d'attraction pour encourager les solutions à devenir similaires à leurs voisines et ainsi améliorer la transmission de bonnes propriétés dans les solutions au cours de la recherche. [Wu et Hao, 2012] ont proposé d'extraire en *préprocessing* des ensembles indépendants de sommets en utilisant un algorithme tabou et obtient de bons résultats sur de grandes instances du problème. [Goudet *et al.*, 2021] ont présenté un algorithme à base de population mettant en œuvre des techniques de descente de gradient pour apprendre des tenseurs de poids continus qui encodent les solutions de coloration. Pour plus de détails sur l'ensemble des algorithmes à base de population proposés dans la littérature, nous renvoyons le lecteur à l'article [Mostafaie *et al.*, 2020]. Les auteurs proposent une revue de 65 articles sur les métaheuristiques appliquées à la coloration de graphe (entre 2010 et 2018) divisées en 12 catégories (Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Bat Algorithm (BA), Cuckoo optimization algorithm (COA), Firefly algorithm (FA), Genetic algorithm (GA), Memetic algorithm (MA), Particle swarm Optimization (PSO), Quantum Annealing (QA), Tabu search (TS), Simulated Annealing (SA) et Hybrid Algorithms (HA)). Un grand nombre de ces articles utilisent des méthodes inspirées de différents phénomènes physiques ou biologiques pour justifier d'une innovation, cependant nous avons remarqué que leurs résultats ne sont pas toujours au niveau de l'état de l'art existant.

2.3 Méthodes Constructives et Apprentissage

Le domaine de l'intelligence artificielle nous a récemment apporté un grand nombre de méthodes d'apprentissage automatique. Ces méthodes permettent d'obtenir ce qui se rapproche le plus d'une expérience humaine et d'une certaine réactivité pour adapter la stratégie en fonction de l'environnement grâce à l'expérience.

Un exemple extrêmement populaire de ces dernières années est l'application de ces techniques d'apprentissage pour le jeu de go avec les algorithmes AlphaGo [Silver *et al.*, 2016], AlphaGo Zero [Silver *et al.*, 2017b] et AlphaZero [Silver *et al.*, 2017a] qui combinent l'algo-

rithme du MCTS (arbres de recherche de Monte Carlo) [Coulom, 2006] avec des réseaux de neurones pour atteindre un niveau de jeu équivalent, voire supérieur, à un joueur expert. Le MCTS est un algorithme de construction et d'exploration d'arbres de recherche. Il se divise en 4 phases : la sélection, l'expansion, la simulation et la mise à jour. La sélection guide le joueur dans l'arbre des coups possibles en partant de la racine pour atteindre un nœud non complètement ouvert en fonction d'une politique de sélection généralement basée sur un critère UCB [Lai *et al.*, 1985]. Ce critère permet de donner un score pour favoriser ou non l'exploration et l'exploitation. L'expansion permet d'agrandir l'arbre à partir du nœud sélectionné. Vient ensuite la simulation qui consiste à compléter la solution obtenue. À la fin, la mise à jour permet d'actualiser les valeurs de nombre de visites et de moyenne de score obtenue dans la branche explorée.

En 2011, [Previti *et al.*, 2011] ont proposé l'utilisation d'un MCTS pour le problème SAT. [Jooken *et al.*, 2023] ont récemment utilisé la méthode MCTS pour résoudre des problèmes d'optimisation combinatoire. Dans le Chapitre 5, nous proposerons un algorithme MCTS pour résoudre des problèmes de coloration et nous utiliserons la méthode de calcul de [Jooken *et al.*, 2023] lors de la phase de sélection des nouveaux nœuds à ouvrir.

Récemment, [Cazenave *et al.*, 2021] ont proposé d'aborder la résolution de problèmes de coloration de graphe avec deux méthodes de type MCTS. Un NMCS (*Nested Monte Carlo Search* [Cazenave, 2009]) et un NRPA (*Nested Rollout Policy Adaptation* [Cazenave, 2017]). Ces deux méthodes échantillonnent un très grand nombre de solutions grâce à des branchements aléatoires, pour améliorer continuellement la qualité des politiques de sélection. La particularité de NRPA est d'utiliser une politique stochastique avec de l'apprentissage basé sur une descente de gradient pour apprendre pendant la recherche à trouver de bonnes solutions plus rapidement.

[Huang *et al.*, 2019] ont tenté d'appliquer directement les principes d'AlphaGo Zero à la coloration de graphe, vu sous la forme d'un jeu à un seul joueur. De l'apprentissage par renforcement et un réseau avec graphe embarqué pour construire une solution de bonne qualité pour plusieurs problèmes d'optimisations a aussi été proposé par [Khalil *et al.*, 2017]. De façon plus large, [Li *et al.*, 2018] ont utilisé un réseau de neurones convolutif adapté aux graphes pour résoudre des problèmes NP-difficiles.

Les méthodes utilisant uniquement de l'apprentissage ont souvent besoin d'améliorer le côté intensification de l'algorithme que l'on peut obtenir facilement avec des méthodes approchées. C'est pourquoi des méthodes combinant des métaheuristiques avec de l'apprentissage ont aussi été proposées dans la littérature, mais ce champ de recherche reste encore largement à explorer.

2.4 Métaheuristiques, Apprentissage et Hyperheuristiques

Les hybridations d'algorithmes sont très populaires dans le domaine des métaheuristiques. Elles permettent de combiner les avantages de plusieurs algorithmes pour renforcer les performances de ceux-ci. Par exemple, l'algorithme mémétique provient de l'hybridation d'un algorithme génétique et d'une recherche locale. Depuis maintenant de nombreuses années, les hybridations se font aussi avec des méthodes d'apprentissage automatique.

2.4.1 Métaheuristiques Guidées par l'Apprentissage

[Zhou *et al.*, 2016] et son amélioration [Zhou *et al.*, 2018] utilisent de l'apprentissage par renforcement pour guider la coloration d'un graphe. D'autres méthodes explorent l'hybridation entre le *data mining* et l'optimisation combinatoire comme [Zhou *et al.*, 2020] qui utilise de la reconnaissance de motifs dans un algorithme mémétique. [Elhag et Özcan, 2015] ont testé de nombreuses hyperheuristiques pour le problème de coloration de graphe. Nous retiendrons celle de [Mısır *et al.*, 2013] mettant en œuvre de l'apprentissage par renforcement. [Sabar et Kendall, 2015, Asta *et al.*, 2016] proposent l'utilisation d'hyperheuristiques dans le cadre d'un MCTS pour différents problèmes d'optimisation combinatoire. Plus récemment, l'algorithme DLMCOL [Goudet *et al.*, 2022] déjà évoqué dans la Section 2.2.3 combine un algorithme mémétique classique, mais guidé par de l'apprentissage avec un réseau de neurones pour le choix des appariements au sein de la population.

2.4.2 Hyperheuristiques

Lorsque des méthodes d'apprentissage sont utilisées non pas pour guider une heuristique, mais pour générer de nouvelles métaheuristiques, on parle alors d'hyperheuristiques.

[Burke *et al.*, 2010a] définit les hyperheuristiques comme "une méthode de recherche ou un mécanisme d'apprentissage pour sélectionner ou générer des heuristiques pour résoudre des problèmes de recherche computationnelle" (en anglais dans le texte). Par exemple, parmi un ensemble de mutations pour un algorithme génétique, l'hyperheuristique va choisir la mutation la plus adaptée pendant la recherche tout en explorant pour donner une chance aux autres opérateurs qui pourraient être plus adaptés en fonction de l'état des solutions dans la population. [Burke *et al.*, 2013] divise les hyperheuristiques en deux catégories : les hyperheuristiques de sélection et les hyperheuristiques de génération. D'un côté, les hyperheuristiques de sélection sélectionnent une heuristique parmi un ensemble d'heuristiques prédéfinies. De l'autre,

les hyperheuristiques de génération génèrent une heuristique à partir d'un ensemble de composants existants. Certaines de ces hyperheuristiques sont de type *online*, c'est-à-dire qu'elles apprennent continuellement pendant la recherche en fonction des résultats des différents opérateurs choisis. D'autres sont de type *offline*, c'est-à-dire qu'elles n'apprennent pas au cours de la résolution d'une instance, mais sont préentraînées en amont, notamment pour choisir le bon algorithme de résolution à utiliser en fonction des caractéristiques d'une instance rencontrée.

[Drake *et al.*, 2020] a récemment proposé une revue plus récente des hyperheuristiques de sélection. Il étend la taxinomie avec plus de notions et une mise à jour qui prend en compte les découvertes les plus récentes dans ce domaine.

Dans le domaine des hyperheuristiques appliquées à la coloration de graphe, on peut citer [Sabar *et al.*, 2012]. Ils proposent l'utilisation d'hyperheuristiques pour le problème de génération d'emploi du temps avec une sélection par roulette (cf. Chapitre 7) pour choisir la bonne heuristique de construction.

Les hyperheuristiques utilisent différentes stratégies de sélection pour prendre leurs décisions. De nombreux critères de sélection ont été proposés dans la littérature [Drake *et al.*, 2020]. Pour comparer notre nouvelle approche présentée dans le Chapitre 7, nous utiliserons trois de ces critères largement utilisés dans la littérature des hyperheuristiques [Thierens, 2005, Hoos, 2012, Goëffon *et al.*, 2016] : deux critères basés sur un biais proportionnel (*Roulette* et *Pursuit*) et un critère basé sur la méthode de bandit manchot *UCB*.

2.5 Bilan de l'État de l'Art

Pour résumer, nous avons regroupé dans le Tableau 2.1 les méthodes que nous estimons être les plus proches des travaux que nous allons présenter dans ce travail de thèse.

Au début de la thèse, fin 2020, les méthodes constructives avec de l'apprentissage étaient peu abordées pour les problèmes de coloration, en particulier pour le WVCP. Poussés par l'idée que les sommets les plus lourds doivent être placés en premier pour ce problème, nous avons considéré qu'une approche séquentielle d'affectation de couleurs de type MCTS pourrait être intéressante. Après de nombreuses expériences et l'hybridation avec des recherches locales (dont certaines ont été proposées par d'autres équipes pendant ma thèse) [Grelier *et al.*, 2022], nous avons exploré l'idée d'incorporer des hyperheuristiques lors du choix de la recherche locale à appliquer pour une instance donnée en apprenant pendant la recherche quel opérateur de recherche locale choisir [Grelier *et al.*, 2023]. Le choix pouvant être influencé par l'état de la solution, nous avons exploré l'hypothèse qu'un réseau de neurones pourrait être capable de

sélectionner un opérateur en fonction de la solution courante rencontrée.

Par ailleurs, les algorithmes mémétiques étant très performants pour la coloration de graphe, [Goudet *et al.*, 2022], mais souvent peu flexibles du point de vue du choix de leurs opérateurs, nous avons aussi cherché dans cette thèse à appliquer des techniques venant du domaine des hyperheuristiques pour sélectionner dynamiquement des opérateurs de croisement et de recherche locale à chaque génération d'un algorithme mémétique. À notre connaissance, cette voie a été jusqu'à présent très peu explorée dans la littérature sur les problèmes de coloration. En parallèle, nous nous sommes aussi penchés sur la programmation par contrainte, la réduction et le calcul de bornes pour le problème du WVCP [Goudet *et al.*, 2023] dans l'objectif d'explorer une hybridation avec des métaheuristiques.

nom de l'algorithme [référence]	détails
	GCP
TabuCol [Hertz et Werra, 1987]	LS, tabou, espace illégal, *one-move*
CISM [Dorne et Hao, 1998]	MA, croisement UIS, *TabuCol*
HEA [Galinier et Hao, 1999]	MA, croisement *GPX*, *TabuCol*
PartialCol [Blöchliger et Zufferey, 2008]	LS, tabou, espace partiellement légal, *grenade*
AntCol [Dowsland et Thompson, 2008]	Colonie de fourmis, utilisation de *TabuCol grenade*
Evo-Div [Porumbel *et al.*, 2010a]	MA, croisement sur plusieurs parents avec critères, gestion des distances entre individus
MACOL [Lü et Hao, 2010]	MA, croisement sur plusieurs parents, distances entre individus pour l'insertion
QACOL [Titiloye et Crispin, 2011]	Algorithme à population, recuit quantique, interactions entre solutions
EXTRACOL [Wu et Hao, 2012]	MA, extraction d'ensembles indépendants
HEAD [Moalic et Gondran, 2018]	MA, variations du croisement *GPX*, *TabuCol* optimisé, 2 individus, solutions élites
PLSCOL [Zhou *et al.*, 2018]	Apprentissage par renforcement
[Huang *et al.*, 2019]	AlphaGo Zero appliqué à la coloration de graphe
[Zhou *et al.*, 2020]	Reconnaissance de patterns
TensCol [Goudet *et al.*, 2021]	MA, descente de gradient et tenseurs
NRPA [Cazenave *et al.*, 2021]	MCTS, politique stochastique avec apprentissage et descente de gradient
	WVCP
R-GRASP [Prais et Ribeiro, 2000]	Reactive GRAPS, paramètre adaptatif sur le choix des sommets à colorer
AFISA [Sun *et al.*, 2018]	Tabou, espace illégal, *one-move*, coefficient adaptatif, perturbations
RedLS [Wang *et al.*, 2020]	LS, *configuration checking*, espace illégal, voisinages *one-move*, poids sur les arêtes, perturbations
ILS-TS [Nogueira *et al.*, 2021]	LS, tabou, espace partiellement légal, voisinages *one-move* et *grenade*, perturbations
DLMCOL [Goudet *et al.*, 2022]	MA, +20000 individus, GPU, *AFISA*, *GPX*, sélection d'individus avec réseau de neurones

TABLE 2.1 – Tableau résumant les algorithmes populaires proches ou utilisés dans nos travaux.

CHAPITRE 3

RÉDUCTION

Ce chapitre présente une procédure itérative de réduction de la taille des instances en supprimant des sommets du graphe. Nous proposons l'amélioration d'une règle de réduction de l'état de l'art et adaptons une règle de réduction du GCP au WVCP. Nos règles de réduction sont ensuite testées sur toutes les instances de benchmark.

Le travail présenté est publié dans l'article [Goudet *et al.*, 2023] pour IJCAI 2023.

3.1 Introduction

Pour faciliter la résolution d'un problème d'optimisation combinatoire, diminuer le nombre de variables sur une instance est un moyen efficace de réduire la taille de l'espace de recherche. Pour la coloration de graphe, il est possible de supprimer des sommets du graphe sans modifier la solution optimale [Cheeseman *et al.*, 1991].

Le GCP a connu de nombreuses tentatives de réduction de la taille du problème en supprimant des sommets, par exemple, [Cai, 2003, Jansen et Kratsch, 2013] pour des familles particulières de graphes. [Cheeseman *et al.*, 1991] résument 3 règles de réduction pour le *k-coloring* :

1. Un sommet peut être supprimé si son degré est inférieur à k, car il y aura toujours une couleur disponible pour lui.
2. Un sommet u peut être supprimé si tous ses voisins sont connectés à un sommet v, car u pourra prendre la couleur de v sans ajouter de contraintes à la solution.
3. Tout sommet connecté à une clique de taille $k-1$ peut être fusionné en un seul sommet, car ils doivent partager la même couleur à la fin.

La première règle a été adaptée dans [Lin *et al.*, 2017] pour supprimer de grands ensembles indépendants et dans [Wang *et al.*, 2020] pour le WVCP.

Nous proposons une amélioration de cette première règle pour le WVCP dans ce chapitre. Nous adaptons aussi la deuxième règle au WVCP : le premier nœud n'est supprimé que si son poids est inférieur ou égal au deuxième nœud. La troisième règle étant intrinsèquement liée au

nombre maximum de couleurs à utiliser, elle ne peut être appliquée au WVCP, car le nombre de couleurs à utiliser est inconnu.

Ce chapitre est organisé comme suit. La Sous-Section 3.1.1 rappelle les notations pour le problème. La Section 3.2 présente la règle R0 de [Wang *et al.*, 2020]. La Section 3.3 présente notre amélioration de R0 avec la règle R1. La Section 3.4 propose l'adaptation de la seconde règle de [Cheeseman *et al.*, 1991] avec notre règle R2. La Section 3.5 présente la procédure itérative de réduction. Enfin, les expérimentations sur différentes instances du GCP et WVCP sont présentées dans la Section 3.6.

3.1.1 Définitions et Notations

Soit $G = (V, E, w)$ un graphe pondéré non orienté où $V = \{v_1, \ldots, v_n\}$ est un ensemble de n sommets, $E = \{e_1, \ldots, e_p\}$ est un ensemble de p arêtes non orientées et w est une fonction $V \to \mathbb{N}^*$ qui associe à chaque sommet de V un poids strictement positif $w(v) \in \mathbb{N}^*$. Notons $N(v) = \{u \in V, (u, v) \in E\}$ l'ensemble des voisins du sommet v dans $G = (V, E, w)$ et $\Delta = \max_{v \in V} |N(v)|$ le degré maximal du graphe G. Une clique C dans G est un sous-ensemble de sommets de V tel que $\forall u, v \in C, (u, v) \in E$. Nous admettrons que les sommets d'une clique $C = \{c_1, \ldots, c_{|C|}\}$ sont triés par ordre décroissant de poids : $(w(c_i) \geq w(c_j)$ for $1 \leq i < j \leq |C|)$. Une solution S de coloration (légale) est une partition $\{V_1, \ldots, V_k\}$ de V en k ensembles indépendants de sommets, c'est-à-dire qu'aucune paire de sommets de chaque couleur V_i n'est connectée dans G. Un ensemble indépendant V_i est un sous-ensemble non vide de V tel que tous les sommets de V_i ne sont connectés par aucune arête. L'objectif pour le GCP et le WVCP est de trouver une coloration $S = \{V_1, \ldots, V_k\}$ dont le score $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$ est minimisé. On rappelle que le problème GCP correspond au cas où tous les poids des sommets sont à 1. Les règles et procédures de réduction que nous présenterons dans ce chapitre sont conçues pour le WVCP, mais peuvent donc aussi naturellement s'appliquer au GCP.

3.2 Règle R0

Une première procédure de réduction pour le WVCP a été proposée par [Wang *et al.*, 2020]. Cette procédure, notée par la suite R0, consiste en une adaptation pour le WVCP de la première règle de [Cheeseman *et al.*, 1991] (cf. Section 3.1). Elle consiste à extraire un ensemble de cliques $Cliset = \{C_1, \ldots, C_t\}$, puis à appliquer la règle atomique R0 définie ci-dessous pour tout couple de sommets v et clique $C \in Cliset$.

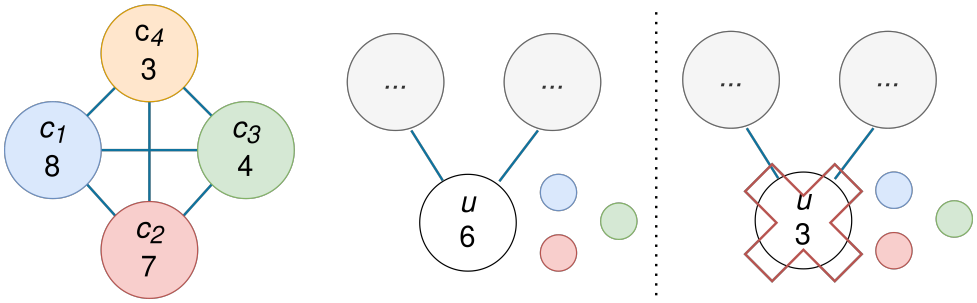


FIGURE 3.1 – Application de la règle R0 à un sommet u dans deux cas avec la présence d'une clique $C = \{c_1, c_2, c_3, c_4\}$. À gauche, u est de poids 6 et possède 2 voisins, dans le pire cas, ses 2 voisins prendront la couleur des sommets les plus lourds de la clique c_1 et $c2$, il ne pourra alors pas être supprimé, car u est plus lourd que c_3 le 3e sommet le plus lourd de la clique. À droite, u est de poids 3, cette fois il pourra être supprimé, car même si ses voisins prennent la couleur de c_1 et c_2, il pourra prendre la couleur de c_3 sans affecter le score. Gauche : u ne peut pas être supprimé. Droite : u peut être supprimé.

R0 : Considérant un graphe $G = (V, E, w)$, une clique C de G, $v \in V \setminus C$ et $|N(v)|$, le degré de v. Si $|N(v)| + 1 \leq |C|$ et $w(v) \leq w(c_{|N(v)|+1})$, alors supprimer v n'a pas d'impact sur le score optimal de G.

La Figure 3.1 illustre l'application de la règle R0 à un sommet u dans deux cas avec la présence d'une clique $C = \{c_1, c_2, c_3, c_4\}$. Le sommet u ne peut être supprimé que dans le cas où son poids est inférieur au 3e (car il possède 2 voisins) sommet le plus lourd de la clique.

3.3 Règle R1

Nous proposons R1, une amélioration de cette règle R0 en prenant en compte le fait que v peut avoir des voisins dans C en utilisant la procédure POSICLIQUE.

Algorithme 4 POSICLIQUE

Entrée: $G = (V, E, w)$ un graphe, v un sommet de V et une clique $C = \{c_1, \ldots, c_{|C|}\}$ **s.t.** $w(c_i) \geq w(c_{i+1})$ pour tout $i \in \{1, \ldots, |C| - 1\}$
Sortie: $d \in \mathbb{N}^*$ le degré de v relatif à C dans le contexte de la réduction

1: $d \leftarrow |N(v)| + 1$
2: $l_C \leftarrow |C| - 1$
3: **pour** i **de** 0 **à** l_C **faire**
4: **si** $c_{|C|-i} \in N(v)$ **et** $|C| - i < d$ **alors**
5: $d \leftarrow d - 1$
6: **retourne** d

R1 : Considérant un graphe $G = (V, E, w)$, une clique C de G, $v \in V \setminus C$ et le degré de v relatif à la clique $d =$PosiClique(v, C). Si $d \leq |C|$ et $w(v) \leq w(c_d)$, alors supprimer v n'a pas d'impact sur le score optimal de G.

Démonstration. Soit S' une coloration optimale pour le WVCP de $G\backslash\{v\}$. Les sommets de la clique C sont tous dans S' (car $v \notin C$), et sont colorés avec $|C|$ couleurs différentes $k_1, k_2, \ldots, k_{|C|}$.

Utilisons la procédure PosiClique définie ci-dessus pour calculer $d =$ PosiClique(v, C) et montrons que l'affirmation suivante $H(d)$ est vraie : "il y a au plus $d-1$ voisins de v prenant une couleur dans $\{k_1, k_2, \ldots, k_d\}$".

Initialisons une variable d à la valeur $|N(v)| + 1$ comme cela est fait dans la procédure PosiClique. Pour cette valeur de d, l'affirmation $H(d)$ est vraie, car il y a au plus $|N(v)|$ voisins de v prenant une couleur dans l'ensemble $\{k_1, k_2, \ldots, k_{|N(v)|+1}\}$.

Maintenant, lorsque nous appliquons la procédure PosiClique, pour tout i de 0 à $|C| - 1$, si $c_{|C|-i} \in N(v)$, $|C| - i \geq d$ et si l'affirmation $H(d)$ est vraie, alors nous savons qu'un voisin de v, à savoir $c_{|C|-i}$, a reçu une couleur qui n'est pas dans l'ensemble $\{k_1, k_2, \ldots, k_{d-1}\}$, donc l'affirmation $H(d-1)$ est vraie, et pour l'itération suivante d est remplacé par la valeur $d-1$.

Quand la procédure itérative PosiClique se termine, elle retourne une valeur d, et nous savons que l'affirmation $H(d)$ est vraie pour cette valeur d, c'est-à-dire qu'il y a au plus $d-1$ voisins de v prenant une couleur dans l'ensemble $\{k_1, k_2, \ldots, k_d\}$. Par conséquent, S' peut être étendu en une coloration optimale et légale S^* du WVCP du graphe original G, en attribuant une couleur dans $\{k_1, \ldots, k_d\}$ au sommet v.

Si $d \leq |C|$, cette opération peut toujours être effectuée sans augmenter le coût optimal, car $w(v) \leq w(c_d)$, et donc $w(v) \leq w(c_i)$ pour $i = 1, \ldots, d$, car nous avons $w(c_1) \geq w(c_2) \geq \cdots \geq w(c_q)$. □

La règle R1 est équivalente à la règle R0 lorsque $N(v) \cap C = \emptyset$. Dans ce cas, $d = |N(v)|+1$ et v est supprimé si $d \leq |C|$ et $w(v) \leq w(c_d)$. Cependant, notre règle est plus forte lorsque $N(v) \cap C \neq \emptyset$, car la valeur d retournée par PosiClique peut être inférieure à $|N(v)| + 1$ ce qui améliore la chance d'un test réussi $w(v) \leq w(c_d)$ en raison de l'ordre décroissant basé sur le poids des sommets des cliques.

La complexité au pire cas de la règle R1 est $O(\Delta^2)$.

La Figure 3.2 illustre le rôle de la procédure PosiClique pour la règle R1. Elle représente deux graphes où une clique $C = \{c_1, c_2, c_3, c_4\}$ et un sommet u de poids 6 sont considérés pour une évaluation de R1. Dans les deux cas, $|N(u)| = 2$. À gauche, $c_2 \in N(u)$. Ainsi, étant

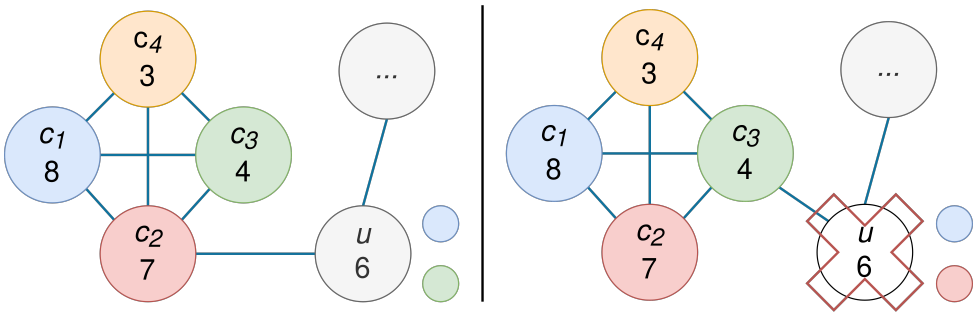


FIGURE 3.2 – Application de la règle R1 à un sommet u de poids 6 et une clique $C = \{c_1, c_2, c_3, c_4\}$ dans deux cas différents. Gauche : u ne peut pas être supprimé. Droite : u peut être supprimé.

donné u et C en entrée, POSICLIQUE retourne $d = 3$ et u ne peut pas être supprimé, car il peut prendre la même couleur que c_3 dans le pire des cas et son poids est supérieur à $w(c_3) = 4$. À droite, $c_3 \in N(u)$. Dans ce cas, POSICLIQUE retourne $d = 2$ et u peut être supprimé, car il peut prendre la couleur bleue ou rouge, et dans les deux cas, il n'y a pas d'impact sur le score WVCP, car son poids de 6 est inférieur à $w(c_1) = 8$ et $w(c_2) = 7$.

3.4 Règle R2

[Cheeseman *et al.*, 1991] liste des réductions applicables au GCP. Par exemple, pour la k-coloration, tout sommet de degré inférieur à k peut être supprimé, car il existera toujours une couleur disponible que ses voisins n'auront pas prise. Cette règle ne peut être appliquée dans notre cas, car le nombre de couleurs n'est pas limité dans le WVCP. Une autre règle est la suivante : un sommet v peut être supprimé s'il existe un sommet u tel que $N(v) \subset N(u)$, car toute coloration de u est aussi une coloration de v (si u et v ne sont pas voisins). Cette règle peut être appliquée dans le cas du WVCP en considérant le poids des sommets :

R2 : Considérant un graphe $G = (V, E, w)$, deux sommets $u, v \in V$, tels que $N(u) \subset N(v)$ et $w(u) \leq w(v)$, alors supprimer u n'a pas d'impact sur le score optimal de G.

Démonstration. Soit deux sommets $u, v \in V$, tels que $N(u) \subset N(v)$ et $w(u) \leq w(v)$. Supposons que S' est une coloration optimale du WVCP du sous-graphe de G induit par $V \backslash \{u\}$. Comme $N(u) \subset N(v)$, S' peut être étendu à une coloration optimale S^* du WVCP du graphe original G, en plaçant le sommet u dans l'ensemble indépendant contenant v, sans augmenter le score, car $w(u) \leq w(v)$. □

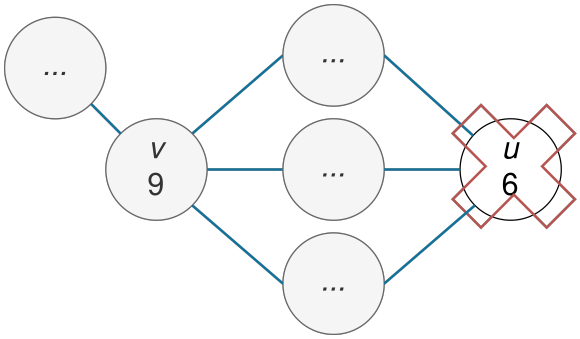


FIGURE 3.3 – Application de la règle R2 à un sommet u de poids 6 et un sommet v de poids 9. Comme tous les voisins de u sont aussi voisins de v, u peut être supprimé.

La complexité de R2 au pire cas est $O(\Delta^2)$.

La Figure 3.3 illustre la règle R2. Le voisinage du sommet u étant complètement inclus dans le voisinage du sommet v, u peut être supprimé sans impact sur le score optimal, car il pourra dans tous les cas prendre la couleur de v.

3.5 Procédure Itérative et Extraction de Cliques

R1 considère la suppression d'un sommet relativement à une seule clique. En examinant cette règle de réduction, nous observons d'abord qu'elle fonctionne si $u \notin C$, et qu'elle est plus susceptible d'être appliquée, (i) si les poids dans C prennent de grandes valeurs, car la règle nécessite que $w(u)$ soit inférieur à $w(c_d)$ avec certains $c_d \in C$, mais aussi (ii) si u est connecté à de nombreux sommets $c \in C$. Nous proposons une procédure itérative de réduction qui considère la suppression d'un sommet relativement à plusieurs cliques. Idéalement, chaque clique C devrait avoir un poids maximal $\sum_{c \in C} w(c)$. Cependant, trouver une clique de poids maximal dans un graphe est NP-difficile. Nous utilisons donc une heuristique rapide pour effectuer cette tâche, à savoir FastWClq [Cai et Lin, 2016] décrit dans l'Algorithme 5.

FastWClq construit itérativement une clique avec des mouvements gloutons. Il utilise n'importe quel sommet comme point de départ. Puis, il ajoute un à un les sommets les plus lourds qui permettront de potentiellement ajouter le plus d'autres sommets lourds à l'itération suivante. Il se base pour cela sur la formule ligne 9 de l'Algorithme 5 qui calcule un bénéfice potentiel pour chaque sommet. Pour garder un temps de calcul raisonnable tout en gardant de la diversité parmi les cliques, nous calculons $|V|$ cliques en générant une seule clique par sommet $v \in V$ en utilisant v comme point de départ pour FastWClq.

Cette procédure est en $O(|V|^3)$.

Algorithme 5 FastWClq de [Cai et Lin, 2016]

Entrée: $G = (V, E, w)$ un graphe, v un sommet de départ
Sortie: C une clique contenant v

$C \leftarrow \{v\}$
$candidates \leftarrow N(v)$
tant que $candidates \neq \emptyset$ **faire**
    $best\_vertex \leftarrow -1$
    $best\_benefit \leftarrow -1$
    pour tout $n \in candidates$ **faire**
        $commun\_neighbors \leftarrow candidates \cap N(n)$
        $potential\_weight \leftarrow \sum_{cn}^{commun\_neighbors} w(cn)$
        $benefit \leftarrow w(n) + (potential\_weight/2)$
        si $benefit > best\_benefit$ **alors**
            $best\_benefit \leftarrow benefit$
            $best\_vertex \leftarrow n$
    $C \leftarrow C \cup \{best\_vertex\}$
    $candidates \leftarrow candidates \setminus n$
    $candidates \leftarrow candidates \cap N(n)$
retourne C

Algorithme 6 Procédure de réduction itérée

Entrée: $G = (V, E, w)$ un graphe, $\mathcal{C} = \{C_1, \ldots, C_l\}$ un ensemble de cliques
Sortie: $G' = (V', E', w')$ le graphe réduit

$L \leftarrow []$
$G' \leftarrow G$
répéter
    pour tout $v \in V \setminus L$ **faire**
        // R1
        si $\exists C \in \mathcal{C}$ tel que $|N'(v)| \leq |C|$ et $w(v) \leq w(c_d)$ où $d =$ PosiClique(G', v, C)
        // R2
        ou $\exists u \in N(v)$ tel que $N(v) \subset N(u), w(v) \leq w(u)$
        alors suppression de v dans G' et dans les cliques de $\mathcal{C}$ et $L \leftarrow L \cup \{v\}$
tant que plus aucun sommet n'est supprimé
retourne G'

L'Algorithme 6 présente la procédure de réduction. Une fois les cliques générées, les deux règles de réduction sont appliquées à chaque sommet par ordre décroissant de poids. Si un sommet est supprimé, il est ajouté à une liste L, puis le graphe et l'ensemble des cliques sont mis à jour pour prendre en compte la suppression. La procédure est répétée jusqu'à ce qu'aucun sommet ne puisse être supprimé. Dans le pire des cas, $|V| - 1$ itérations peuvent être effectuées. En pratique, pour les instances considérées dans ce travail, au plus 10 itérations doivent être effectuées jusqu'à ce qu'aucun sommet ne puisse être supprimé. La procédure itérative de réduction complète ne prend pas plus de quelques secondes à calculer pour la majorité des graphes (voir la Section 3.6).

Cette procédure itérative est dans le pire cas en $O(\Delta^2|V|^3)$.

Lorsqu'une solution est trouvée pour le graphe réduit, il est possible de retrouver une solution pour le graphe original en coloriant les sommets de la liste L avec un algorithme glouton dans l'ordre inverse de leur arrivée dans L.

3.6 Expérimentations

Nous présentons d'abord les instances de benchmark utilisées pour les expériences et analysons l'impact de nos procédures de réduction sur la taille des instances.

Instances de benchmark. Au total, 188 instances de WVCP ont été considérées : 30 graphes rxx et 35 graphes pxx [Prais et Ribeiro, 2000] provenant de problèmes de décomposition de matrices et 123 graphes provenant des compétitions DIMACS et COLOR [Sun *et al.*, 2018]. Pour le GCP, l'ensemble des instances DIMACS et COLOR sont utilisées ainsi que les instances spécifiques au WVCP. 244 instances sont donc disponibles. Dans cette section, les résultats de la réduction sont présentés pour un sous-ensemble d'une quinzaine d'instances présentant des caractéristiques variées (tailles, densités, distributions des poids). Pour accéder aux résultats sur l'ensemble des instances, nous référons le lecteurs à des tableaux complets disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_instances`.

Paramètres expérimentaux. Les expériences ont été réalisées sur un Intel Core i7-10875H 2.30GHz. Toutes les règles de réduction, ainsi que l'heuristique FastWClq, [Cai et Lin, 2016] sont implémentées en Python 3. Le code, les instances et les résultats complets sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_instances`.

3.6.1 Réduction pour le WVCP

Le Tableau 3.1 présente l'impact des différentes règles de réduction sur l'ensemble des 188 instances de benchmark du WVCP. Pour R0 et R1, nous avons précalculé un ensemble de cliques $\mathcal{C}$ comme discuté dans la Section 3.5 puis appliqué chaque règle individuellement à chaque paire (v, c) avec $v \in V$ et $c \in \mathcal{C}$ (lignes R0 et R1). Pour R2, nous avons appliqué la règle à chaque paire de sommets non adjacents. (R1 + R2) correspond à l'application conjointe des deux règles de réduction. Enfin, *Iterative* correspond à la procédure de réduction itérative (voir Section 3.5) qui applique R1 et R2 jusqu'à ce qu'aucun sommet ne puisse être supprimé.

Nous voyons que R1 fonctionne légèrement mieux que R0 en termes de nombre d'instances réduites (# RI) et de pourcentage moyen de réduction (% RV) pour ces instances réduites. Dans l'ensemble, nous observons une grande amélioration due à la procédure itérative. Le temps de calcul des différentes réductions reste raisonnable en moyenne par rapport à la règle R0 (colonne 7). La moyenne ne considère que les instances réduites par la règle étudiée.

Nous observons que l'application de chaque règle améliore les chances de succès de la suivante. Cela est dû au fait que lorsque les sommets sont supprimés, cela réduit le degré de leurs voisins et leur donne plus de chances d'être supprimés lors du prochain passage de la procédure de réduction itérative. L'ajout de R1 et R2 nous permet de réduire 3 instances supplémentaires. Au total, les nouvelles règles couplées à la procédure itérative réduisent presque deux fois plus les instances en moyenne. Les instances les plus notables sont p29/31, GEOM20a/20/30/40a/40/50, inithx.i.1/2/3, miles250/500, zeroin.i.1/2/3 et DSJR500.1 pour lesquelles plus de 50% de l'instance est réduite.

Le temps de réduction est inférieur à 10 secondes pour 175 instances pour R1 et 171 instances pour *Iterated*, entre 1 et 11 minutes pour 3 instances pour R1 et 8 instances pour *Iterated*. Le temps le plus long apparaît pour l'instance C2000.9 qui prend presque 1h30 pour R1 et 6 minutes de plus pour *iterated*.

	$\# RI$	$\# RV$ avg	$\# RV$ max	$\%RV$ avg	$\%RV$ max	t(s) avg
R0	82	34.2	469	13.4	65	2.6
R1	84	39.5	574	14.7	66.4	3.8
R1+R2	85	41.7	596	15.4	69	4.1
Iterative	85	54.3	683	23.3	80.9	9.8

TABLE 3.1 – Impact des procédures de réduction pour le WVCP. $\# RI$: nombre d'instances réduites (sur 188), $\# RV$: nombre de sommets réduits, $\%RV$: pourcentage de réduction et temps $t(s)$ en secondes.

instance	$\|V\|$	density	R0	R1	R1+R2	Iterated	time(s)
DSJC125.1g	125	0.1	0	0	0	0	0.03
DSJC125.5g	125	0.5	0	0	0	0	0.26
DSJC125.9g	125	0.9	0	0	0	0	3.22
DSJR500.1	500	0.03	78	80	80	**256**	1.32
GEOM110	110	0.11	6	9	9	**23**	0.09
inithx.i.1	864	0.05	469	574	596	**683**	19.45
le450_15a	450	0.08	28	28	28	**30**	1.38
le450_25b	450	0.08	90	90	90	**105**	2.26
mulsol.i.5	186	0.23	28	53	75	**82**	1.16
queen10_10	100	0.59	0	0	0	0	0.08
p42	138	0.12	1	1	1	**3**	0.1
r30	301	0.09	0	0	0	0	0.48

TABLE 3.2 – Nombre de sommets supprimés par les règles de réduction pour des instances du WVCP. Les nombres en gras indiquent que plus de sommets ont été supprimés par rapport à la méthode R0.

Le Tableau 3.2 présente les résultats des règles de réduction proposées dans ce travail et une comparaison avec la règle de réduction existante R0 proposée dans [Wang *et al.*, 2020].

La colonne 1 montre le nom de l'instance. Les colonnes 2 et 3 correspondent respectivement au nombre de sommets $|V|$ et à la densité d'arêtes avant l'application d'une règle de réduction. La colonne 4 (R0) correspond à la règle de réduction proposée dans [Wang *et al.*, 2020]. La colonne 5 (R1) correspond à la version améliorée de R0 proposée dans la Section 3.3. Les méthodes R0 et R1 partent du même ensemble de cliques obtenues avec l'heuristique FastWClq comme détaillé dans la Section 3.5 mais ne réalisent pas la réduction itérative. Chaque sommet est considéré une fois par ordre décroissant de son poids. La colonne 6 (R1 + R2) correspond à la combinaison des règles de réduction R1 et R2 proposées dans les Sections 3.3 et 3.4, sans réduction itérative. La colonne 7 (*Iterated*) affiche la procédure de réduction itérative globale détaillée dans la Section 3.5 consistant à appliquer successivement les règles de réduction R1 et R2 jusqu'à ce qu'aucun sommet ne puisse être supprimé. La colonne 8 rapporte le temps global en secondes nécessaire à cette réduction itérative.

Nous observons d'abord que pour les instances DJSC, aucun sommet n'est supprimé pendant la procédure de réduction. Cela est dû au fait que les connexions entre les sommets sont purement aléatoires dans ces instances. Ainsi, la distribution des degrés est très homogène, de sorte que la règle R1 n'a presque aucune chance d'être appliquée. Cependant, nous observons qu'il y a beaucoup plus de sommets supprimés pour l'instance DSJR500.1, qui est une instance géométrique avec une distribution de degrés hétérogènes. Ainsi, dans cette instance, il y a de

grandes cliques, mais aussi de nombreux sommets avec un faible degré, ce qui les rend susceptibles d'être supprimés avec les règles R1 et R2. Nous observons que notre règle de réduction améliorée R1 fonctionne mieux que la règle de réduction R0 de [Wang *et al.*, 2020] pour cette instance, mais aussi pour d'autres telles que inithx.i.1 et mulsol.i.5.

Cela met en évidence que prendre en compte la connexion entre le sommet à supprimer et chaque clique considérée peut être bénéfique. Pour cette instance DSJR500.1, nous observons une grande amélioration due à la procédure itérative (colonne 7) lorsque le nombre de sommets supprimés passe de 80 à 256. Cela est dû au fait que lorsque les premiers sommets sont supprimés, cela réduit le degré de leurs voisins et leurs donne plus de chances d'être supprimés lors du prochain passage de la procédure de réduction itérative. Nous observons dans la colonne 8 que le temps de calcul nécessaire pour effectuer l'extraction de cliques et la procédure de réduction itérative reste raisonnable même pour les plus grandes instances.

Nous ne considérerons que les instances réduites dans les expériences décrites dans le reste de la thèse. Notez que l'utilisation des instances réduites permet d'améliorer considérablement les résultats par rapport à l'utilisation des instances originales (cf. annexe technique (*technical appendix*) de l'article [Goudet *et al.*, 2023] disponible à l'adresse suivante `https://github.com/Cyril-Grelier/gc_wvcp_cp`).

3.6.2 Réduction pour le GCP

Le Tableau 3.3 présente l'impact des différentes règles de réduction sur l'ensemble des 244 instances de benchmark du GCP avec la même organisation que le Tableau 3.1.

Nous observons que les règles de réduction R0 et R1 produisent la même réduction. Cela est dû au fait que les sommets ont tous le même poids dans le GCP, donc tout sommet pouvant être supprimé avec R1 sera déjà supprimé par R0. Nous constatons aussi que la règle R2 permet de réduire 14 instances supplémentaires par rapport à la règle R0 ou R1.

	$\# RI$	$\# RV$ avg	$\# RV$ max	$\%RV$ avg	$\%RV$ max	t(s) avg
R0	132	80.2	1199	26.3	87.7	16.3
R1	132	80.2	1199	26.3	87.7	16.3
R1+R2	146	115.4	1960	29.6	87.7	78.1
Iterative	146	182.5	4033	58.5	97.7	96.7

TABLE 3.3 – Impact des procédures de réduction pour le GCP. $\# RI$: nombre d'instances réduites (sur 244), $\# RV$: nombre de sommets réduits, $\%RV$: pourcentage de réduction et temps $t(s)$ en secondes.

De manière générale, on observe que plus de sommets sont réduits pour le GCP que pour le WVCP. Ceci est dû au fait que les sommets ont tous le même poids dans le cas du GCP, ce qui permet d'atteindre un pourcentage de réduction plus élevé, quelle que soit la règle appliquée. Le temps de réduction est en moyenne plus long, mais rien de surprenant, car le nombre de sommets réduit est plus important. Aussi, il prend en compte des instances qui n'existent pas pour le WVCP comme DSJR500.1c ou r1000.1c qui prennent toutes deux beaucoup de temps à réduire (2252 et 9608 secondes).

Le Tableau 3.4 liste quelques instances du GCP et le nombre de sommets réduits par les différentes règles de réduction. La colonne *time(s)* correspond au temps nécessaire pour appliquer la procédure de réduction itérative. Les valeurs en gras dans la colonne *Iterated* mettent en valeur les cas où la réduction de [Wang *et al.*, 2020] est améliorée avec notre procédure.

Les réductions étant plus importantes, il ne reste pour certaines instances qu'une seule clique, ce qui donne directement le score optimal du GCP (égal à la taille de la clique). Ce cas arrive pour 64 instances sur 244, entre autres, DSJR500.1, r250.1, r125.1c et r125.1, plusieurs GEOM, miles, pxx, rxx ou les zeroin. Par exemple, il faut 13 itérations pour réduire l'instance DSJR500.1 (500 sommets) en une seule clique de taille 12 et trouver son score optimal.

Les instances DSJC ne sont toujours pas réduites. En raison de leur nature aléatoire il est très difficile de se retrouver dans le cas où un sommet peut être retiré.

instance	|V|	density	R0	R1	R1+R2	Iterated	time(s)
DSJC125.1	125	0.1	0	0	0	0	0
DSJC125.5	125	0.5	0	0	0	0	0.12
DSJC125.9	125	0.9	0	0	0	0	0.98
DSJR500.1	500	0.0	150	150	151	**488**	0.04
GEOM110	110	0.1	17	17	17	**101**	0.01
inithx.i.1	864	0.1	705	705	709	**769**	3.83
le450_15a	450	0.1	41	41	41	**43**	0.11
le450_25b	450	0.1	131	131	131	**156**	0.23
mulsol.i.5	186	0.2	106	106	108	**114**	0.26
queen10_10	100	0.3	0	0	0	0	0.01
p42	138	0.1	10	10	10	**124**	0.01
r30	301	0.1	0	0	0	0	0.06
r1000.1	1000	0.0	99	99	99	**954**	0.36
r1000.5	1000	0.5	15	15	16	**34**	1196.34
wap04a	5231	0.0	1199	1199	1199	1199	25.82

TABLE 3.4 – Nombre de sommets supprimés par les différentes règles de réduction pour des instances du GCP. Le temps correspond au temps pour la version *Iterated*.

3.7 Conclusion

Nous avons proposé une procédure itérative de réduction incluant une amélioration d'une règle de l'état de l'art ainsi que l'adaptation d'une règle appliquée au GCP.

Nos règles s'appliquent aussi bien sur le GCP que sur le WVCP. Pour une partie des instances, ces règles permettent de réduire considérablement la taille des instances (voir ne laisser qu'une clique ou un petit ensemble de sommets très dense). Appliquer une réduction au préalable, pour une instance donnée, permet aux méthodes exactes et approchées de la résoudre en un temps plus court, car du temps passé à l'affectation de couleurs pour ces sommets est économisé.

Les instances réduites seront donc utilisées pour toutes les expérimentations présentées dans la suite de cette thèse.

CHAPITRE 4

NOUVELLES BORNES POUR LE WVCP

Ce chapitre prouve de nouvelles bornes supérieures sur le score et le nombre de couleurs nécessaires pour construire des solutions optimales du WVCP. Ces bornes peuvent être utiles pour accélérer la résolution des instances de ce problème avec des méthodes exactes ou bien heuristiques. Le calcul de ces nouvelles bornes repose sur la recherche des nombres chromatiques (ou au moins des bornes supérieures de ces nombres chromatiques) de chaque sous-graphe de G restreint à l'ensemble des sommets ayant le même poids.

Ce travail a été publié dans l'article [Goudet *et al.*, 2023] pour IJCAI 2023.

4.1 Introduction

Il est possible de réduire l'espace de recherche pour les problèmes de coloration en utilisant des bornes inférieures et supérieures, à la fois sur le score, mais aussi sur le nombre de couleurs à utiliser pour obtenir une solution optimale.

Les bornes sur le score sont notamment très importantes pour accélérer la résolution d'une instance avec une méthode exacte ou heuristique. On sait que le score optimal est prouvé pour une instance donnée lorsque les bornes supérieures et inférieures sur le score sont égales. Par exemple, dans [Wang *et al.*, 2020], pour une instance donnée du WVCP, les auteurs calculent une borne inférieure sur le score avec une recherche de cliques de poids maximal, puis ils appliquent une recherche locale qui permet d'établir une nouvelle borne supérieure sur le score. Lorsque la borne supérieure sur le score atteint la borne inférieure, la recherche peut s'arrêter et la meilleure solution obtenue est prouvée optimale.

Par ailleurs, les bornes sur le nombre de couleurs à utiliser pour obtenir une solution optimale peuvent être très intéressantes pour le WVCP que ce soit pour une heuristique ou bien une méthode exacte. Nous avons vu dans le Chapitre 1, que le nombre de couleurs pour construire une solution du WVCP n'est pas fixé à l'avance. D'une manière générale, l'espace de recherche pour le WVCP correspond à l'ensemble des partitions des sommets du graphe en k groupes de couleurs avec $1 \leq k \leq |V|$. Or, il est possible de trouver des bornes inférieures et supérieures

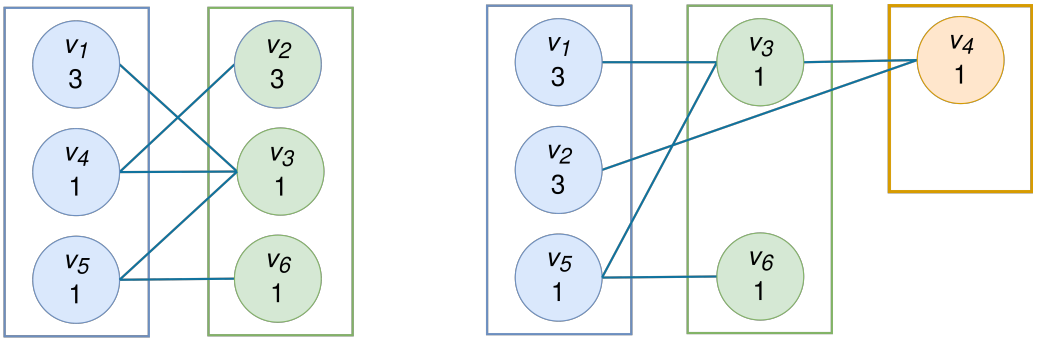


FIGURE 4.1 – Dans cette figure nous avons deux fois le même graphe avec deux colorations différentes. À gauche, coloré avec 2 couleurs, optimal pour le GCP, avec un score de 6 pour le WVCP. À droite, coloré avec 3 couleurs avec un score de 5 pour le WVCP, soit le score optimal.

sur ce nombre k de couleurs à utiliser pour construire une partition optimale, ce qui permet de réduire grandement l'espace de recherche à explorer.

Nous pourrions imaginer que le nombre maximum de couleurs à utiliser pour trouver une solution optimale du WVCP est le nombre chromatique $\chi(G)$ du graphe. Pour résoudre une instance du WVCP, il serait alors possible de procéder en deux temps : résoudre le GCP pour l'instance considérée, puis travailler avec ce nombre de couleurs fixé pour résoudre le problème d'affectation des poids dans les différents groupes de couleurs de façon à minimiser le score du WVCP. Or, nous montrons avec le contre-exemple présenté sur la Figure 4.1 que $\chi(G)$ ne constitue pas une borne supérieure sur le nombre de couleurs pour le WVCP.

La Figure 4.1 montre un exemple de graphe, dont le nombre chromatique $\chi(G)$ est égal à 2, avec deux colorations différentes. La coloration de gauche correspond à une minimisation du score du WVCP, avec un score de 6, où le nombre maximum de couleurs autorisées est égal à $\chi(G)$, 2. La coloration de droite est optimale pour le WVCP s'il n'y a pas de restriction sur le nombre de couleurs à utiliser, avec un score de 5, donc plus petit que 6. Pour obtenir cette solution optimale, il a été nécessaire d'utiliser une couleur supplémentaire, soit 3 couleurs au lieu de 2. On voit donc dans ce cas, qu'utiliser une borne supérieure sur le nombre de couleurs égale à $\chi(G)$ ne permet pas de trouver la solution optimale du WVCP.

Les principales bornes existantes de la littérature sont les suivantes :

- **la borne inférieure sur le score** : donnant le score minimal pour la solution optimale. Une telle borne peut être obtenue par des relaxations du problème, ou bien avec des solveurs ILP ou *branch and bound* [Malaguti *et al.*, 2009, Cornaz *et al.*, 2017]. Une autre approche pour la calculer consiste à extraire des cliques de poids maximal du graphe comme dans [Wang *et al.*, 2020] (cf. Proposition 1).

- **la borne supérieure sur le score** : donnant le score maximal pour la solution optimale. Une valeur naturelle pour ce score est la somme des poids de tous les sommets du graphe. Lorsqu'une nouvelle solution légale de meilleur score est obtenue pour une instance, ce score devient une nouvelle borne supérieure. Les bornes supérieures connues dans la littérature sont indiquées dans la colonne *Best Known Score* ou BKS du Tableau 1.1 (Chapitre 1) pour différentes instances de référence.
- **la borne inférieure sur le nombre de couleurs** : donnant le nombre minimal de couleurs pour la solution optimale. Une borne inférieure sur le nombre de couleurs peut être trouvée en utilisant la taille de la plus grande clique du graphe.
- **la borne supérieure sur le nombre de couleurs** : donnant le nombre maximal de couleurs pour la solution optimale. Une borne supérieure globale sur le nombre de couleurs pour le WVCP a été établie par [Demange *et al.*, 2007] comme étant $\Delta + 1$, avec Δ, le degré maximal des sommets dans G.

Dans ce chapitre, nous proposons une nouvelle borne supérieure sur le score du WVCP qui peut être précalculée pour une instance donnée sans utiliser un *Best Known Score* connu. Ensuite, nous présentons notre contribution principale de ce chapitre qui est le raffinement de la borne supérieure sur le nombre de couleurs pour le WVCP de [Demange *et al.*, 2007] pour certaines classes de graphes. Cette nouvelle borne supérieure sur le nombre de couleurs permet dans certains cas de réduire encore plus l'espace de recherche à explorer pour une méthode exacte ou une heuristique. [1] Enfin, dans ce chapitre, nous proposons une évaluation de l'impact des différentes bornes venant de la littérature et de celles que nous proposons pour résoudre de façon exacte des instances de références avec des modèles de programmation par contrainte mis au point dans [Goudet *et al.*, 2023] (modèles non détaillés dans ce manuscrit).

Ce chapitre est organisé comme suit. La Section 4.2 introduit des notations qui seront utiles dans ce chapitre. La Section 4.3 présente une première propriété pour l'établissement de la borne supérieure sur le score présentée dans la Section 4.4 et la borne supérieure sur le nombre de couleurs de la Section 4.5. Enfin, des expérimentations qui montrent l'impact de ces bornes pour la résolution de différentes instances du WVCP sont présentées dans la Section 4.6.

1. Les nouvelles bornes supérieures que nous proposons pour le WVCP pourraient naturellement s'appliquer pour le GCP, mais n'auraient aucun intérêt dans le cas du GCP comme nous allons le voir dans ce chapitre.

4.2 Définitions et Notation

Dans ce chapitre, nous notons $W = \{w(v), v \in V\}$ l'ensemble des poids différents des sommets donnés par la fonction w appliquée sur chaque sommet de V. $w_{max} = \max(W)$ est défini comme le poids maximal dans W. Pour chaque poids w, il est possible d'extraire un sous-graphe $G_w = (V_w, E_w)$ de G où $V_w = \{v \in V, w(v) = w\}$ et $E_w = \{(u, v) \in E, w(u) = w$ et $w(v) = w\}$. Le nombre chromatique de chaque sous-graphe G_w est noté $\chi(G_w)$. Si aucun sommet de V n'a de poids égal à w alors G_w est un graphe vide et par convention $\chi(G_w) = 0$.

Étant donné une solution S, correspondant à une partition des sommets de V en k sous-ensembles indépendants $S = \{V_1, \ldots, V_k\}$, nous notons $w(V_i)$ le poids maximal d'un sommet du sous-ensemble V_i : $w(V_i) = \underset{v \in V_i}{max}\, w(v)$. Le nombre de sous-ensembles indépendants dans S avec un poids égal à w est noté $k_w^S = |\{w(V_i) = w, i \in [1, k]\}|$.

4.3 Propriété d'une Solution Optimale

Nous introduisons dans cette section un premier Lemme qui permettra d'établir les bornes supérieures sur le score et le nombre de couleurs présentées dans les sections suivantes.

Lemme 1. *Étant donné un graphe pondéré $G = (V, E, w)$ et une solution optimale S^* pour cette instance du WVCP, nous avons :*

$$\sum_{i=1}^{w_{max}} i \times (\chi(G_i) - k_i^{S^*}) \geq 0 \tag{4.1}$$

Démonstration. Définissons un opérateur g_w prenant en entrée une solution légale S et retournant une solution légale S' où tous les sommets de poids w dans S ont été supprimés de leur groupe de couleurs et placés dans χ_w nouveaux groupes de couleurs disjoints et indépendants uniquement composés des sommets de même poids w. Cette opération est toujours possible par définition du nombre chromatique d'un graphe. Par convention, s'il n'y a aucun poids dans W de valeur w, cette opération n'a pas d'effet. Lorsque cette opération est appliquée de manière itérative pour chaque poids de W en partant de la solution S^*, nous obtenons w_{max} nouvelles solutions légales : $S^1 = g_1(S^*)$, $S^2 = g_2 \circ g_1(S^*)$, …, $S^{w_{max}} = g_{w_{max}}(S^{w_{max}}) \circ \cdots \circ g_1(S^*)$. Le score de chaque nouvelle solution légale S^j pour $j = 1 \ldots w_{max}$ est égal à :

$$f(S^j) = f(S^*) + \sum_{i=1}^{j} i \times (\chi(G_i) - k_i^{S^*}) \tag{4.2}$$

Comme S^* est une solution optimale, nous avons $f(S^j) \geq f(S^*)$ et :

$$\sum_{i=1}^{w_{max}} i \times (\chi(G_i) - k_i^{S^*}) \geq 0 \text{ pour } j = 1, \ldots, w_{max} \tag{4.3}$$

□

La Figure 4.2 illustre la division du graphe en sous-graphes induits par les poids des sommets. Le graphe G est divisé en deux sous-graphes G_1 et G_3.

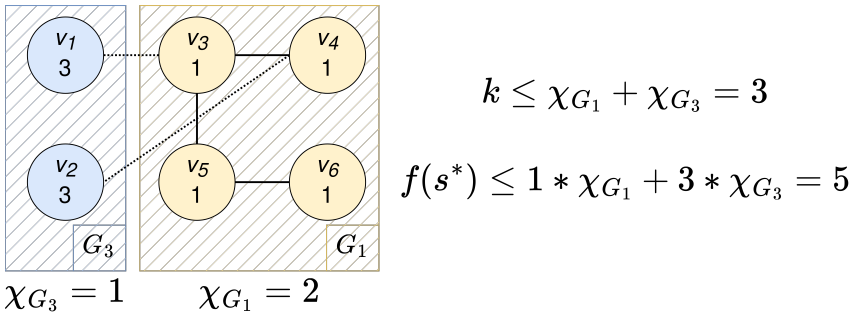


FIGURE 4.2 – Division du graphe en sous-graphes induits par les poids des sommets. Les sommets de poids 1 sont en jaune (G_1) et ceux de poids 3 en bleu (G_3). Leurs nombres chromatiques sont $\chi_{G_1} = 2$ et $\chi_{G_3} = 1$, car il faut 2 et 1 couleurs pour colorer G_1 et G_3 respectivement. Une borne supérieure pour le nombre de couleurs nécessaire pour colorer G est $\chi_{G_1} + \chi_{G_3} = 3$. Une borne supérieure pour le score de G est la somme des nombres chromatiques des sous-graphes multipliés par le poids correspondant, soit $2 \times 1 + 1 \times 3 = 5$. Les sections suivantes donnent la démonstration de ces bornes.

4.4 Borne Supérieure sur le Score

À partir du Lemme 1 établi dans la section précédente, nous établissons de façon immédiate une nouvelle borne pour le score du WVCP.

Théoreme 1. *Étant donné un graphe pondéré $G = (V, E, w)$ et une solution optimale S^* correspondant à une partition des sommets de V en k sous-ensembles indépendants $S^* = \{V_1, \ldots, V_k\}$, nous avons :*

$$f(S^*) \leq \sum_{w \in W} w \times \chi(G_w)$$

.

Démonstration. Soit $S^* = \{V_1, \ldots, V_k\}$ une solution optimale pour le graphe $G = (V, E, w)$ correspondant à une partition des sommets de V en k sous-ensembles indépendants.

Partant du Lemme 1, nous avons :

$$\sum_{i=1}^{w_{max}} i \times (\chi(G_i) - k_i^{S^*}) \geq 0 \tag{4.4}$$

Or, par définition du score du WVCP comme :

$$f(S^*) = \sum_{i=1}^{w_{max}} i \times k_i^{S^*} \tag{4.5}$$

nous obtenons :

$$f(S^*) \leq \sum_{i=1}^{w_{max}} i \times \chi(G_i) \tag{4.6}$$

ce qui prouve la borne supérieure sur le score. □

À titre d'exemple, considérons à nouveau le graphe WVCP montré dans la Figure 4.2. La borne supérieure sur le score dérivée du Théorème 1 est égale à $3 \times 1 + 1 \times 2 = 5$, ce qui correspond au score optimal pour cette instance.

Pour certaines classes d'instances, cette borne supérieure peut en effet être très intéressante. On remarque par exemple que la borne supérieure donne le score optimal pour tout graphe G qui inclut une clique contenant χ_{G_w} sommets de poids w pour chaque poids différent $w \in W$. On verra cependant dans la Section 4.6 expérimentale ci-dessous que la plupart des instances de référence ne présentent pas cette structure, ce qui donne des bornes supérieures assez larges. Ce cas apparaît notamment pour les instances rxx et pxx caractérisées par une petite clique pondérée maximale et de nombreux poids différents.

Borne supérieure sur le score

Pour résumer, en divisant une solution optimale pour une instance en de multiples solutions indépendantes (optimales du point de vue du nombre de couleurs) pour chaque sous-graphe induit par les poids existant du graphe, le score de la solution optimale est inférieur ou égal à la somme des scores de chaque solution indépendante. Donc, si nous divisons un graphe en sous-graphes pour chaque poids et trouvons pour chacun de ces sous-graphes leur nombre chromatique (problème du GCP), puis multiplions ce nombre chromatique par le poids correspondant, nous obtenons une borne supérieure sur le score du WVCP.

4.5 Borne Supérieure sur le Nombre de Couleurs

Dans cette section, nous proposons maintenant une borne supérieure sur le nombre de couleurs requises pour construire une solution optimale du WVCP.

Théoreme 2. *Étant donné un graphe pondéré $G = (V, E, w)$ et une solution optimale S^* correspondant à une partition des sommets de V en k sous-ensembles indépendants $S^* = \{V_1, \ldots, V_k\}$, nous avons :*

$$k \leq \sum_{w \in W} \chi(G_w)$$

.

Démonstration. Faisons l'hypothèse que $k > \sum_{w \in W} \chi(G_w)$ et montrons que nous arrivons à une contradiction. Partant du principe que $k = \sum_{w \in W} k_w^{S^*}$, nous avons :

$$\sum_{w \in W} (k_w^{S^*} - \chi(G_w)) > 0 \tag{4.7}$$

Montrons maintenant par récurrence que pour tout $c \in [1, w_{max}]$, nous avons :

$$\sum_{i=1}^{m-c} (i + c - m - 1)(\chi(G_i) - k_i^{S^*}) \geq 0. \tag{4.8}$$

En sommant les deux membres de l'inégalité de l'Équation 4.1 donné par le Lemme 1 pour $j = w_{max}$ et l'inégalité donnée par l'Équation 4.7 multipliée par w_{max}(strictement positif), nous obtenons :

$$\sum_{i=1}^{w_{max}} i \times (\chi_(G_i) - k_i^{S^*}) + w_{max} \times \sum_{i=1}^{w_{max}} (k_i^{S^*} - \chi_(G_i)) > 0, \tag{4.9}$$

et ensuite :

$$\sum_{i=1}^{w_{max}-1} (i - w_{max})(\chi_(G_i) - k_i^{S^*}) > 0. \tag{4.10}$$

Par conséquent, la propriété donnée par l'Équation 4.8 est vraie lorsque $c = 1$. Maintenant, nous supposons que cette propriété est vraie au rang c et montrons qu'elle restera vraie au rang $c + 1$, pour $c = 1, \ldots, w_{max} - 2$.

En sommant les deux membres de l'inégalité de l'Équation 4.1 donné par le Lemme 1 pour $j = w_{max} - c$ et l'inégalité donnée par l'Équation 4.8 multipliée par $w_{max} - c$ (strictement

positif comme $c < w_{max}$), nous obtenons :

$$\sum_{i=1}^{m-c} i(\chi_{G_i} - k_i^{S^*}) + (m-c)\sum_{i=1}^{m-c}(i+c-m-1)(\chi_{G_i} - k_i^{S^*}) > 0.$$

$$\sum_{i=1}^{m-c}(i + mi + mc - m^2 - m - ci - c^2 + cm + c) \times (\chi_{G_i} - k_i^{S^*}) > 0.$$

En réarrangeant les termes, et en factorisant par $(i + c - w_{max})$, cela donne

$$\sum_{i=1}^{m-c}(i+c-m)(m+1-c)(\chi_{G_i} - k_i^{S^*}) > 0.$$

Ensuite, en divisant les deux termes de cette inégalité par $w_{max} + 1 - c$ (nombre strictement positif), nous obtenons

$$\sum_{i=1}^{m-c}(i+c-m)(\chi_{G_i} - k_i^{S^*}) > 0,$$

et ensuite :

$$\sum_{i=1}^{m-c-1}(i+c-m)(\chi_{G_i} - k_i^{S^*}) > 0,$$

ce qui prouve la propriété donnée par l'Équation 4.8 au rang $c + 1$.

Maintenant en utilisant l'Équation 4.8 pour $c = w_{max} - 1$, nous obtenons :

$$(\chi_1 - k_1^{S^*}) < 0,$$

Ce qui contredit l'Équation 4.1 donnée par le Lemme 1 pour $j = 1$ et termine la preuve. □

Nous pouvons d'abord observer que cette borne supérieure sur le nombre de couleurs est cohérente lorsque tous les poids des sommets de V sont égaux à la même valeur w car le Théorème 1 indique que pour la solution optimale, le nombre de couleurs utilisées est $k \leq \chi_{G_w}$, or par définition du nombre chromatique d'un graphe, nous savons que $k = \chi_{G_w}$. D'autre part, lorsque tous les poids des sommets sont différents, le Théorème 1 indique que $k \leq |V|$ ce qui est trivial selon la définition du problème.

Considérons à nouveau le graphe WVCP montré dans la Figure 4.2. La borne supérieure dérivée du Théorème 2 sur le nombre de couleurs est égale à $3 = \chi_3 + \chi 1 = 1 + 2$, et ce nombre de 3 couleurs permet dans ce cas d'obtenir la solution optimale.

Notons que calculer les bornes supérieures des Théorèmes 1 et 2 implique de résoudre $|W|$ sous-problèmes GCP pour obtenir les nombres chromatiques χ_{G_w} des graphes G_w ($w \in W$). Résoudre le GCP est NP-difficile, [Malaguti *et al.*, 2011] montre qu'il est difficile pour les méthodes exactes de trouver une solution pour le GCP lorsque le nombre de sommets est supérieur à 100 pour les graphes aléatoires. En pratique, nous pouvons calculer rapidement des bornes supérieures sur les nombres chromatiques des sous-graphes induits en utilisant des heuristiques pour le GCP telles que l'algorithme *TabuCol* [Hertz et Werra, 1987] (cf. Chapitre 2).

Cette borne supérieure sur le nombre de couleurs nécessaires pour construire une solution optimale peut être utile pour prouver l'optimalité de certaines instances du WVCP, car elle réduit l'espace de recherche de toutes les solutions possibles (voir Section 4.6.3).

Borne supérieure sur le nombre de couleurs

Pour résumer, de la même manière que pour la borne supérieure sur le score, diviser le graphe en autant de sous-graphes qu'il existe de poids différents et en trouvant le nombre chromatique pour chacun de ces graphes, la somme de ces nombres chromatiques est une borne supérieure sur le nombre de couleurs. Dans le pire cas, il ne sera pas utilisé plus de couleurs que ce nombre dans une solution optimale.

4.6 Expérimentations

Cette section présente les différences sur les instances avec nos nouvelles bornes comparées aux résultats de [Demange *et al.*, 2007]. Les instances utilisées pour les expériences sont les instances réduites présentées dans le Chapitre 3.

Paramètres expérimentaux Les expériences ont été réalisées sur un Intel Core i7-10875H 2.30GHz. Il n'y a pas de limites de temps strictes pour le calcul des bornes. Le code est implémenté en Python 3 sauf pour la recherche du nombre chromatique pour les sous-graphes. Celui-ci calcule un nombre décroissant de couleurs avec la recherche locale *TabuCol* de [Hertz et Werra, 1987] proposée dans [Moalic et Gondran, 2018] en C++ avec une limite de 1000 itérations sans amélioration pour chaque sous-graphe. Le code et les résultats complets sont disponibles à l'adresse : `https://github.com/Cyril-Grelier/gc_wvcp_cp`.

4.6.1 Réduction des Domaines de Couleurs

Le Tableau 4.1 affiche les résultats de la réduction de domaine de couleurs expliquée dans la Section 4.5 sur un sous-ensemble d'instances.

Les colonnes 1 à 3 montrent les caractéristiques des instances obtenues après la procédure de réduction (cf. Chapitre 3). La colonne 4 est la borne supérieure globale pour chaque instance obtenue en résolvant $|W|$ sous-problèmes GCP. Il ne faut pas plus de 0.1 seconde pour chaque instance pour calculer la borne supérieure globale avec cette méthode. La colonne 5 montre la taille moyenne du domaine des sommets de chaque instance donnée en tenant compte seulement de la borne établie par [Demange *et al.*, 2007], tandis que la colonne 6 rapporte la taille moyenne du domaine des sommets lors de la prise en compte supplémentaire de notre nouvelle borne de couleurs présentée dans la Section 4.5. La colonne 7 rapporte la réduction moyenne de la taille du domaine de couleurs lors de la prise en compte de la nouvelle borne.

La prise en compte de la nouvelle borne sur le nombre de couleurs n'a aucun impact pour les instances à faible densité, mais est plus utile lorsque l'instance est dense.

Instance	$\|V\|'$	$\|E\|'$	$\bar{k}$	$\|N(v)+1\|_{avg}$	$min(\|N(v)+1\|,\bar{k})_{avg}$	réduction
DSJC125.1g	125	736	14	12.8	11.96	-0.84
DSJC125.5g	125	3891	34	63.3	34	-29.3
DSJC125.9g	125	6961	72	112.4	72	-40.4
DSJC250.1	250	3218	45	26.7	26.7	0
DSJC250.5	250	15668	88	126.3	88	-38.3
DSJC250.9	250	27897	162	224.2	162	-62.2
DSJR500.1	244	1904	45	16.6	16.6	0
flat1000_76_0	1000	246708	187	494.4	187	-307.4
GEOM110	87	495	24	12.3	12.3	0
inithx.i.1	181	6754	78	75.6	58.3	-17.3
le450_15a	420	7936	61	38.8	37.9	-0.9
mulsol.i.5	104	2121	58	41.8	40.2	-1.6
p42	135	1149	67	18.0	18.0	0
queen10_10	100	1470	41	30.4	30.4	0
R100_1g	98	503	14	11.3	10.8	-0.5
R100_5g	100	2456	30	50.1	30	-20.1
R100_9g	100	4438	59	89.8	59	-30.8
r30	301	4122	236	28.4	28.4	0
wap01a	2157	104235	121	97.6	87.2	-10.4

TABLE 4.1 – Taille moyenne du domaine de couleurs pour les sommets des différentes instances lors de l'utilisation de différentes règles de réduction de domaine de couleurs.

4.6.2 Bornes sur le Nombre de Couleurs et le Score

Le Tableau 4.2 montre les bornes inférieures et supérieures sur le score et le nombre de couleurs requis pour une solution optimale donnée par les Théorèmes 1 et 2 ou provenant d'études antérieures. Les colonnes 1 à 4 montrent les différentes caractéristiques des instances. La colonne 4 introduit une mesure de l'hétérogénéité des poids définie par $h_W = \frac{|W|}{|V|}$. La colonne 5 rapporte la valeur $\Delta + 1$ (Δ désigne le degré maximum des sommets dans le graphe), borne supérieure sur le nombre de couleurs proposée par [Demange *et al.*, 2007]. La colonne 6 rapporte une borne inférieure sur le nombre de couleurs qui correspond à la taille maximale d'une clique dans G trouvée avec la procédure d'extraction de cliques appliquée lors de la première étape de prétraitement de réduction (voir la Section 3.5). La colonne 7 correspond à la borne supérieure sur le nombre de couleurs donnée par le Théorème 1 en résolvant $|W|$ sous-problèmes GCP. Cette borne supérieure est écrite en gras lorsqu'elle est meilleure que la borne $\Delta + 1$, ce qui arrive presque tout le temps, sauf pour des instances spécifiques telles que $r30$ caractérisée par une forte hétérogénéité des poids h_W. La colonne 8 est une borne inférieure sur le score calculé avec la méthode proposée par [Wang *et al.*, 2020] (voir Proposition 1) en utilisant l'ensemble des cliques pondérées maximales calculées avec l'algorithme FastWClq. La colonne 9 rapporte la borne supérieure sur le score calculée selon le Théorème 1. Cette borne inférieure est égale au BKS pour 72 instances réduites sur 188, mais seulement pour 2 avec les instances originales, d'où l'importance d'appliquer une procédure de réduction efficace en amont.

					colors bounds		score bounds	
Instance	$\|V'\|$	density	h_W	$\Delta + 1$	lb	ub	lb	ub
DSJC125.1g	125	0.1	0.04	24	4	**14**	19	42
DSJC125.5g	125	0.5	0.04	76	10	**34**	42	105
DSJC125.9g	125	0.9	0.04	121	32	**72**	124	220
DSJR500.1	244	0.03	0.08	26	12	26	166	477
GEOM110	87	0.11	0.11	20	9	20	65	151
inithx.i.1	181	0.05	0.1	169	54	**78**	569	800
le450_15a	420	0.08	0.05	99	15	**61**	206	628
le450_25b	345	0.08	0.06	108	25	**73**	307	735
mulsol.i.5	104	0.23	0.18	88	31	**58**	367	574
queen10_10	100	0.59	0.19	36	10	36	153	420
p42	135	0.12	0.46	25	14	25	2466	8108
r30	301	0.09	0.76	35	19	35	9816	104285

TABLE 4.2 – Bornes inférieures et supérieures sur le score et les couleurs. Les valeurs en gras indiquent une amélioration de la borne supérieure par rapport à la borne de [Demange *et al.*, 2007] ($\Delta + 1$).

4.6.3 Impact des Bornes

Le Tableau 4.3 montre l'impact de la borne supérieure sur le nombre de couleurs (voir le Théorème 1) ainsi que l'impact de l'introduction simultanée de toutes les bornes sur le modèle primal de l'article [Goudet *et al.*, 2023], non présenté en détail dans ce manuscrit. Nous avons aussi testé l'impact de la borne sur le nombre de couleurs dans le cadre du MCTS présenté dans le Chapitre 5, cet ajout permet un léger gain de temps pour prouver l'optimalité de certaines instances, mais n'a pas un impact important de manière générale.

La colonne 1 est le nom de l'instance. La colonne 2 rapporte le meilleur score connu (BKS) de la littérature. Certains de ces BKS sont obtenus dans des conditions spécifiques, comme une journée de calcul en parallèle sur un processeur graphique (GPU) dans [Goudet *et al.*, 2022], et sont donc très difficiles à atteindre. Une étoile est ajoutée au score lorsqu'il a été prouvé optimal. La plupart de ces preuves d'optimalité ont été obtenues avec la formulation MIP de [Cornaz *et al.*, 2017] résolue pendant 10h à l'aide de CPLEX [Nogueira *et al.*, 2021]. Un score est écrit en gras lorsqu'il correspond au BKS. Le temps en secondes nécessaire pour prouver l'optimalité est rapporté, sinon, "tl" est affiché, ce qui signifie que la limite d'une heure de recherche a été atteinte. Le score est souligné s'il s'agit d'une nouvelle preuve d'optimalité qui n'avait jamais été prouvée auparavant dans la littérature.

		primal		primal ub color		primal all bounds	
instance	BKS	score	time(s)	score	time(s)	score	time(s)
DSJC125.1g	23	**<u>23*</u>**	862	**<u>23*</u>**	435	**<u>23*</u>**	451
DSJC125.5g	71	78	tl	78	tl	78	tl
DSJC125.9g	169*	176	tl	176	tl	176	tl
DSJR500.1	169	187	tl	177	tl	169	tl
GEOM110	68*	69	tl	**68***	1893	**68***	1729
inithx.i.1	569*	569	tl	569	tl	**569***	54
le450_15a	212	245	tl	234	tl	234	tl
le450_25b	307	307	tl	307	tl	**<u>307*</u>**	322
mulsol.i.5	367*	367	tl	367	tl	**367***	31
queen10_10	162	170	tl	169	tl	169	tl
p42	2466*	2480	tl	2466	tl	**2466***	2908
r30	9816*	9831	tl	9831	tl	9831	tl
#BKS		101/188		105/188		107/188	
#Optimal		72/188		75/188		95/188	

TABLE 4.3 – Impact des bornes sur le modèle primal. L'étoile indique la preuve d'optimalité, les valeur en gras un score est égal au BKS et les valeurs soulignées les nouvelles preuves d'optimalité.

Nous voyons que la borne supérieure sur le nombre de couleurs (colonnes 5-6) dérivant du Théorème 1 peut réduire considérablement le temps passé par le solveur sur chaque instance, car elle réduit le domaine des couleurs disponibles pour chaque sommet. Les colonnes 7-8 correspondent aux résultats obtenus lorsque toutes les bornes inférieures et supérieures présentées dans le Tableau 4.2 sont activées simultanément, ce qui permet d'augmenter le nombre de BKS atteints (107 sur 188), ainsi que le nombre de preuves d'optimalité (95 sur 188) pour l'ensemble des instances. Par exemple, l'instance p42 peut être résolue exactement en 2908 secondes avec le modèle primal lorsque toutes les bornes précalculées sont utilisées.

4.7 Conclusion

Nous avons établi de nouvelles bornes supérieures sur le score et le nombre de couleurs nécessaires pour résoudre de manière optimale le WVCP.

Ces bornes sont basées sur la résolution de sous-problèmes GCP et permettent d'améliorer les bornes existantes pour une grande partie des instances de la littérature. Celles-ci sont utiles pour le développement de méthodes de résolution exactes et approchées pour le WVCP. En effet, elles permettent de réduire l'espace de recherche en termes de nombre de couleurs nécessaires. Comme pour la réduction de sommets, cette réduction de l'espace de recherche peut être effectuée en phase de prétraitement avant d'appliquer tout type de méthodes par la suite. Nous utiliserons notamment la borne supérieure sur le nombre de couleurs du WVCP dans l'algorithme présenté dans le chapitre suivant.

CHAPITRE 5

MCTS ET ALGORITHMES GLOUTONS

Ce chapitre présente la méthode Monte Carlo Tree Search (MCTS) combinée à des heuristiques dédiées pour résoudre le GCP et le WVCP. En plus de l'algorithme MCTS de base, nous étudions plusieurs variantes de MCTS où la simulation aléatoire conventionnelle est remplacée par des heuristiques gloutonnes. Nous menons des expériences sur des benchmark bien connus de l'état de l'art pour évaluer ces variantes de MCTS. Nous fournissons des preuves empiriques pour mettre en lumière les avantages et les limites de chaque stratégie de simulation.

Ce chapitre présente une partie du travail réalisé pour l'article [Grelier *et al.*, 2022] et présenté à la conférence EVOCOP 2022. Il a bénéficié de la possibilité d'être proposé sous la forme d'une version étendue pour la revue *SN Computer Science* (en cours de soumission).

5.1 Introduction

L'algorithme du MCTS, inventé par [Coulom, 2006], est un algorithme de recherche heuristique qui a suscité un intérêt considérable en raison de son succès spectaculaire pour le jeu de Go [Gelly *et al.*, 2006], et dans bien d'autres domaines [Browne *et al.*, 2012]. Il a été récemment revisité en combinaison avec des techniques modernes d'apprentissage profond pour les jeux à deux joueurs difficiles comme le jeu de Go (cf. AlphaGo [Silver *et al.*, 2016]). Le MCTS a également été appliqué à des problèmes d'optimisation combinatoire, considérés comme des jeux à un seul joueur, tels que le problème du voyageur de commerce [Edelkamp et Greulich, 2014] ou le problème du sac à dos [Jooken *et al.*, 2023]. Une variante de cet algorithme MCTS, NRPA, a récemment été implémentée avec un certain succès pour le GCP dans [Cazenave *et al.*, 2021]. Dans ce travail, nous étudions pour la première fois l'approche MCTS pour résoudre le WVCP et le testons aussi sur le GCP.

Dans le MCTS, un arbre est construit de manière incrémentale et asymétrique. À chaque itération, une politique de sélection dans l'arbre équilibrant l'exploration et l'exploitation est utilisée pour trouver le nœud le plus critique à développer. Une simulation est ensuite exécutée à partir du nœud développé et l'arbre de recherche est mis à jour avec le résultat de cette simula-

tion. Ses propriétés incrémentales et asymétriques font du MCTS un candidat prometteur pour le WVCP, car dans ce problème, seul le sommet le plus lourd de chaque groupe de couleurs a un impact sur le score objectif. Par conséquent, apprendre à colorer les sommets les plus lourds du graphe avant de colorer le reste du graphe semble particulièrement pertinent pour ce problème.

Dans ce chapitre, nous présentons un algorithme MCTS dédié au WVCP, mais aussi applicable au GCP, qui considère le problème du point de vue de la coloration séquentielle avec un ordre de sommets prédéfini. L'exploration de l'arbre est accélérée avec l'utilisation de règles d'élagage spécifiques, qui offrent la possibilité d'explorer l'arbre entier en un temps raisonnable pour les petites instances et d'obtenir des preuves d'optimalité.

Le chapitre est organisé comme suit. La Section 5.2 présente le problème de coloration des sommets pondérés et l'approche constructive avec un arbre. La Section 5.3 décrit l'algorithme MCTS conçu pour résoudre le problème et les différents opérateurs pour la simulation. La Section 5.4 rend compte des résultats de calcul de différentes versions de MCTS. La Section 5.5 discute des contributions et présente des perspectives de recherche future.

5.2 Approche Constructive avec un Arbre

Cette section présente une approche basée sur un arbre pour le WVCP, qui vise à explorer l'espace de recherche partiel et légal de ce problème.

5.2.1 Espace de Recherche Partiel et Légal

L'espace de recherche Ω_p étudié dans notre algorithme concerne les colorations légales, mais potentiellement partielles (cf. définition de cet espace avec l'Équation 1.4 du Chapitre 1).

Une coloration partielle légale S avec k couleurs déjà utilisées est une partition de l'ensemble des sommets V en k ensembles indépendants V_i ($1 \leq i \leq k$), et un ensemble de sommets non colorés $U = V \setminus \bigcup_{i=1}^{k} V_i$. Un ensemble indépendant V_i est un ensemble de sommets mutuellement non adjacents du graphe : $\forall u, v \in V_i, (u, v) \notin E$. Pour le WVCP, le nombre de couleurs k qui peuvent être utilisées n'est pas imposé à priori. Nous pouvons tout de même utiliser des bornes supérieures sur le nombre de couleurs établies dans le chapitre précédent, de façon à réduire l'espace de recherche à explorer. La borne supérieure sur le nombre de couleurs que nous utilisons pour résoudre une instance donnée sera notée $\bar{k}$ dans ce chapitre. Nous verrons que cette borne globale sera utilisée en phase d'expansion du MCTS, de façon à limiter "en largeur" la construction de l'arbre de recherche et à faciliter son élagage. Toutefois, en phase de

simulation (ou *playout*), la borne supérieure sur le nombre de couleurs ne sera pas forcément respectée de façon à toujours permettre la construction d'une solution légale (même si celle-ci n'est pas optimale) à chaque itération du MCTS et ainsi toujours retourner un nouveau score pour continuer à apprendre.

Une solution du WVCP est dite partielle si $U \neq \emptyset$ et complète sinon. L'objectif du WVCP est de trouver une solution complète $S = \{V_1, \ldots, V_k\}$ avec un score minimum $f(S)$ donné par : $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$. Nous rappelons que l'objectif du GCP est de minimiser ce même score quand tous les poids sont égaux (dans ce cas le score correspond au nombre de couleurs utilisées). L'algorithme MCTS que nous proposons dans ce chapitre pourra donc naturellement s'appliquer aussi au GCP.

5.2.2 Recherche Arborescente pour la Coloration Pondérée

La recherche arborescente basée sur le backtracking est une approche populaire pour le problème de coloration de graphe [Brélaz, 1979, Kubale et Jackowski, 1985, Lewis, 2015]. Dans notre cas, un algorithme de recherche arborescente peut être utilisé pour explorer l'espace de recherche partiel et légal du WVCP précédemment défini.

À partir d'une solution où aucun sommet n'est coloré (c'est-à-dire $U = V$) et qui correspond au nœud racine R de l'arbre, les nœuds enfants C sont successivement sélectionnés dans l'arbre, consistant à colorer un nouveau sommet à la fois. Ce processus est répété jusqu'à ce qu'un nœud terminal T soit atteint (tous les sommets sont colorés). Une solution complète (c'est-à-dire une coloration légale) correspond donc à une branche allant du nœud racine à un nœud terminal.

La sélection de chaque nœud enfant correspond à l'application d'un mouvement à la solution partielle actuelle en cours de construction. Un mouvement consiste à attribuer une couleur particulière i à un sommet non coloré $u \in U$, noté $< u, U, V_i >$. L'application d'un mouvement à la solution partielle actuelle S donne une nouvelle solution $S \oplus < u, U, V_i >$. Cet algorithme de recherche arborescente ne considère que les mouvements légaux pour rester dans l'espace légal partiel. Pour une solution partielle $S = \{V_1, ..., V_k, U\}$, un mouvement $< u, U, V_i >$ est dit légal si aucun sommet de V_i n'est adjacent au sommet u. À chaque niveau de l'arbre, si on ne restreint pas le nombre de couleurs à utiliser, il y a toujours au moins un mouvement légal possible qui applique à un sommet une nouvelle couleur qui n'a jamais été utilisée auparavant (ou mettant ce sommet dans un nouvel ensemble vide V_i, $k + 1 \leq i \leq n$).

L'application d'une succession de n mouvements légaux à partir de la solution initiale donne une coloration légale du WVCP et atteint un nœud terminal de l'arbre. Au cours de ce processus, au niveau t de l'arbre ($0 \leq t < n$), la solution partielle légale actuelle $S = \{V_1, ..., V_k, U\}$ a

déjà utilisé k couleurs et t sommets ont déjà reçu une couleur. Par conséquent $|U| = n - t$.

À chaque niveau de l'arbre, une première approche naïve pourrait consister à considérer tous les mouvements légaux possibles, correspondant au choix d'un sommet dans l'ensemble U et à l'attribution au sommet d'une couleur i, avec $1 \leq i \leq n$. Ce type de choix peut fonctionner avec de petits graphes, mais avec de grands graphes, le nombre de mouvements légaux possibles devient trop grand. En effet, à chaque niveau t, le nombre de mouvements légaux possibles peut aller jusqu'à $(n - t) \times n$.

Pour réduire l'ensemble des possibilités de mouvement, nous proposons de considérer les sommets du graphe dans un ordre prédéfini $(u_1, \ldots, u_n)$. De plus, pour choisir une couleur pour le sommet entrant, nous ne considérons que les couleurs déjà utilisées dans la solution partielle plus une couleur (création d'un nouvel ensemble indépendant) tant que ce nombre reste inférieur au degré du sommet plus un, bornant le nombre de couleurs nécessaires pour colorer un sommet (borne locale pour un sommet définie par [Demange *et al.*, 2007]). Ainsi, pour la solution courante, partielle et légale, $S = \{V_1, ..., V_k, U\}$, au plus $d(u) + 1$ mouvements sont considérés pour le prochain sommet u à colorer. A priori, sans tenir compte encore de la borne globale $\bar{k}$ sur le nombre à utiliser (qui peut être plus petite que $d(u) + 1$), mais si k couleurs ont déjà été utilisées, l'ensemble des mouvements légaux pour placer u est

$$\mathcal{L}(S) = \{< u, U, V_i >, 1 \leq i \leq k, \forall v \in V_i, (u, v) \notin E\} \cup \{< u, U, V_{k+1} >\}, \tag{5.1}$$

si $k < d(u) + 1$, ou bien

$$\mathcal{L}(S) = \{< u, U, V_i >, 1 \leq i \leq d(u) + 1, \forall v \in V_i, (u, v) \notin E\} \tag{5.2}$$

si $k \geq d(u) + 1$.

Cette décision coupe les symétries dans l'arbre tout en réduisant le nombre de facteurs de branchement à chaque niveau de l'arbre.

5.2.3 Ordre des Sommets Prédéfini

Nous proposons de considérer un ordre prédéfini des sommets, triés par poids puis par degré. Les sommets avec des poids plus élevés sont placés en premier. Si deux sommets ont le même poids, alors le sommet avec le degré le plus élevé est placé en premier. Cet ordre est intuitivement pertinent pour le WVCP, car il est plus important de placer d'abord les sommets

avec des poids lourds qui ont le plus d'impact sur le score ainsi que les sommets avec le degré le plus élevé, car ils sont les variables de décision les plus contraintes. Un tel ordre a déjà montré son efficacité avec des approches constructives gloutonnes pour le GCP [Brélaz, 1979] et le WVCP [Nogueira *et al.*, 2021].

De plus, cet ordre des sommets permet un calcul simple du score lors de la construction de l'arbre. En effet, comme les sommets sont triés par ordre décroissant de leurs poids, et que le score du WVCP ne compte que le poids maximum de chaque groupe de couleurs, avec cet ordre des sommets, le score n'augmente que de la valeur $w(u)$ lorsqu'un nouveau groupe de couleurs est créé pour le sommet u.

5.3 Arbre de Recherche de Monte Carlo

L'arbre de recherche présenté dans la dernière sous-section peut être très grand, en particulier pour les instances les plus difficiles. Par conséquent, en pratique, il est souvent impossible d'effectuer une recherche exhaustive de cet arbre, en raison du temps de calcul et des exigences de mémoire élevées. C'est pourquoi nous proposons dans ce travail de nous tourner vers une adaptation de l'algorithme MCTS pour le WVCP pour explorer cet arbre de recherche. Le MCTS garde en mémoire un arbre (ci-après dénommé *arbre MCTS*) qui ne correspond qu'aux nœuds déjà explorés de l'arbre de recherche présenté dans la dernière sous-section. Dans l'arbre MCTS, une feuille est un nœud dont les enfants n'ont pas encore tous été explorés tandis qu'un nœud terminal correspond à une solution complète. Le MCTS peut guider la recherche vers les branches les plus prometteuses de l'arbre, en équilibrant l'exploitation et l'exploration et en apprenant continuellement à chaque itération.

5.3.1 Framework Général

L'Algorithme 7 présente le MCTS pour le WVCP. Il prend en entrée un graphe pondéré et tente de trouver une coloration légale S avec le score minimum $f(S)$. Il commence par une solution initiale où le premier sommet est placé dans le premier groupe de couleurs. C'est le nœud racine de l'arbre MCTS. Ensuite, l'algorithme répète plusieurs itérations jusqu'à ce qu'un critère d'arrêt soit atteint. À chaque itération, une solution légale est entièrement construite, ce qui correspond à parcourir un chemin du nœud racine à un nœud feuille de l'arbre MCTS et à effectuer une simulation jusqu'à ce qu'un nœud terminal de l'arbre de recherche soit atteint (lorsque tous les sommets sont colorés).

Algorithme 7 Algorithme du MCTS

Input : Graphe pondéré $G = (V, E, w)$ et borne de couleurs à utiliser $\bar{k}$
Output : Meilleure solution S^* trouvée
$S^* = \emptyset$ and $f(S^*) = MaxInt$
tant que la condition de fin n'est pas atteinte **faire**
    $C \leftarrow R$ ▷ nœud courant correspondant au nœud racine de l'arbre
    $S \leftarrow \{V_1, U\}$ with $V_1 = \{v_1\}$ and $U = V \backslash V_1$ ▷ Initialisation de la solution avec le premier sommet
    /* Sélection */ ▷ Section 5.3.2
    tant que C n'est pas une feuille **faire**
        $C \leftarrow$ UCT(C) avec un mouvement légal $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$
    /* Expansion */ ▷ Section 5.3.3
    si C a un enfant potentiel, pas encore ouvert **alors**
        $C \leftarrow$ ajout_enfant(C) avec un mouvement légal $< u, U, V_i >$ tel que $i \leq \bar{k}$
        $S \leftarrow S \oplus < u, U, V_i >$
    /* Simulation */ ▷ Section 5.3.4
    complète_solution_partielle(S)
    /* Mise à jour */ ▷ Section 5.3.5
    tant que $C \neq R$ **faire**
        mise_à_jour(C,f(S))
        $C \leftarrow$ parent(C)
    si $f(S) < f(S^*)$ **alors**
        $S^* \leftarrow S$
        /* Élagage */ ▷ Section 5.3.6
        Application des règles d'élagage
retourne S^*

Chaque itération de l'algorithme MCTS implique l'exécution de 5 étapes pour explorer l'arbre de recherche avec des mouvements légaux (cf. Section 5.2) :

1. **Sélection** À partir du nœud racine de l'arbre MCTS, des nœuds enfants successifs sont sélectionnés jusqu'à ce qu'un nœud feuille soit atteint. Le processus de sélection équilibre le compromis exploration-exploitation. Le score d'exploitation est lié au score moyen obtenu après avoir sélectionné ce nœud enfant et est utilisé pour guider l'algorithme vers une partie de l'arbre où les scores sont les plus bas (problème de minimisation). Le score d'exploration est lié au nombre de visites du nœud enfant et incitera l'algorithme à explorer de nouvelles parties de l'arbre.

2. **Expansion** L'arbre MCTS se développe en ajoutant un nouveau nœud enfant au nœud

feuille atteint lors de la phase de sélection, tout en respectant la borne supérieure globale $\bar{k}$ sur le nombre de couleurs à utiliser pour construire une solution optimale.

3. **Simulation** À partir du nouveau nœud ajouté, la solution partielle actuelle est complétée avec des mouvements légaux, au hasard ou en utilisant des heuristiques (sans contrainte globale sur le nombre de couleurs à utiliser).
4. **Mise à jour** Après la simulation, le score moyen et le nombre de visites de chaque nœud sur la branche explorée sont mis à jour.
5. **Élagage** Si un nouveau meilleur score est trouvé, certaines branches de l'arbre MCTS peuvent être élaguées, s'il n'est pas possible d'améliorer le meilleur score actuel avec celui-ci. De plus, les branches sont limitées en choix par les coups légaux possibles, le nombre de couleurs déjà utilisées et la borne globale sur le nombre de couleurs.

L'algorithme se poursuit jusqu'à ce qu'il ne reste plus de nœuds enfants à développer dans l'arbre MCTS, ce qui signifie que l'arbre de recherche a été entièrement exploré. Dans ce cas, le meilleur score trouvé est prouvé optimal. Ou, jusqu'à ce qu'un temps limite soit atteint. Le meilleur score trouvé jusqu'à présent est alors retourné.

5.3.2 Sélection

La sélection commence à partir du nœud racine de l'arbre MCTS et sélectionne les nœuds enfants jusqu'à ce qu'un nœud feuille soit atteint. À chaque niveau t de l'arbre MCTS, si le nœud courant C_t correspond à une solution partielle $S = \{V_1, ..., V_k, U\}$ avec t sommets déjà colorés et k couleurs utilisées, il existe l mouvements légaux possibles, avec $1 \leq l \leq k+1$. Par conséquent, à partir du nœud C_t, l enfants potentiels $C^1_{t+1}, \ldots, C^l_{t+1}$ peuvent être sélectionnés.

Si $l > 1$, la sélection du nœud enfant le plus prometteur peut être vue comme un problème de bandit à plusieurs bras [Lai *et al.*, 1985] avec l leviers. Ce problème de choix du prochain nœud peut être résolu avec l'algorithme UCT pour la recherche d'arbre de Monte Carlo en sélectionnant l'enfant avec la valeur maximale de l'expression suivante [Jooken *et al.*, 2023] :

$$normalized\_score(C^i_{t+1}) + c \times \sqrt{\frac{2 * ln(nb\_visits(C_t))}{nb\_visits(C^i_{t+1})}}, \text{ for } 1 \leq i \leq l. \tag{5.3}$$

Ici, $nb\_visits(C)$ correspond au nombre de fois où le nœud C a été sélectionné. c est un coefficient réel positif permettant d'équilibrer le compromis entre l'exploitation et l'exploration. Il est par défaut fixé à la valeur de 1. Une analyse de sensibilité de cet hyperparamètre est présentée

en Section 5.4.2. $normalized\_score(C^i_{t+1})$ correspond à un score normalisé du nœud enfant C^i_{t+1} ($1 \leq i \leq l$) donné par :

$$normalized\_score(C^i_{t+1}) = \frac{rank(C^i_{t+1})}{\sum_{i=1}^{l} rank(C^i_{t+1})},$$

où $rank(C^i_{t+1})$ est défini comme le rang entre 1 et l des nœuds C^i_{t+1} obtenus en les triant selon leurs valeurs moyennes $avg\_score(C^i_{t+1})$ (les nœuds qui semblent plus prometteurs obtiennent un score plus élevé). $avg\_score(C^i_{t+1})$ est le score moyen sur la sous-branche avec le nœud C^i_{t+1} sélectionné obtenu après toutes les simulations précédentes.

5.3.3 Expansion

Depuis le nœud C de l'arbre MCTS atteint pendant la procédure de sélection, un nouvel enfant de C est ouvert et son mouvement légal correspondant est appliqué à la solution courante. Parmi les enfants non ouverts, le nœud associé au plus petit nombre de couleurs i est sélectionné. Par conséquent, le nœud enfant nécessitant la création d'une nouvelle couleur (et augmentant le score) sera sélectionné en dernier. Aussi, les possibilités de couleurs associées aux nœuds enfants sont limitées au minimum entre le degré du sommet plus une couleur, et la borne supérieure globale $\bar{k}$ du nombre de couleurs.

5.3.4 Simulation

La simulation prend en entrée la solution partielle et légale courante trouvée après la phase d'expansion. Elle colore les sommets restants pour obtenir une solution légale. Cette phase ne prend pas en compte la limite globale $\bar{k}$ sur le nombre de couleurs à utiliser de façon à garantir systématiquement la possibilité de compléter les solutions sans conflit. Dans l'algorithme MCTS original, la simulation consiste à choisir des mouvements aléatoires dans l'ensemble de tous les mouvements légaux $\mathcal{L}(S)$ défini par les Équations 5.1 et 5.2 jusqu'à ce que la solution soit complétée. Nous appelons cette première version *MCTS+Random* (ou MCTS+R). Comme le montre la section expérimentale, cette version n'est pas très efficace, car le nombre de couleurs augmente rapidement. Par conséquent, nous proposons d'autres procédures de simulation :

- un algorithme glouton contraint qui choisit un mouvement légal en priorisant les mouvements qui n'augmentent pas localement le score de la solution partielle courante

$S = \{V_1, ..., V_k, U\}$:

$$\mathcal{L}^g(S) = \{< u, U, V_i >, 1 \leq i \leq k, \forall v \in V_i, (u, v) \notin E\} \tag{5.4}$$

Il choisit le mouvement $< u, U, V_{k+1} >$, consistant à ouvrir un nouveau groupe de couleurs et augmentant le score courant de $w(u)$, seulement si $\mathcal{L}^g(S) = \emptyset$. Nous appelons cette version *MCTS+Greedy-Random* (ou MCTS+GR).

- un algorithme glouton déterministe qui choisit toujours un mouvement légal dans $\mathcal{L}(S)$ (cf. équation (5.1) et (5.2)) avec la première couleur disponible i. Nous appelons cette version *MCTS+Greedy* (ou MCTS+G). Il est le plus proche du fonctionnement du glouton de [Welsh et Powell, 1967].
- basé sur le fonctionnement de *DSatur* de [Brélaz, 1979], nous proposons une légère modification pour le WVCP consistant à prioriser le poids avant la saturation lors du prochain sommet à colorer.
- basé sur le fonctionnement de *RLF* de [Leighton, 1979], la modification pour le WVCP consiste, cette fois aussi, à considérer le poids du prochain sommet à colorer en plus de maximiser le nombre de voisins déjà impossibles à colorer avec la couleur courante.

Dans le cas du GCP, les cinq versions présentées ici fonctionnent en considérant des poids à 1, les sommets sont alors simplement triés par degré. Les algorithmes *DSatur* de [Brélaz, 1979] et *RLF* de [Leighton, 1979] présentés dans le Chapitre 2 fonctionnent comme les originaux.

5.3.5 Mise à Jour

Une fois la simulation terminée, une solution complète du WVCP est obtenue. Si cette solution est meilleure que la meilleure solution enregistrée trouvée jusqu'à présent S^* (c'est-à-dire $f(S) < f(S^*)$), S devient la nouvelle meilleure solution globale S^*.

Ensuite, une procédure de rétropropagation met à jour chaque nœud C de toute la branche de l'arbre MCTS qui a conduit à cette solution :

- le score moyen de chaque nœud C de la branche est mis à jour avec le score $f(S)$:

$$avg\_score(C) \leftarrow \frac{avg\_score(C) \times nb\_visits(C) + f(S)}{nb\_visits(C) + 1} \tag{5.5}$$

- le compteur de visites $nb\_visits(C)$ de chaque nœud de la branche est augmenté de 1.

5.3.6 Élagage

Pendant une itération de MCTS, trois règles d'élagage sont appliquées :

1. pendant l'expansion : si le score $f(S)$ de la solution partielle associée à un nœud visité pendant cette itération de MCTS est égal ou supérieur au score actuel $f(S^*)$ de la meilleure solution trouvée, alors le nœud est supprimé, car le score d'une solution partielle ne peut pas diminuer lorsque plus de sommets sont colorés.
2. quand le meilleur score $f(S^*)$ est trouvé, l'arbre est *nettoyé*. Une heuristique parcourt tout l'arbre et supprime les nœuds enfants déjà ouverts ou pouvant l'être en un coup lors de la phase d'expansion, et associés à un score partiel $f(S)$ égal ou supérieur au meilleur score $f(S^*)$.
3. si un nœud est *complètement exploré*, il est supprimé et ne sera plus exploré dans l'arbre MCTS. Un nœud est dit *complètement exploré* s'il s'agit d'un nœud feuille sans enfants, ou si tous ses enfants ont déjà été ouverts une fois et ont tous été supprimés. Notez que cette troisième étape d'élagage est récursive, car la suppression d'un nœud peut entraîner la suppression de son parent s'il n'a plus d'enfants, et ainsi de suite. On notera aussi que la suppression des nœuds avec cette règle est accentuée si la borne globale $\bar{k}$ sur le nombre de couleurs à utiliser est la plus basse possible, car elle limite potentiellement le nombre d'enfants ouverts pour chaque feuille.

Les trois règles d'élagage et le fait que les symétries soient coupées dans l'arbre en restreignant l'ensemble des mouvements légaux considérés à chaque étape (cf. Section 5.2.2) offrent la possibilité d'explorer tout l'arbre en un temps raisonnable pour les petites instances. Cette particularité de l'algorithme permet d'obtenir une preuve d'optimalité pour de telles instances.

5.3.7 Exemple d'Exploration

La Figure 5.1 affiche une itération de MCTS pour le WVCP sur un petit graphe à sept sommets nommés A–G avec différents poids entre 2 et 9. Sur chaque diagramme est affiché l'état actuel de la solution de coloration partielle en cours de construction (à droite) et l'état actuel de l'arbre de recherche (à gauche). Dans l'arbre de recherche, chaque carré représente un nœud et le nombre en bas à droite d'un carré est le score de la solution partielle correspondante. En haut de chaque carré sont écrits le score moyen et le nombre de visites de chaque nœud. En plus du nœud racine (sommet A coloré en bleu), cinq nœuds ont déjà été ouverts dans l'arbre de recherche (cinq itérations de MCTS).

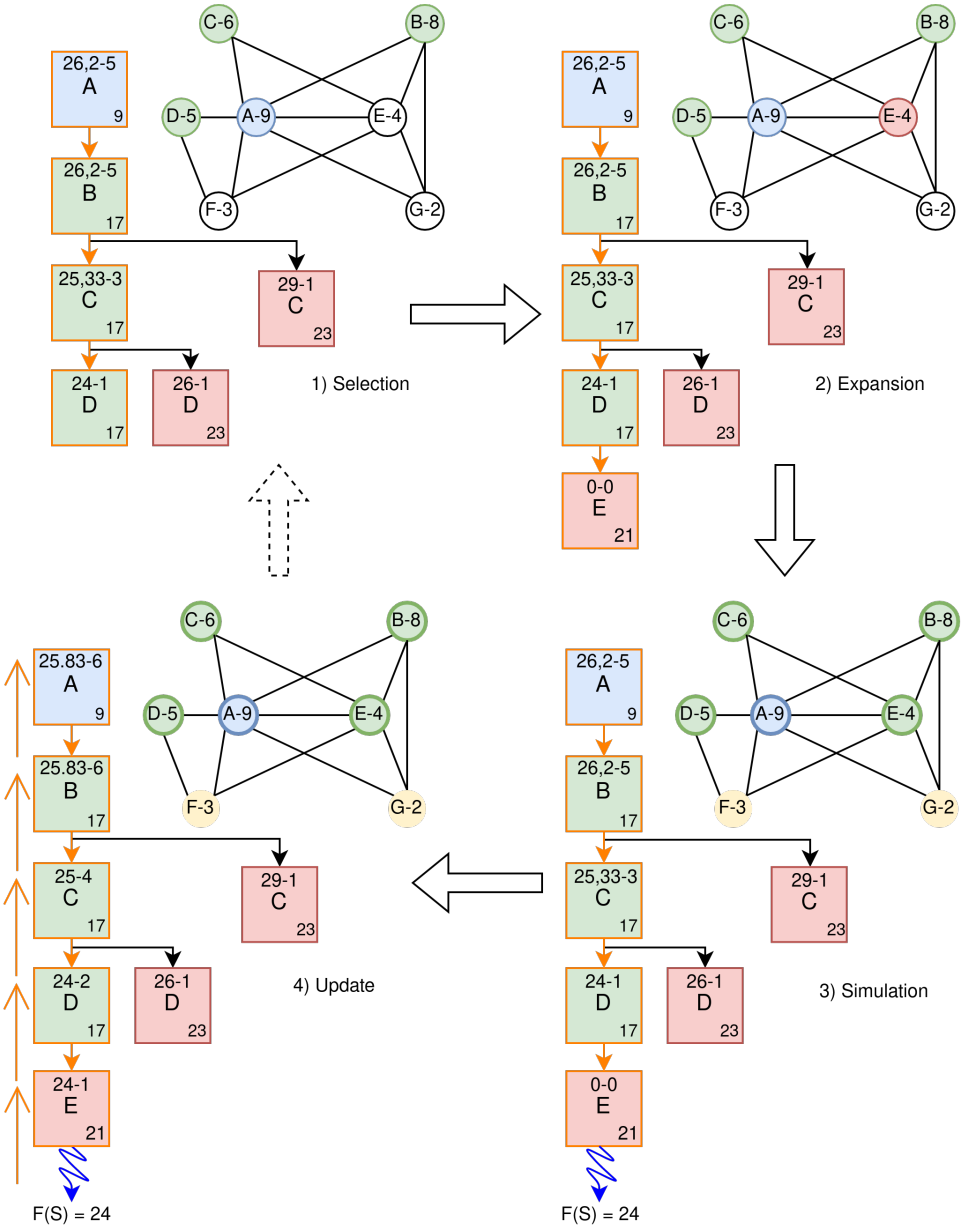


FIGURE 5.1 – Exemple d'une itération de MCTS.

La sixième itération de MCTS se déroule comme suit :

- **Sélection** : à partir du nœud racine, le seul enfant possible correspondant au sommet B en vert est sélectionné. De là, il y a deux options car le sommet C peut être coloré en vert ou en rouge. L'option la plus intéressante est choisie (sommet C en vert) en considérant le score et le nombre de visites de chaque enfant (cf. Équation 5.3). Ensuite, la feuille la plus prometteuse est sélectionnée (D en vert).

- **Expansion** : à partir du nœud D en vert, un nouveau nœud est ajouté à l'arbre. Il correspond à E en rouge (car il ne peut pas prendre la couleur bleue ni verte).
- **Simulation** : à partir de là, la solution est complétée avec un algorithme glouton pour obtenir une solution légale complète avec un score de 24.
- **Mise à Jour** : ce score de 24 est rétropropagé sur la branche explorée (mise à jour du score moyen et du nombre de visites de chaque nœud de la branche).
- **Élagage** : Figure 5.2 présente l'état de l'arbre après quelques itérations. Comme le meilleur score trouvé est 24, toute branche de l'arbre avec un score supérieur ou égal à 24 est supprimée (indiquée par une croix rouge).

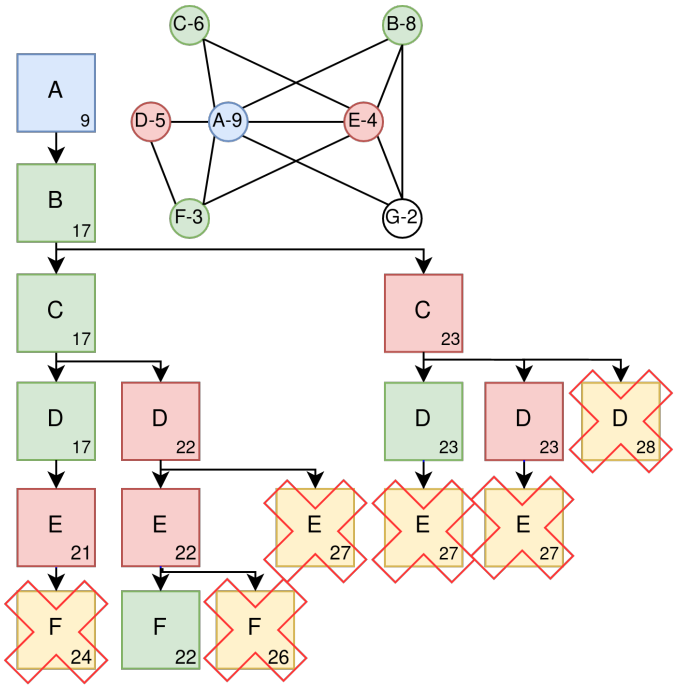


FIGURE 5.2 – Exemple de l'élagage de l'arbre de recherche.

5.4 Expérimentations

Cette section décrit d'abord les paramètres expérimentaux utilisés dans ce travail, puis une analyse du coefficient exploration versus exploitation est effectuée. Ensuite, nous analysons expérimentalement les impacts des différentes stratégies de coloration gloutonnes utilisées pendant la phase de simulation du MCTS sur le WVCP puis sur le GCP. Cette étude sur le GCP est nouvelle dans ce travail de thèse et n'était pas réalisée dans la publication originale [Grelier *et al.*, 2022] qui portait uniquement sur le WVCP. Nous l'avons ajoutée très récemment, notamment pour pouvoir comparer notre approche du MCTS à une version concurrente proposée par [Cazenave *et al.*, 2021] qui a été appliquée seulement sur le GCP.

5.4.1 Paramètres Expérimentaux

Nous utilisons les 188 instances réduites du WVCP et les 244 instances réduites du GCP présentées dans le Chapitre 3. Toutes les instances originales et réduites sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_instances`.

Tous les algorithmes présentés sont codés en C++, compilés et optimisés avec le compilateur g++ 12.1. Le code source ainsi que les résultats complets des algorithmes sur toutes les instances sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts`. Pour résoudre chaque instance, 20 exécutions indépendantes ont été effectuées sur un ordinateur équipé d'un processeur Intel Xeon ES 2630, 2,66 GHz avec une limite de temps d'une heure, sauf pour les tests de coefficient d'exploration par rapport à l'exploitation où 5 à 15 heures ont été utilisées. L'exécution de la procédure de benchmark DIMACS, dfmax [1], sur notre ordinateur a pris 8,94 secondes pour résoudre l'instance r500.5 en utilisant gcc 12.1 sans optimisation. Des changements dans les résultats peuvent être observés par rapport à la publication [Grelier *et al.*, 2022] dus à quelques optimisations dans le code, l'utilisation du flag LTO à la compilation et une réduction des instances plus complète. Les meilleurs scores ou scores optimaux sont aussi mis à jour au niveau de l'état de l'art du moment de l'écriture de la thèse et non au moment de la publication des articles.

Les résultats de l'algorithme NRPA présentés dans la Section 5.4.4 sont réalisés avec le code source de [Cazenave *et al.*, 2021] où 20 exécutions indépendantes d'une heure sont réalisées par instance. Si une solution légale avec un nombre de couleurs est trouvée, une nouvelle exécution d'une heure est réalisée avec une couleur de moins. Pour les résultats de *TabuCol*, ceux-ci sont

1. `http://archive.dimacs.rutgers.edu/pub/dsj/clique/`

obtenus en réduisant le nombre de couleurs pendant la recherche lorsqu'une solution légale est trouvée jusqu'à atteindre un score prouvé optimal avec une limite d'une heure de calcul.

Les 188 instances du WVCP sont séparées en quatre ensembles : (i) **pxx** : 35 instances pxx [Prais et Ribeiro, 2000], (ii) **rxx** : 30 instances rxx [Prais et Ribeiro, 2000], (iii) **DIMACS_easy** : 75 instances DIMACS et COLOR, dont 72 résolues à l'optimalité. Soit par l'algorithme exact MWSS [Cornaz *et al.*, 2017] rapporté dans [Nogueira *et al.*, 2021] (10 heures de recherche), ou par les modèles CP de [Goudet *et al.*, 2023] ou les variantes gloutonnes du MCTS, (iv) **DIMACS_hard**, 48 instances DIMACS difficiles qui n'ont jamais été résolues de manière optimale à l'exception de 5 d'entre elles, qui restent difficiles.

Les 244 instances du GCP sont séparées en trois ensembles, (i) **easy** : 183 instances plus simples, où un glouton est généralement proche de l'optimal, (ii) **medium** : 30 instances demandant plus de recherche qu'un glouton et (iii) **hard** : 31 instances difficiles à résoudre même avec un algorithme mémétique.

Pour toutes les différentes versions de l'algorithme MCTS, le coefficient c, équilibrant le compromis entre l'exploitation et l'exploration, est fixé à la valeur de 1 (cf. Équation 5.3). Une analyse de sensibilité de cet hyperparamètre important est menée dans la section suivante.

5.4.2 Analyse du Coefficient Exploitation vs Exploration

Un élément clé de l'algorithme MCTS est le coefficient c équilibrant le compromis entre exploration et exploitation dans l'Équation 5.3. Nous étudions ici l'importance de ce coefficient en le faisant varier et en présentant l'évolution du score au fil du temps. Pour cette expérimentation, nous avons fait varier le coefficient c de 0 (pas d'exploration) à 5 (encourager l'exploration). Pour chaque valeur de coefficient, nous avons effectué 20 exécutions de la variante MCTS+Greedy-Random par instance pendant 5h par exécution (15h pour les très grandes instances C2000.x).

La Figure 5.3 affiche 6 graphiques montrant l'évolution de la moyenne des meilleurs scores au fil du temps pour les instances DSJC500.5, latin_square_10, le450_25a, wap01a, C2000.5 et C2000.9 pour les différentes valeurs du coefficient c. Ces 6 instances proviennent de l'ensemble d'instances DIMACS_hard et peuvent être considérées comme très difficiles.

Quatre motifs observés pour ces instances retrouvés sur d'autres instances sont les suivants :

- P1 : les instances nécessitant beaucoup d'exploration,
- P2 : les instances nécessitant plus d'exploration que d'exploitation,
- P3 : les instances nécessitant plus d'exploitation que d'exploration,
- P4 : les instances nécessitant beaucoup d'exploitation.

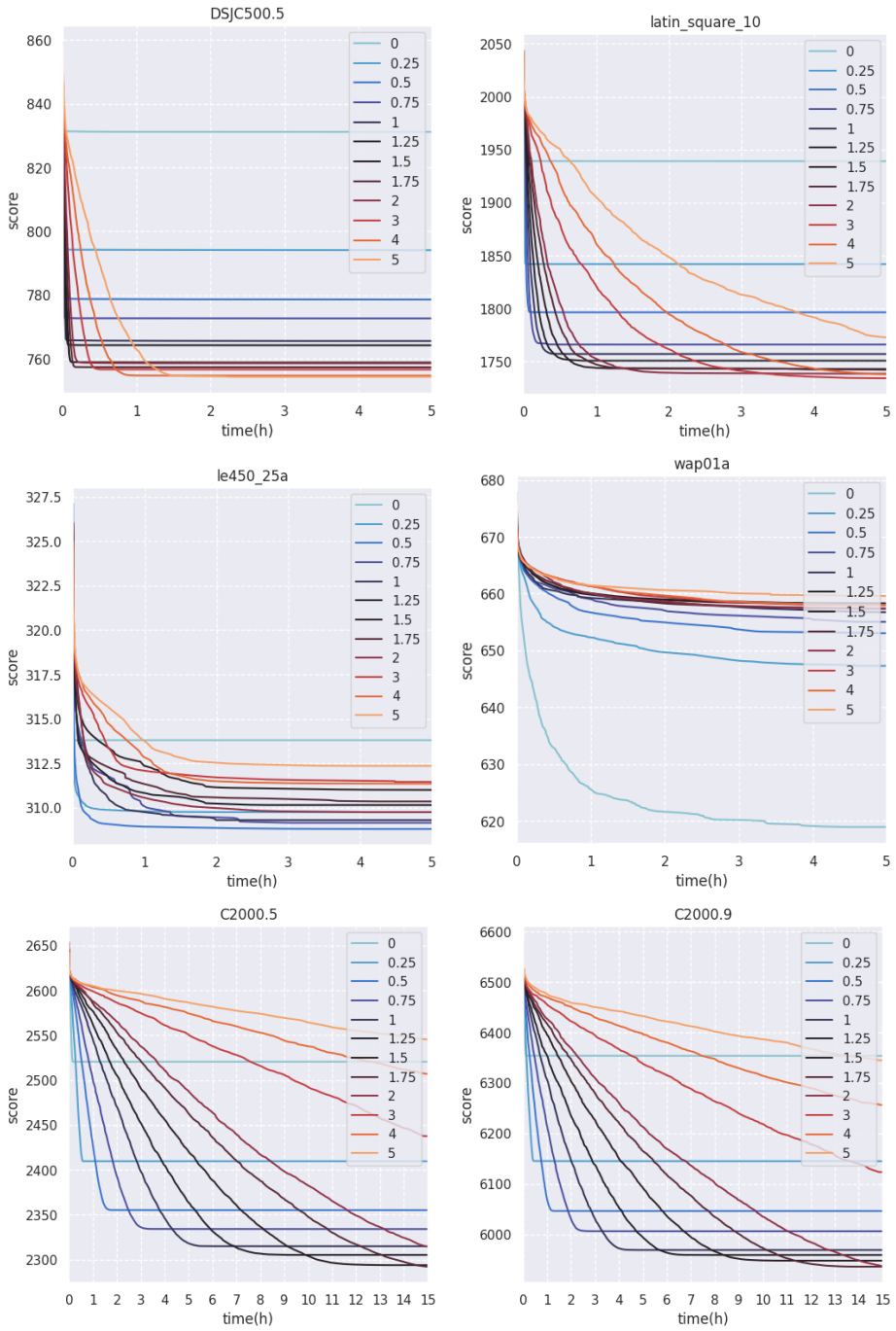


FIGURE 5.3 – Graphiques de l'évolution des moyennes des scores au fil du temps pour différentes valeurs du coefficient c entre 0 et 5, pour les instances DSJC500.5, latin_square_10, le450_25a, wap01a, C2000.5 et C2000.9. Pour chaque configuration, 20 exécutions sont lancées avec la variante MCTS+Greedy-Random pendant 5h et 15h pour les instances C2000.

Le premier motif P1 est observé pour l'instance DSJC500.5 et également pour les instances queen. Pour ces instances, le manque d'exploration conduit à de mauvais résultats, et de meilleurs résultats sont atteints à mesure que le coefficient c augmente. Le motif P2 est observé pour l'instance latin_square_10, mais aussi pour d'autres instances telles que flat1000 où le meilleur score obtenu en fonction du coefficient c a une forme de U, avec une valeur optimale de c entre 1 et 2. Ce phénomène peut également être observé sur des instances telles que C2000.5 et C2000.9 où il devient rapidement plus intéressant d'explorer jusqu'à un certain point. Le motif P3 trouvé pour l'instance le450 montre les meilleurs résultats lorsqu'il n'y a qu'une faible exploration, mais les résultats sont pires lorsque c est fixé à zéro.

En général, pour les motifs P1, P2 et P3, ne pas avoir d'exploration conduit rapidement à un piège de minimum local et il semble préférable de sécuriser un minimum de diversité pour atteindre un meilleur score, tandis que, pour le motif P4, trouvé pour les instances wap (instances très grandes), donner une chance à l'exploration conduit l'algorithme à être perdu dans l'espace de recherche. Pour les instances très grandes, comme l'arbre de recherche est très grand et ne peut pas être suffisamment exploré en raison de la limite de temps, il semble plus bénéfique pour l'algorithme de favoriser davantage l'intensification pour mieux rechercher une bonne solution dans une petite partie de l'arbre. En résumé, le coefficient d'exploration versus exploitation le plus adapté dépend donc de l'instance considérée. Trouver le bon coefficient est une tâche difficile. Par défaut, un coefficient $c = 1$ peut-être un bon compromis et nous l'utiliserons dans toute la suite de ce travail.

5.4.3 MCTS et Algorithmes Gloutons pour le WVCP

Le Tableau 5.1 résume les résultats du MCTS avec les heuristiques gloutonnes comme simulation (cf. Section 5.3.4) face aux recherches locales de l'état de l'art, *AFISA* [Sun *et al.*, 2018], *RedLS* [Wang *et al.*, 2020] et *ILS-TS* [Nogueira *et al.*, 2021] détaillées dans le Chapitre 2. Les en-têtes de colonne montrent les ensembles d'instances et leur taille (une étoile est ajoutée si les instances de l'ensemble ont été résolues de manière optimale), la dernière est un résumé de tous les ensembles d'instances. Les lignes montrent ensuite pour chaque méthode, Glouton (Random/R, Greedy-Random/GR, Greedy/G, *DSatur* et *RLF*), MCTS (MCTS+Glouton) et les recherches locales, le nombre d'instances dont le meilleur score connu est atteint. Les versions MCTS+Glouton prouvent aussi quelques optimalités pour les instances pxx et DIMACS_easy. Les meilleurs scores proviennent de l'état de l'art, en particulier [Nogueira *et al.*, 2021] (1h de calcul) et [Goudet *et al.*, 2022] (plusieurs jours de calcul), ou d'autres articles publiés pendant cette thèse.

	pxx 35*	rxx 30*	DIMACS easy 75 - 72*	DIMACS hard 48 - 5*	Total 188 - 142*
R	2	0	3	0	5
MCTS+R	34 – 26*	0	41 – 22*	0	75 – 48*
GR	14	0	15	0	29
MCTS+GR	35 – 26*	22	57 – 23*	0	114 - 49*
G	14	3	10	0	27
MCTS+G	35-26*	11	46 – 22*	0	92 – 48*
DSatur	14	2	11	0	27
MCTS+*DSatur*	35-25*	11	47 – 22*	0	93 – 47*
RLF	2	0	3	0	5
MCTS+*RLF*	33-25*	1	36 – 20*	0	70 – 45*
AFISA	35	5	71	3	114
RedLS	31	3	70	8	112
ILS-TS	35	30	75	19	159

TABLE 5.1 – Résumé du nombre de fois où le meilleur score connu est atteint pour chaque algorithme. Les valeurs avec une étoile indiquent le nombre de fois où un score a été prouvé optimal.

Tout d'abord, nous observons que toutes les variantes de MCTS dominent les algorithmes gloutons (R, G, GR, *DSatur* et *RLF*) utilisés seuls, mettant ainsi en évidence un premier apport du MCTS.

Avec les différentes variantes des algorithmes MCTS, toutes les instances pxx sont résolues à l'exception de MCTS+R et MCTS+*RLF*. On remarquera aussi une difficulté pour *RedLS* sur ces instances ainsi que pour les instances rxx, qui sont plus difficiles à résoudre pour le MCTS, sauf pour la version MCTS+GR. Les instances de DIMACS_easy sont partiellement résolues par chaque variante MCTS. Les instances de DIMACS_hard montrent une réelle difficulté pour toutes ces variantes MCTS, car aucune méthode n'atteint les meilleurs scores de la littérature. Au total, *ILS-TS* obtient le plus de meilleurs scores connus suivi par *AFISA* et MCTS+GR.

Pour mieux comparer ces différents algorithmes, et ne pas nous fier uniquement au nombre de meilleurs scores connus atteints, nous avons effectué des comparaisons par paires entre les algorithmes basés sur les scores moyens obtenus et une analyse statistique sur chaque instance avec un test de Wilcoxon résumés dans le Tableau 5.2. Les nombres de chaque ligne correspondent au nombre d'instances pour lesquelles la méthode est significativement meilleure qu'une autre (avec un maximum de 188 instances). Une méthode est dite significativement meilleure qu'une autre sur une instance donnée si son score moyen mesuré sur 20 exécutions est significativement meilleur (un test de rang signé de Wilcoxon non paramétrique avec une valeur $p \leq 0,001$).

	R	MCTS+R	GR	MCTS+GR	G	MCTS+G	*DSatur*	MCTS+*DSatur*	*RLF*	MCTS+*RLF*	*AFISA*	*RedLS*	*ILS-TS*
R	-	0	0	0	0	0	0	0	10	0	0	0	0
MCTS+R	**186**	-	**169**	0	**92**	1	**90**	1	**164**	32	22	46	0
C	**186**	10	-	0	4	0	7	0	**103**	10	0	0	0
MCTS+GR	**186**	**122**	**178**	-	**151**	23	**152**	32	**178**	**92**	**77**	**85**	0
D	**186**	36	**133**	10	-	0	37	1	**167**	44	30	43	0
MCTS+G	**186**	**121**	**178**	**44**	**158**	-	**160**	30	**183**	**97**	**98**	**103**	1
DSatur	**186**	40	**139**	9	**47**	0	-	0	**169**	45	31	41	0
MCTS+*DSatur*	**186**	**121**	**178**	**48**	**157**	**35**	**158**	-	**183**	**99**	**93**	**107**	0
RLF	**166**	12	37	5	8	0	7	0	-	0	7	4	0
MCTS+*RLF*	**184**	**87**	**163**	14	**116**	14	**116**	14	**183**	-	58	79	0
AFISA	**186**	**96**	**173**	33	**141**	46	**140**	50	**176**	**89**	-	**53**	0
RedLS	**186**	**79**	**155**	51	**136**	60	**137**	57	**179**	**88**	41	-	14
ILS-TS	**186**	**126**	**178**	**101**	**161**	**94**	**161**	**95**	**183**	**118**	**104**	**96**	-

TABLE 5.2 – Comparaison entre toutes les variantes gloutonnes et MCTS. Par exemple, la ligne pour MCTS+R(Random) signifie que la méthode est meilleure sur 186 instances par rapport à la procédure aléatoire (R), sur 169 instances par rapport à la procédure gloutonne aléatoire (GR), et jamais par rapport à MCTS+Greedy-Random (MCTS+GR). Les valeurs en gras mettent en évidence la valeur la plus élevée entre deux méthodes, la variante MCTS+Greedy est plus souvent (44 fois) significativement meilleure que MCTS+Greedy-Random (qui est meilleure que MCTS+Greedy 23 fois).

Le Tableau 5.2 met en évidence le nombre d'instances où une méthode est significativement meilleure qu'une autre. Pour les heuristiques gloutonnes, celle purement aléatoire (R) est totalement dominée par toutes les méthodes à l'exception de *RLF* pour qui 10 instances restent plus difficiles. Notre implémentation de *RLF* pour le WVCP ne semble pas obtenir de bons résultats face aux autres méthodes de manière générale, car la prise en compte des poids dans l'algorithme est délicate à mettre en parallèle avec la sélection des nœuds déjà présente dans l'algorithme d'origine. À l'inverse, notre implémentation de *DSatur* est meilleure que les autres gloutons. Pour les combinaisons du MCTS+Glouton, la variante MCTS+*DSatur* est significativement meilleure que les autres, suivie par MCTS+Greedy. Il semble que pour le WVCP, forcer les sommets les plus lourds à être regroupés dans les premières couleurs tout en considérant le nombre de leurs voisins déjà colorés permette une meilleure organisation des groupes de couleurs. Cela est particulièrement vrai pour les plus grandes instances, où le choix de mouvements aléatoires dans l'ensemble de tous les mouvements légaux n'est pas très efficace, car

le nombre de groupes de couleurs augmente rapidement. Cela explique également pourquoi la variante MCTS+Random fonctionne mal sur les instances plus grandes ou plus denses comme les instances rxx ou certaines instances difficiles de DIMACS.

Cependant, avec la simulation déterministe de MCTS+Greedy ou MCTS+*DSatur*, il n'y a pas d'échantillonnage des mouvements légaux comme dans la variante MCTS+Greedy-Random permettant une plus grande exploration de l'espace de recherche et une meilleure estimation des branches les plus prometteuses de l'arbre de recherche. Cette particularité de la variante MCTS+Greedy-Random nous permet de trouver le BKS pour plus d'instances. Par exemple, en regardant le Tableau 5.3, on peut voir que l'instance R75_1gb de l'ensemble DIMACS_easy est prouvée optimale par MCTS+Greedy-Random, mais pas par MCTS+Greedy. Avec l'aide stochastique, la version MCTS+Greedy-Random peut atteindre le meilleur score connu de 70, ce qui conduit à un élagage précoce de l'arbre qui permet de prouver l'optimalité plus tôt.

Le Tableau 5.3 résume les résultats des meilleures versions du MCTS et des recherches locales pour un panier d'instances représentatives des instances de référence de l'état de l'art.

Pour chaque instance présentée, la colonne BKS (*Best Known Score*) rappelle les meilleurs scores de la littérature, une étoile est indiquée si le score est optimal ensuite les résultats de chaque algorithme sont présentés, d'abord le meilleur score atteint sur les 20 exécutions ("best"), la moyenne ("mean") si celle-ci est différente du meilleur score, et le temps pour trouver le meilleur score ("time"). Le pied de tableau est un résumé des performances des méthodes sur les 188 instances de l'état de l'art. #BKS correspond au nombre de fois où la méthode atteint les meilleurs scores connus, #Best le nombre de fois où le meilleur score est le meilleur parmi les méthodes présentées, #Best Avg le nombre de fois où la moyenne est la meilleure et #Optimal le nombre d'instances pour lesquelles le score est prouvé optimal. Les instances affichées sont majoritairement celles de l'ensemble DIMACS_hard et quelques instances parmi les trois autres ensembles.

Nous pouvons remarquer que *ILS-TS* et *RedLS*, actuellement les meilleures recherches de l'état de l'art pour le WVCP, obtiennent tous les deux de très bons résultats sur ces instances difficiles où le MCTS peine à obtenir des solutions de bonne qualité. Celui-ci est cependant meilleur que *AFISA* et *RedLS* sur les instances rxx qui restent difficiles. Généralement, *ILS-TS* obtient les meilleurs résultats sauf pour une poignée d'instances difficiles comme latin square, flat, quelques DSJC ou les C2000. Face à ces méthodes, le MCTS a l'avantage de pouvoir prouver l'optimalité de certaines instances bien qu'il ne soit pas vraiment conçu dans cette optique. Ces preuves d'optimalité apparaissent principalement pour des instances de petite taille.

instance	BKS	MCTS+GR			MCTS+G			MCTS+*DSatur*			*AFISA*			*RedLS*			*ILS-TS*		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	2505	2537.7	3422	2385		1685	2397	2398.8	3390	2384	2403.4	3601	**2167**	**2193.8**	2403	2237	2266.4	3498
C2000.9	5477	6233	6272.9	3575	6125	6147.8	3238	6275		163	6582	6650.1	0	**5502**	**5528.1**	3303	5910	5969.9	3587
DSJC125.1g	23*	25	25.4	22	25		0	25		0	**23**	24.4	13	**23**	23.4	0	**23**		2
DSJC125.5g	71	74	75.3	52	77		0	74		67	**71**	72.2	1360	72		257	**71**		126
DSJC125.9g	169*	170	172.7	25	171		1630	171		21	170	174.1	2	**169**		0	**169**		182
DSJC250.1	127	134	141.4	13	141		4	139		1151	129	133.6	3294	130	131.6	1	**127**	**127.8**	1608
DSJC250.5	392	422	429.4	14	427		92	421		705	411	424.2	541	398	401.2	103	**393**	**397.6**	2567
DSJC250.9	934*	973	988.7	2793	986		16	984		559	949	976.1	232	**934**	**935.6**	718	936	942.1	3053
DSJC500.1	184	203	208.3	148	203		638	203		2943	198	201.5	1817	**187**	201.9	537	188	**188.8**	1744
DSJC500.5	685	754	765.6	136	755		635	775	780.9	3542	762	778.1	1307	**706**	**716.1**	2840	724	735.5	1744
DSJC500.9	1662	1771	1787.4	113	1794		171	1795	1797.1	3453	1744	1775.5	652	**1670**	**1675.1**	945	1720	1742	3039
DSJC1000.1	300	334	337.4	1618	333		2823	340		2474	319	325	2182	**305**	**307.2**	1235	**305**	307.4	1574
DSJC1000.5	1185	1271	1293.6	1668	1318		1437	1338		203	1308	1330	3531	**1198**	**1213.7**	2381	1245	1269.2	231
DSJC1000.9	2836	3040	3070.4	978	3078		2863	3172		3338	3066	3107	3601	**2840**	**2858.5**	2953	3026	3066.8	3580
DSJR500.1	169*	**169**		46	177		0	176		21	**169**		54	171	184.5	0	**169**		0
flat1000_50_0	924	1236	1255.8	1452	1251	1251.1	3037	1303		1843	1267	1293.3	2736	**1155**	**1173.8**	3238	1222	1235	1712
flat1000_60_0	1162	1275	1295.8	1478	1260	1260.7	3424	1343		311	1309	1323.2	3584	**1191**	**1205.7**	1080	1250	1270	125
flat1000_76_0	1165	1252	1269.7	1477	1244	1248.2	3557	1313		1798	1288	1304.4	3278	**1176**	**1194**	1107	1232	1247.5	1198
GEOM120a	105*	106	106.6	330	**105**		1129	109		5	**105**	106.6	136	**105**	109.2	9	**105**		0
GEOM120b	35*	37		0	37		498	37		110	**35**	36.2	422	**35**	35.5	0	**35**		0
GEOM120	72*	**72**	73	0	**72**		0	**72**		36	**72**	73	0	**72**	75.2	0	**72**		0
inithx.i.1	569*	**569**		0	**569**		0	**569**		0	**569**	573.1	0	**569**		11	**569**		16
inithx.i.2	329*	**329**	329.9	1720	330		0	331		0	334	337.6	448	**329**	329.1	311	**329**		8
inithx.i.3	337*	**337**	337.7	1449	339		0	339		0	340	346.4	1184	**337**	341.6	75	**337**		29
latin_square_10	1480	1721	1757.1	966	1726	1726.8	3418	1805		2360	1607	1652.4	2257	**1505**	**1523**	2369	1559	1581.2	1368
le450_25a	306	307	310	3364	312		5	310		1919	311	316.3	2699	**306**	307.4	503	**306**		174
le450_25b	307*	309	309.1	993	309		0	309		224	308	312.6	1891	**307**	313.4	56	**307**		10
le450_25c	342	365	372.7	2844	369		950	376		1572	355	364.4	3436	351	354.8	43	**348**	**351.8**	3207
le450_25d	330	354	358.8	2802	364		1657	369		1411	351	357.8	1630	**332**	**338.9**	154	339	342.5	1999
myciel5g	22*	**22***		0	**22***		0	**22***		0	**22**	22.1	11	**22**	22.1	34	**22**		0
myciel6gb	94*	96	98	1045	98		0	99		0	**94**		42	**94**	95	0	**94**		2
myciel6g	26*	**26**		0	27		0	27		0	**26**	26.4	268	**26**		15	**26**		0
myciel7gb	109*	117	118.3	476	**109**		42	**109**		38	**109**	111.8	1129	**109**	116.4	0	**109**		4
queen10_10	162	165	169.1	986	171		0	169		8	165	166.9	524	**162**	166.2	504	**162**		13
queen11_11	172	178	180.4	941	180		80	179		50	179	181.1	722	174	177.1	761	**172**	**172.6**	1820
queen12_12	185	189	196.8	2	193		1102	197		1823	193	196.7	2062	188	190.2	7	**185**	**186.1**	1261
queen13_13	194	204	207.6	1125	205		5	199		71	203	206.7	288	195	198.7	1528	**194**	**195.7**	3475
queen14_14	215	224	226.1	38	224		13	225		2351	224	230.5	3113	217	222.4	18	**216**	**217.3**	2018
queen15_15	223	237	241.3	216	239		49	238		315	236	240.2	1572	**227**	229.7	1109	**227**	**228.4**	2167
R75_1gb	70*	**70***	73.6	2679	76		184	75		9	**70**	73.7	0	**70**	78.3	0	**70**		0
R75_1g	18*	**18***	18.8	1164	**18***		956	**18***		1348	**18**		94	**18**	19.1	0	**18**		0
R75_5gb	186*	**186**	191.7	237	192		0	192		2234	**186**	188.6	28	**186**	192.1	0	**186**		1
R75_5g	51*	**51**	52.7	725	52		203	**51**		1956	**51**	51.9	17	**51**	51.3	108	**51**		0
R75_9gb	396*	**396**	397.9	1902	**396**		208	**396**		191	**396**	396.4	8	**396**		0	**396**		10
R75_9g	110*	**110**		39	**110**		7	**110**		202	**110**	110.2	3	**110**		0	**110**		0
wap01a	545	657	659.8	2965	599		3272	595		2904	653	664.5	3413	563	688.4	252	**548**	**552**	3263
wap02a	538	645	648.7	1301	588		3265	590		262	658	666.2	3416	552	586.5	589	**540**	**542.7**	2658
wap03a	562	746	749	803	653		1096	647		1805	767	788	0	**569**	**573.1**	2893	576	578.6	3102
wap04a	563	755	757.8	3085	649		603	643		828	773	792.9	0	**564**	**572.5**	3280	569	574.6	3529
wap05a	541	583	591.1	1867	566		172	565		2882	563	572.5	3378	543	544.5	970	**541**	**543.4**	3306
wap06a	516	591	595.4	2280	566		9	561		664	559	567.2	2859	**518**	590.4	1005	519	**522.4**	1123
wap07a	555	693	697.2	3186	635		165	639		986	670	683.2	3584	729	745.7	0	**564**	**569.5**	2680
wap08a	529	664	672	3539	612		2401	604		2583	654	665.3	3402	**536**	613.9	3035	543	**549.5**	3585
p32	2480*	**2480***		0	**2480***		0	**2480***		0	**2480**		0	**2480**	2480.5	0	**2480**		0
p41	2688*	**2688**		0	**2688**		0	**2688**		0	**2688**	2707.1	1225	2718	2787.3	0	**2688**		0
p42	2466*	**2466**	2466.8	33	**2466**		0	**2466**		4	**2466**	2474.8	1180	**2466**	2522.5	0	**2466**		4
r28	9407*	**9407**	9409.9	1530	**9407**		0	**9407**		0	9415	9511.4	1819	9410	9563	45	**9407**		8
r29	8693*	**8693**	8695.5	1316	8694		0	8694		3	8715	8799	379	8696	8817.6	0	**8693**		1
r30	9816*	**9816**	9819.6	11	9818		0	9818		0	9826	9960	2378	9836	9988.2	1	**9816**		5
#BKS			114			92			93			114			112			159	
#Best			114			92			93			114			131			170	
#Best Avg			75			91			93			45			47			171	
#Optimal			49			48			47			0			0			0	

TABLE 5.3 – Partie des résultats pour le WVCP des meilleures versions du MCTS face aux recherches locales de l'état de l'art sur un ensemble d'instances variées. En pied de tableau se trouvent les statistiques de nombre de BKS (*Best Known Score* - meilleurs scores connus dans l'état de l'art), nombre de fois où la méthode est la meilleure (#Best) ou obtient la meilleure moyenne (# Best Avg) et le nombre d'instances prouvées optimales (#Optimal) sur l'ensemble des 188 instances de l'état de l'art.

5.4.4 MCTS et Algorithmes Gloutons pour le GCP

Dans cette section, nous réalisons une étude de l'utilisation du MCTS dans le cas du GCP. Nous y comparons notamment l'utilisation du MCTS avec les différentes versions de glouton, y compris *DSatur* et *RLF* présentés dans le Chapitre 2 ainsi que *TabuCol* et NRPA (*Nested Rollout Policy Adaptation*) de [Cazenave *et al.*, 2021], un algorithme MCTS utilisant une technique de descente de gradient lors de sa politique de simulation pour mieux diriger la recherche, qui a récemment été proposé dans la littérature pour le GCP (cf. Section 2.3).

Le Tableau 5.4 donne un résumé des résultats sur les différentes instances de l'état de l'art découpées en trois ensembles. L'en-tête donne le nombre d'instances par ensemble suivi d'un second nombre avec une étoile indiquant le nombre d'instances pour lesquelles le nombre chromatique est connu. Nous pouvons constater que NRPA et MCTS+*RLF* sont capables de résoudre toutes les instances les plus faciles et que notre MCTS prouve l'optimalité pour une grande partie d'entre elles (109/183). Les instances medium sont résolues en grande partie par *TabuCol* et pour plus de la moitié par NRPA. En revanche les variantes du MCTS ont beaucoup plus de mal à résoudre ces instances. Seuls NRPA et *TabuCol* sont capables de résoudre respectivement 2 et 6 des instances les plus difficiles.

	easy 183 – 168*	medium 30 - 22*	hard 31 - 16*	Total 244 - 206*
R	75	0	0	75
MCTS+R	171 – 108*	1 – 1*	0	172 – 109*
C	152	0	0	152
MCTS+GR	178 – 108*	2 – 1*	0	180 – 109*
D	122	0	0	122
MCTS+D	177 – 109*	1 – 1*	0	178 – 110*
DSatur	148	0	0	148
MCTS+*DSatur*	180 – 109*	9	0	189 – 109*
RLF	170	1	0	171
MCTS+*RLF*	183 – 109*	7 – 1*	0	190 – 110*
NRPA	183	17	2	202
TabuCol	179	28	6	213

TABLE 5.4 – Résumé du nombre de fois où le meilleur score connu est atteint pour chaque algorithme.

	R	MCTS+R	GR	MCTS+GR	G	MCTS+G	*DSatur*	MCTS+*DSatur*	*RLF*	MCTS+*RLF*	NRPA	*TabuCol*
R	-	0	0	0	0	0	0	0	0	0	36	0
MCTS+R	**177**	-	**80**	0	**81**	2	50	0	17	1	43	9
C	**177**	34	-	0	47	0	19	0	2	0	42	9
MCTS+GR	**177**	**80**	**108**	-	**114**	20	**76**	10	29	2	45	10
D	**177**	51	**82**	7	-	0	23	0	1	0	44	9
MCTS+D	**177**	**78**	**108**	**24**	**119**	-	**83**	10	29	0	48	9
DSatur	**177**	**61**	**106**	15	**80**	6	-	0	0	0	44	9
MCTS+*DSatur*	**177**	**81**	**108**	**46**	**122**	**46**	**95**	-	**50**	10	**55**	12
RLF	**177**	**83**	**124**	**45**	**116**	**32**	**84**	19	-	0	**58**	9
MCTS+*RLF*	**177**	**80**	**108**	**55**	**122**	**60**	**94**	**34**	**66**	-	**63**	10
NRPA	**176**	**78**	**105**	**53**	**118**	**59**	**92**	40	57	28	-	11
TabuCol	**175**	**78**	**106**	**60**	**120**	**64**	**94**	**50**	**71**	**52**	**46**	-

TABLE 5.5 – Comparaison entre toutes les variantes gloutonnes et MCTS avec NRPA et *TabuCol* sur les 244 instances de l'état de l'art. Par exemple, la ligne pour MCTS+R signifie que la méthode est meilleure sur 177 instances par rapport à la procédure aléatoire (R), sur 80 instances par rapport à la procédure gloutonne aléatoire (GR), et jamais par rapport à MCTS+Greedy-Random (MCTS+GR). Les valeurs en gras mettent en évidence la valeur la plus élevée entre deux méthodes, la variante MCTS+Greedy(MCTS+G) est plus souvent (sur 24 instances) significativement meilleure que MCTS+Greedy-Random(MCTS+GR) (qui est meilleure que MCTS+Greedy sur 20 instances).

Le Tableau 5.5 résume les différences deux à deux sur toutes les méthodes gloutonnes, du MCTS avec la simulation gloutonne en question et avec NRPA et *TabuCol*. On y retrouve le nombre d'instances pour lesquelles la moyenne des résultats d'une méthode est significativement meilleure qu'une autre. Comme pour le WVCP, les méthodes MCTS+Glouton dépassent les versions avec glouton seul. Pour les gloutons, on peut remarquer que *RLF* est particulièrement efficace, même face aux versions avec le MCTS ou NRPA. Nous voyons aussi que *TabuCol* domine partiellement toutes les autres méthodes. MCTS+*RLF* est meilleur que NRPA sur 63 instances où l'inverse arrive 28 fois.

Les différences entre ces deux méthodes sont mises en avant dans le Tableau 5.6 qui présente une partie des résultats de *DSatur*, MCTS+DSatur, *RLF*, MCTS+*RLF*, de NRPA de [Cazenave *et al.*, 2021] et de *TabuCol* [Hertz et Werra, 1987].

instance	BKS	*DSatur*		MCTS+*DSatur*			*RLF*		MCTS+*RLF*		NRPA			*TabuCol*		
		best	time	best	mean	time	best	time	best	time	best	mean	time	best	mean	time
C2000.5	145	214	0	210	210.7	3569	195	163	192	2482	211	213.1	2596	**162**	**163.2**	1802
C2000.9	408	581	1	565		265	511	343	511	322	587	628.5	3185	**412**	**413**	2758
C4000.5	259	392	3	383	383.1	3342	356	1314	355	2621	397	401.4	2545	**304**	**305.3**	2303
DSJC125.1	5*	6	0	**5**		0	6	0	6	0	**5**		0	**5**		0
DSJC125.5	17*	24	0	19		16	20	0	18	2268	18		115	**17**		0
DSJC125.9	44*	55	0	46		193	49	0	45	144	**44**		171	**44**		0
DSJC250.1	8	11	0	9		0	10	0	9	0	**8**	8.1	1248	**8**		0
DSJC250.5	28	39	0	33		3108	34	0	32	235	31	31.2	2117	**28**		19
DSJC250.9	72*	98	0	79		973	83	0	79	387	75	77	3247	**72**		0
DSJC500.1	12	16	0	14		3084	15	0	14	3	13	13.6	1945	**12**		77
DSJC500.5	47	70	0	64		412	61	2	58	22	58	58.9	2372	**49**		476
DSJC500.9	126	175	0	159		2917	151	4	148	356	148	149.8	1197	**126**	**126.5**	2328
DSJC1000.1	20	28	0	26		1	24	3	23	1161	24		569	**21**		0
DSJC1000.5	82	123	0	116		200	108	19	105	2062	110	112	1414	**88**	**88.1**	1321
DSJC1000.9	222	316	0	303		1579	280	39	273	2188	291	294.9	2821	**224**	**225.1**	3181
DSJR500.1c	85*	97	0	89		48	90	0	87	148	118	123.5	0	106	113.9	0
DSJR500.1	12*	**12**	0	**12***		0	**12**	0	**12***	0	**12**	12.2	309	**12**		0
GEOM120a	16*	17	0	**16**		0	**16**	0	**16**	0	**16**		0	**16**		0
GEOM120b	16*	17	0	**16**		0	17	0	**16**	0	**16**		3	**16**		0
GEOM120	11*	**11**	0	**11***		0	**11**	0	**11***	0	**11**		211	**11**		0
latin_square_10	97	151	0	126		96	122	19	122	17	122	124.9	1758	**100**	**101.2**	1976
le450_15c	15*	26	0	22		3272	23	0	22	0	19	19.9	3136	**15**	**15.9**	1
le450_15d	15*	25	0	23		33	23	0	22	5	20	20.5	2266	**16**	**16.1**	48
le450_25c	25*	30	0	27		597	28	0	27	7	**26**		636	**26**		0
le450_25d	25*	30	0	27		201	28	0	27	4	**26**		958	**26**		0
queen10_10	11*	14	0	**11**		1524	13	0	12	0	**11**	11.2	1019	**11**		0
queen11_11	11*	16	0	13		0	14	0	13	0	**12**	12.9	1848	**12**		0
queen12_12	12*	16	0	14		30	15	0	14	0	14		2	**13**		0
queen13_13	13*	18	0	15		96	16	0	15	0	15		10	**14**		0
r1000.1c	98	124	0	108		1624	106	10	**103**	2514	139	171.3	17	134	155.4	86
r1000.5	234	278	0	246		3080	251	16	247	1674	**239**	**240.4**	2276	244	246	3404
R75_1g	4*	5	0	**4***		0	5	0	**4***	0	**4**		0	**4**		0
R75_5g	13	17	0	**13**		0	14	0	**13**	0	**13**		0	**13**		0
R75_9g	33*	38	0	**33**		0	**33**	0	**33**	0	**33**		2	**33**		0
R100_1g	5	6	0	**5**		0	6	0	**5**	0	**5**		0	**5**		0
R100_5g	14	20	0	16		3	16	0	15	4	**14**	14.6	1558	**14**		0
R100_9g	35*	45	0	37		3	39	0	36	5	36		33	**35**		0
wap01a	41*	54	0	44		3230	47	6	44	567	44	45	2839	**42**	**43.2**	1306
wap02a	40*	48	0	44		6	44	5	43	40	44	44.2	1869	**41**	**41.2**	362
wap03a	43	55	0	51		611	51	41	49	194	54	54.5	1577	**44**	**46.2**	1125
wap04a	41	49	0	46		119	47	40	45	187	48	48.8	1608	**42**	**43.3**	2997
wap06a	40*	45	0	42		529	42	1	42	1	41		880	**40**	41.5	3318
wap07a	41	48	0	44		203	46	7	44	144	43	43.4	2148	**42**	**42.6**	523
wap08a	40*	49	0	43		1515	46	7	44	169	42	42.8	1410	**41**	**41.8**	1846
#BKS		148		189			171		190		202			213		
#Best		148		189			171		191		206			235		
#Best Avg		148		189			171		192		159			229		
#Optimal		0		109			0		110		0			0		

TABLE 5.6 – Tableau de résultats pour un sous-ensemble d'instance pour le GCP des algorithmes gloutons *DSatur* et *RLF*, du MCTS avec les deux gloutons, NRPA et *TabuCol*. Le pied de tableau résume le nombre de meilleurs scores de la littérature atteints (#BKS), le nombre de fois où la méthode atteint les meilleurs scores (#Best) et moyennes (#Best Avg) ainsi que le nombre d'instances prouvées optimales (#Optimal) sur les 244 instances de l'état de l'art. Les moyennes égales au meilleur score rencontré ne sont pas affichées.

La première colonne indique le nom des instances et la deuxième colonne le meilleur score connu dans l'état de l'art (BKS - *Best Known Score*) celui-ci est suivi d'une "*" lorsque le score est optimal. Les colonnes 3-6 présentent les résultats de *RLF* et MCTS+*RLF*, les moyennes ne sont pas affichées, car *RLF* est déterministe et le MCTS n'est pas déterministe, mais si proche de l'être que les moyennes sont égales au meilleur score donc non affichées, une étoile est affichée si le MCTS a prouvé une instance optimale. Les colonnes 7-9 présentent les résultats de NRPA, une case vide dans les moyennes signifie que celle-ci est égale au meilleur score rencontré. Les quatre dernières lignes rappellent pour chaque méthode combien de fois le meilleur score connu est atteint ("#BKS"), le nombre de fois où la méthode obtient les meilleurs résultats ("#Best") et les meilleures moyennes ("#Best Avg") parmi les trois méthodes présentées ainsi que le nombre d'instances où l'optimalité est prouvée ("#Optimal"). Les temps sont affichés en secondes. Les valeurs en gras indiquent les meilleurs résultats par instance (sans prendre en compte le meilleur score connu). Les instances affichées comptent parmi les plus difficiles de la littérature.

On remarque que NRPA atteint plus de meilleurs scores connus et de meilleurs scores en général, mais en moyenne est dépassé par MCTS+*RLF*. Le MCTS dépasse systématiquement *RLF* sur toutes les instances, on remarquera cependant un temps d'exécution très long pour les grandes instances, par exemple *RLF* prend 343s et 1314s pour traiter C2000.9 et C4000.5. Notre MCTS et NRPA ne sont pas performants sur les mêmes instances, par exemple, NRPA est meilleur sur flat300_20_0, DSJR500.5, r1000.5, r250.5 ou school1_nsh qui sont des instances géométriques ou avec certains patterns, tandis que notre MCTS est meilleur sur des instances plus aléatoires, grandes ou denses comme C2000.5/9, C4000.5, r1000.1c, DSJR500.1c, DSJC1000.5/9, les flat1000 ou les plus grandes wap.

Les deux méthodes obtiennent de bons résultats sur toutes les instances de petites tailles ou simples et sur un bon nombre d'instances légèrement plus difficiles, mais peinent à obtenir de bons résultats sur les grandes instances les plus difficiles.

5.5 Conclusion

Dans ce travail, nous avons étudié la recherche arborescente de Monte Carlo appliquée au problème de coloration standard et au problème de coloration pondérée. Nous avons étudié différentes stratégies gloutonnes utilisées pour la phase de simulation.

Tout d'abord, le coefficient exploration vs exploitation utilisé lors de la sélection du MCTS a un impact très important sur les résultats de celui-ci. Nous avons découvert différents patterns sur des instances difficiles, certaines favorisant l'exploration, d'autres l'exploration. Dans le

cadre de nos tests, nous sommes restés sur une valeur plutôt neutre n'encourageant ni l'une ni l'autre.

Pour les instances les plus difficiles, que ce soit pour le GCP ou WVCP, le MCTS avec un algorithme glouton comme simulation montre rapidement des faiblesses face à des méthodes telles qu'une recherche locale, plus à même d'intensifier la recherche avec une solution complète, sans repartir d'une solution vide à chaque fois, comme dans le cas du MCTS. Pour des instances moyennes, le MCTS semble capable d'obtenir de bons résultats, qui sont dans le cas du GCP presque comparable au très récent algorithme NRPA [Cazenave *et al.*, 2021], et dans le cas du WVCP, comparable à des recherches locales. Pour des instances plus petites ou faciles, l'avantage du MCTS est de pouvoir échantillonner efficacement l'espace de recherche en particulier avec des procédure gloutonnes qui présente une part d'aléatoire. De plus, pour ces instances, notre algorithme MCTS est capable de fournir des preuves d'optimalité.

D'autres travaux futurs pourraient être envisagés. Par exemple, une étude intéressante consisterait à choisir automatiquement le coefficient d'équilibre entre l'exploitation et l'exploration au fur et à mesure de la résolution de chaque instance spécifique. Lors de l'exploration de l'arbre, d'autres stratégies pourraient être mises en place. Le nombre de couleurs disponibles pour un sommet pourrait être réduit selon le même type de règles que celles utilisées dans le cas de la réduction présentée dans le Chapitre 3. Une sélection plus dynamique du prochain sommet à colorer pourrait également être envisagée. En s'inspirant de *DSatur* [Brélaz, 1979], le prochain sommet à colorer pourrait être choisi en fonction du nombre de voisins déjà colorés, au lieu d'utiliser un ordre prédéfini des sommets à colorer dans l'arbre.

Dans l'article [Grelier *et al.*, 2022] nous avons aussi étudié la combinaison du MCTS avec une procédure de recherche locale comme simulation. Ce choix est guidé par le fait qu'après l'application d'un algorithme glouton, on obtient une solution satisfaisante, mais qu'une solution de bien meilleure qualité, et proche dans l'espace de recherche, pourrait être atteinte facilement par une recherche locale. Cet aspect sera présenté dans le prochain chapitre.

CHAPITRE 6

MCTS ET RECHERCHES LOCALES

Ce chapitre présente l'utilisation de recherches locales lors de la phase de simulation d'un arbre de recherche de Monte Carlo (MCTS) pour le WVCP. Nous étudions l'impact de quatre recherches locales dédiées au WVCP, *AFISA*, *RedLS* et *ILS-TS*, qui proviennent de l'état de l'art, ainsi qu'une nouvelle recherche locale que nous proposons, nommée *TabuWeight*. Des expériences sur des benchmarks du problème sont menées pour mettre en évidence les avantages et les limites de chaque couplage du MCTS avec une recherche locale spécifique.

Ce chapitre est la deuxième contribution présentée dans la conférence EVOCOP 2022 [Grelier *et al.*, 2022] qui fait suite au chapitre précédent.

6.1 Introduction

De nombreuses méthodes exactes et approchées ont été proposées dans la littérature pour résoudre le WVCP. [Malaguti *et al.*, 2009, Cornaz *et al.*, 2017, Goudet *et al.*, 2023] sont les plus récentes pour les méthodes exactes. Pour les méthodes approchées, nous retrouvons principalement des algorithmes de recherche locale [Nogueira *et al.*, 2021, Prais et Ribeiro, 2000, Sun *et al.*, 2018, Wang *et al.*, 2020] et un algorithme mémétique [Goudet *et al.*, 2022]. Ces heuristiques peuvent fournir des solutions approximatives de bonne qualité pour les instances de taille moyenne et grande, qui ne peuvent pas être résolues en un temps raisonnable par des méthodes exactes.

De par la définition du problème, l'espace de recherche du WVCP est composé de beaucoup de plateaux, la somme des sommets les plus lourds de chaque couleur efface l'importance des autres sommets du graphe. C'est pourquoi ces recherches locales de l'état de l'art proposent des stratégies de perturbations fortes pour se déplacer dans l'espace recherche : un coefficient adaptatif et une perturbation à base de recherche locale bloquant les sommets jusqu'à la fin de la perturbation *AFISA*, le déplacement des sommets d'une couleur à une autre et la mise en place de poids sur les arêtes *RedLS*, ou le fait de retirer la couleur aux sommets les plus lourds d'une à trois couleurs *ILS-TS*. Ces stratégies mettent en avant une réelle difficulté à explorer

efficacement l'espace de recherche du WVCP, même avec une liste tabou. Le point de départ de ces algorithmes peut aussi influencer leur direction. C'est pourquoi utiliser un algorithme tel que l'arbre de recherche de Monte Carlo pourrait permettre d'explorer continuellement l'espace de recherche depuis de nouveaux points de départ, et apprendre quelles sont les zones de l'espace de recherche les plus prometteuses pour une recherche locale.

L'organisation du chapitre est la suivante. Nous proposons d'abord une nouvelle recherche locale, *TabuWeight* et comparons les différentes recherches locales entres elles. Puis nous proposons une intégration entre le MCTS et des recherches locales, le MCTS peut alors vu dans ce cas comme un outil pour découvrir de nouveaux points de départ pour la recherche locale. Enfin, nous étudions l'impact des différents opérateurs de recherche locale combinés à l'algorithme MCTS présenté dans le chapitre précédent.

6.2 Recherches Locales pour le WVCP

Les recherches locales de l'état de l'art, *AFISA* [Sun *et al.*, 2018] (espace illégal), *RedLS* [Wang *et al.*, 2020] (espace illégal) et *ILS-TS* [Nogueira *et al.*, 2021] (espace partiellement légal) sont trois recherches locales relativement complexes que l'on peut toutes classer dans la famille des recherches locales itérées (cf. Chapitre 2). Une recherche locale est exécutée sur une solution partielle ou illégale jusqu'à atteindre une solution légale ou un nombre d'itérations limites. Ensuite, une perturbation est appliquée pour se déplacer dans l'espace de recherche, par exemple en forçant un déplacement des sommets les plus lourds vers une autre couleur comme dans l'algorithme *ILS-TS*. *AFISA* et *RedLS* implémentent aussi des stratégies qui leur permettent de naviguer entre différentes zones de solutions légales en acceptant temporairement de passer par des solutions illégales. *AFISA* minimise explicitement un compromis entre la somme du score du WVCP et le nombre de conflits de la solution courante multipliés par un coefficient adaptatif qui augmente lorsque la recherche locale ne parvient pas à revenir vers des solutions légales pour donner plus d'importance aux sommets en conflit qu'aux sommets les plus lourds de chaque couleur. *RedLS* donne des poids aux arêtes du graphe, lorsque des sommets sont en conflit, le poids est utilisé comme pénalité, si le conflit n'est pas résolu et que la recherche est bloquée, le poids de l'arête augmente et sera prioritaire pour être résolu ensuite et il sera plus difficile de placer ces deux sommets dans la même couleur ensuite, car plus difficile à séparer.

En complément de ces recherches locales de l'état de l'art, nous avons aussi proposé ci-dessous une recherche locale, nommée *TabuWeight*, qui correspond à une adaptation de l'algorithme *TabuCol* du GCP pour le WVCP. Cet algorithme, relativement simple, nous servira

notamment de *baseline* pour effectuer des comparaisons avec les autres opérateurs.

TabuWeight (TW) [Grelier *et al.*, 2022] est une recherche locale avec tabou inspiré de *TabuCol* pour le GCP [Hertz et Werra, 1987]. À partir d'une solution légale, *TabuWeight* l'améliore de manière itérative en utilisant un opérateur *one move*, qui consiste à déplacer un sommet de son groupe de couleurs vers un autre groupe de couleurs, sans créer de conflits. À chaque itération, le meilleur *one move* qui n'est pas interdit par la liste tabou est sélectionné. Chaque fois qu'un mouvement est effectué, le mouvement inverse est ajouté à la liste tabou et interdit pour les tt itérations suivantes où tt est un paramètre appelé *tabu tenure*. Un mouvement tabou peut encore être appliqué exceptionnellement s'il conduit à une solution qui est meilleure que la meilleure solution trouvée jusqu'à présent (critère d'aspiration). Le lecteur averti remarquera que l'algorithme ne présente pas de perturbations ou de stratégies particulières, hors liste tabou, pour quitter des minima locaux. Effectivement, cette recherche locale est plutôt orientée pour travailler avec un algorithme apportant de la diversité tel que le MCTS ou un algorithme mémétique. Pour une utilisation en tant que recherche locale seule, perturber la solution en déplaçant les sommets les plus lourds d'une couleur à une autre en cas de non-amélioration pourrait améliorer ses résultats. Nous verrons cependant dans les expérimentations que *TabuWeight* peut être plus efficace que des algorithmes de recherche locale plus sophistiqués comme *AFISA* ou *RedLS* sur des instances difficiles des benchmarks, lorsqu'il est combiné au MCTS.

Dans la section suivante, nous proposons une intégration des recherches locales avec l'algorithme MCTS présenté dans le chapitre précédent.

6.3 Combinaison du MCTS avec la Recherche Locale

Nous explorons ici la possibilité d'améliorer l'algorithme du MCTS avec une recherche locale. Le couplage du MCTS avec un algorithme de recherche locale est motivé par le fait qu'après la phase de simulation, la solution complète obtenue peut être proche d'une solution encore meilleure dans l'espace de recherche qui pourrait être découverte par la recherche locale. De plus, du point de vue de la recherche locale, repartir de nouveaux points de départ peut permettre de mieux diversifier la recherche.

Le cadre d'étude détaillé dans cette section est proposé pour le WVCP. Il pourrait aussi être utilisé pour le GCP. Cependant, nos tests n'ont pas montré d'apports d'une combinaison avec des recherches locales dans ce cas du GCP, donc ces résultats ne seront pas présentés. Cet aspect sera discuté en conclusion.

Dans ce chapitre, nous présentons le couplage du MCTS pour le WVCP avec quatre recherches locales : la nouvelle procédure *TabuWeight* présentée ci-dessus, ainsi que les trois meilleurs algorithmes de recherche locale de l'état de l'art, dédiés au WVCP, *AFISA* [Sun *et al.*, 2018], *RedLS* [Wang *et al.*, 2020] et *ILS-TS* [Nogueira *et al.*, 2021].

Le fonctionnement de l'algorithme MCTS est exactement le même que celui présenté dans le chapitre précédent, à l'exclusion de la phase de simulation qui met en jeu maintenant une phase de recherche locale. L'Algorithme 8 reprend l'algorithme du MCTS présenté dans le chapitre précédent et met en évidence les changements en gras.

Pendant la phase de simulation, la solution est d'abord complétée avec un algorithme glouton, puis améliorée par la procédure de recherche locale. Notez que dans la première version de ce travail publiée dans [Grelier *et al.*, 2022], pour rester cohérents avec l'arbre de recherche appris par MCTS, nous avons permis à la procédure de recherche locale de déplacer uniquement les sommets de la solution complète S qui sont encore non colorés après les phases de sélection et d'expansion. Cependant, nous avons réalisé entre-temps que le blocage des sommets pour la recherche locale peut conduire à beaucoup de temps passé à vérifier les sommets bloqués dans le voisinage complexe exploré par les différentes procédures de recherche locale. Cela conduit également à manquer des opportunités de se déplacer dans l'espace de recherche et à une dégradation des résultats. Par conséquent, une version plus efficace de l'algorithme est présentée ici où les sommets ne sont pas gelés pendant la recherche locale. Dans cette nouvelle version, lors du couplage du MCTS avec un algorithme de recherche locale, l'heuristique résultante peut être considérée comme un algorithme qui tente d'apprendre un bon point de départ pour la procédure de recherche locale, en sélectionnant différents meilleurs squelettes prometteurs de solutions partielles à chaque itération pendant la phase de sélection.

Comme une itération n'a pas la même signification pour chaque recherche locale, nous utilisons une limite de temps commune à toutes pendant la simulation. Une étude de ce temps sera proposée dans la partie qui détaille les expérimentations. Une fois que la procédure de recherche locale a atteint la limite de temps, le score correspondant à la meilleure solution légale obtenue par la procédure de recherche locale est utilisé pour mettre à jour tous les nœuds de la branche qui a conduit à l'initiation de la simulation.

Algorithme 8 MCTS avec recherche locale

Entrée: Graphe pondéré $G = (V, E, w)$, o **un opérateur de recherche locale**.
Sortie: Meilleure solution S^* trouvée

$S^* = \emptyset$ et $f(S^*) = MaxInt$
$D = \emptyset$.

tant que tant que la condition d'arrêt n'est pas satisfaite **faire**
    $C \leftarrow R$ ▷ noeud courant correspondant au noeud racine de l'arbre
    $S \leftarrow \{V_1, U\}$ avec $V_1 = \{v_1\}$ et $U = V \backslash V_1$ ▷ premier sommet dans le premier groupe de couleurs

    /* Étape 1 - Sélection */
    tant que C n'est pas une feuille **faire**
        $C \leftarrow$ select_best_child(C) avec une coloration légale $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$

    /* Étape 2 - Expansion */
    si C a un enfant potentiel, pas encore ouvert **alors**
        $C \leftarrow$ open_first_child_not_open(C) avec une coloration légale $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$

    /* Étape 3 - stratégie de simulation avec Recherche Locale */
    $S \leftarrow$ glouton(S) ▷ **compléter la solution courante avec un algorithme glouton**
    $S' \leftarrow o(S)$ ▷ **améliorer la solution avec l'opérateur de recherche locale**

    /* Étape 4 - Mise à jour */
    tant que $C \neq R$ **faire**
        update(C,f(S'))
        $C \leftarrow$ parent(C)
    si $f(S') < f(S^*)$ **alors**
        $S^* \leftarrow S'$

    /* Étape 5 - Élagage */
    application des règles d'élagage

retourne S^*

6.4 Expérimentations

Cette section présente les expériences menées pour évaluer les différentes stratégies de simulation et les recherches locales.

6.4.1 Paramètres Expérimentaux

Nous considérons les 188 instances de benchmark du WVCP réduites (cf. Chapitre 3). Toutes les instances originales et réduites sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_instances`.

Parmi ces instances, 142 ont un score optimal connu. L'objectif de ces variantes MCTS couplées à une procédure de recherche locale n'est pas de prouver l'optimalité. Par conséquent, lors de l'évaluation d'une méthode donnée sur une instance où le score optimal est connu, celle-ci s'arrête lorsque ce score est atteint. La découpe des instances en quatre ensembles est la même que dans le chapitre précédent. Nous effectuons 20 exécutions indépendantes de chaque méthode pour résoudre chaque instance sur un ordinateur équipé d'un processeur Intel Xeon ES 2630, 2,66 GHz. La limite de temps pour chaque algorithme est d'une heure.

Les algorithmes sont codés en C++, compilés et optimisés avec un compilateur g++ 12.1. Le code source et les résultats complets des algorithmes sur toutes les instances sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts`.

Des changements dans les résultats peuvent être observés par rapport à la publication [Grelier *et al.*, 2022] dus à quelques corrections et optimisations dans le code, l'utilisation du flag LTO à la compilation et d'une réduction des instances plus complète. Les meilleurs scores ou scores optimaux sont aussi mis à jour au niveau de l'état de l'art au moment de l'écriture de la thèse et non au moment de la publication de l'article.

Dans ce qui suit, lorsqu'une méthode est dite meilleure qu'une autre sur une instance donnée, cela signifie que la différence entre les scores moyens calculés sur 20 exécutions est en faveur de la première méthode et que cette différence est significative (un test de rang signé de Wilcoxon non paramétrique avec une valeur $p \leq 0,001$).

6.4.2 Temps de Recherche Locale

Le temps passé dans la recherche locale lors de la simulation a un impact important sur la recherche. Non seulement il règle le temps passé à intensifier sur une branche de l'arbre, mais surtout il consomme du temps qui pourrait être utilisé pour explorer plus en profondeur

l'arbre. Nous avons donc étudié les impacts sur les quatre différentes recherches locales avec un temps dépendant du nombre de sommets dans le graphe, $t = r * |V|$ secondes où $r \in \{0.01, 0.02, 0.04, 0.08, 0.1, 0.2\}$.

Le comportement du MCTS observé lorsque le temps passé dans la recherche locale augmente est différent en fonction des recherches locales. Par exemple, pour *AFISA* et *ILS-TS*, il est préférable de passer plus de temps pour obtenir en moyenne de meilleurs scores. A contrario, pour *TabuWeight* et *RedLS*, les temps les plus courts sont préférables pour obtenir de meilleurs scores en moyenne. Pour le temps passé pour atteindre un même score, il vaut mieux un temps de recherche locale court dans la majorité des cas. L'impact sur le nombre de fois où le nombre de meilleurs scores connus est atteint n'a rien de significatif entre les différents temps. Afin d'explorer au maximum l'arbre, de trouver de bons scores rapidement et de maximiser les chances de trouver des bons scores, nous avons décidé d'utiliser $t = 0.02 * |V|$ secondes de temps de recherche locale dans le MCTS, avec $|V|$ le nombre de sommets du graphe. La section suivante étudie l'impact du MCTS en combinaison avec la recherche locale.

6.4.3 Arbre de Recherche de Monte Carlo avec Recherche Locale

Cette section étudie la combinaison du MCTS avec une heuristique de recherche locale.

Le Tableau 6.1 résume les résultats des procédures de recherche locale présentées dans la Section 6.2 suivies de la combinaison du MCTS avec chacune de ces procédures pendant la phase de simulation (cf. Section 6.3). Chaque ligne présente le nombre de fois où un meilleur score connu dans l'état de l'art est atteint avec la méthode.

Nous observons dans ce tableau que la procédure de recherche locale permettant d'atteindre le plus grand nombre de meilleurs scores connus est *ILS-TS*, suivi de *AFISA* puis *RedLS* et enfin *TabuWeight*. Pour les combinaisons du MCTS avec les procédures de recherche locale, MCTS+*ILS-TS* est premier, suivi de MCTS+*RedLS* puis MCTS+*TabuWeight* et enfin MCTS+*AFISA*.

instances	|I|	*AFISA*	MCTS+*AFISA*	TW	MCTS+TW	*RedLS*	MCTS+*RedLS*	*ILS-TS*	MCTS+*ILS-TS*
pxx	35*	**35**	34	29	**35**	31	**35**	**35**	**35**
rxx	30*	5	3	2	16	3	25	**30**	**30**
DIMACS_easy	75(72*)	71	73	61	72	70	**75**	**75**	**75**
DIMACS_hard	48(5*)	3	5	7	9	8	17	**19**	15
Total	188	114	115	99	132	112	152	**159**	155

TABLE 6.1 – Résumé du nombre de fois où le meilleur score connu est atteint pour les procédures de recherche locale et le MCTS combiné aux différentes procédures de recherche locale. Les valeurs en gras mettent en évidence les meilleurs résultats pour chaque ligne.

Les instances pxx sont presque toutes résolues par toutes les méthodes, contrairement aux instances rxx. On remarque un apport du MCTS pour *TabuWeight* et *RedLS* et un désavantage pour *AFISA* sur ces instances. Le MCTS apporte aussi du soutien pour les instances DIMACS_easy pour toutes les méthodes. Pour les instances DIMACS_hard, seul *ILS-TS* est désavantagé par la combinaison avec le MCTS. De manière générale, *TabuWeight*, *AFISA* et *RedLS* améliorent leurs résultats lorsqu'ils sont couplés au MCTS. En particulier, la variante MCTS+*RedLS* trouve le meilleur score connu pour 17 instances difficiles de l'ensemble DIMACS_hard. De plus, au moment de la publication, cette combinaison a permis de trouver deux nouveaux scores pour les instances difficiles DIMACS le450_15a et queen12_12gb qui n'avaient jamais été reportés dans la littérature. Cela montre que le *framework* MCTS proposé peut aider un algorithme de recherche locale tel que *RedLS* à trouver continuellement de nouveaux points de départ prometteurs dans l'espace de recherche.

Cependant, lorsqu'il est couplé à l'algorithme *ILS-TS* (variante MCTS+*ILS-TS*), il ne semble pas améliorer les résultats. Cela peut s'expliquer par le fait que *ILS-TS* est un algorithme de recherche locale itérée intégrant déjà divers mécanismes de perturbation permettant d'échapper aux optima locaux. Par conséquent, trouver de nouveaux bons points de départ dans l'espace de recherche guidée par MCTS semble moins intéressant pour *ILS-TS* que pour les autres procédures de recherche locale.

Le Tableau 6.2 montre les comparaisons par paires entre toutes les méthodes étudiées. Nous pouvons remarquer que, malgré un bon nombre de meilleurs scores connus atteints, *RedLS* est rarement significativement meilleur par rapport aux autres méthodes, cela est dû au fait que l'algorithme est capable d'atteindre de très bons scores, mais qu'il reste souvent bloqué dans des minima locaux.

Lorsque l'on compare les performances des algorithmes de recherche locale en regardant le nombre de fois où ils sont significativement meilleurs que les autres en termes de score moyen, le classement diffère du tableau précédent. *ILS-TS* domine toujours, suivi d'*AFISA* puis de *TabuWeight* et enfin de *RedLS*. Comme pour les variantes avec le MCTS, MCTS+*RedLS* domine, suivi de MCTS+*ILS-TS*, puis de MCTS+*TabuWeight* et enfin de MCTS+*AFISA*.

Lorsque l'on compare MCTS+Greedy-Random et MCTS+*DSatur* du chapitre précédant aux recherches locales seules ou bien aux variantes hybrides du MCTS couplé aux recherches locales, les deux se défendent bien contre *AFISA*, MCTS+*AFISA*, *TabuWeight* et *RedLS* mais pas contre les variantes du MCTS combiné aux procédures de recherche locale *TabuWeight*, *RedLS* et *ILS-TS*.

Les variantes du MCTS couplé à une recherche locale améliorent les résultats et sont plus

	MCTS+GR	MCTS+*DSatur*	*AFISA*	MCTS+*AFISA*	*TabuWeight*	MCTS+TW	*RedLS*	MCTS+*RedLS*	*ILS-TS*	MCTS+*ILS-TS*
MCTS+GR	-	32	**77**	**63**	**77**	33	**85**	0	0	1
MCTS+*DSatur*	**48**	-	**93**	**72**	**89**	42	**107**	12	0	2
AFISA	33	50	-	35	**49**	17	**53**	0	0	0
MCTS+*AFISA*	40	39	**40**	-	**73**	10	**86**	0	1	0
TabuWeight	56	54	40	40	-	25	**48**	1	0	3
MCTS+TW	**52**	**64**	**74**	**73**	**78**	-	**101**	5	1	0
RedLS	51	57	41	35	47	29	-	11	14	15
MCTS+*RedLS*	**94**	**92**	**103**	**82**	**107**	**72**	**102**	-	**20**	**28**
ILS-TS	**101**	**95**	**104**	**82**	**106**	**75**	**96**	19	-	**18**
MCTS+*ILS-TS*	**101**	**93**	**102**	**82**	**103**	**73**	**92**	15	2	-

TABLE 6.2 – Comparaison entre toutes les meilleures variantes du MCTS+Glouton et des MCTS+LS avec les procédures de recherche locale. Chaque ligne correspond au nombre de fois où la méthode de la ligne est significativement meilleure que la méthode de la colonne sur les 188 instances. Par exemple, sur 188 instances, *TabuWeight* (TW) est meilleur que MCTS+*TabuWeight* sur 25 instances, 48 fois contre *RedLS* et 1 fois contre MCTS+*RedLS*. Les valeurs en gras signifient que la méthode a est plus souvent meilleure que la méthode b lorsque l'on regarde combien de fois b est significativement meilleure que a.

souvent significativement meilleures que la recherche locale correspondante seule, sauf pour *ILS-TS*, qui est meilleur par lui-même et domine la variante MCTS+*ILS-TS*.

La variante MCTS+*RedLS* est significativement meilleure que les autres méthodes, bien que très proche de *ILS-TS*. *ILS-TS* garde l'avantage sur quelques instances rxx, wap, quelques queen de petite taille, et les C2000 où MCTS+*RedLS* est meilleur sur certains DSJC, sur les flat1000, latin_square, le450 et les queen de grande taille.

Ces résultats indiquent que combiner MCTS avec une recherche locale est intéressant pour améliorer les procédures de recherche locale sous-jacentes telles que *AFISA*, *TabuWeight* et *RedLS*.

Nous avons aussi constaté que certaines instances sont mieux résolues par certaines recherches locales. Par exemple, *ILS-TS* est excellent pour résoudre les instances de petite et moyenne taille (pxx, rxx, DIMACS_easy), mais est dépassé par *RedLS* sur des instances de grande taille comme les C2000, DSJC, latin_square, flat1000. La combinaison du MCTS à ces recherches locales permet de gommer l'avantage de certaines petites instances tout en restant avantageux sur de grandes instances. De plus, le fait de reprendre la recherche de différents points est très intéressant pour *RedLS* qui est très bon pour trouver rapidement un bon score,

mais en moyenne reste bloqué dans des optima locaux ce qui montre sa difficulté à naviguer dans l'espace de recherche.

Le Tableau 6.3 présente les résultats des recherches locales et combinaisons avec le MCTS. Les valeurs en gras indiquent les meilleurs résultats parmi les méthodes, pas systématiquement égaux au meilleur score connu. Nous pouvons constater avec le pied du tableau qu'à l'exception d'*ILS-TS*, le MCTS permet d'obtenir de meilleurs résultats et une meilleure moyenne sur le double d'instances pour *AFISA* et *TabuWeight* et le triple pour *RedLS*. Pour *RedLS*, nous obtenons une détérioration des résultats sur les instances C2000 ou DSJC1000 avec le MCTS, mais celui-ci lui permet de résoudre les instances rxx ou queen avec plus de facilité.

instance	BKS	*AFISA*			MCTS+*AFISA*			*TabuWeight*			MCTS+*TabuWeight*			*RedLS*			MCTS+*RedLS*			*ILS-TS*			MCTS+*ILS-TS*		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	2384	2403.4	3601	2635	2648.9	2665	2318	2332.2	3477	2410	2423.6	1578	**2167**	**2193.8**	2403	2354	2369	2091	2237	2266.4	3498	2318	2340.2	697
C2000.9	5477	6582	6650.1	0	6547	6570.2	1107	6049	6073.4	2819	6242	6293.6	1845	**5502**	**5528.1**	3303	6060	6093.6	1107	5910	5969.9	3587	6103	6119.9	430
DSJC125.1gb	90*	**90**	91.1	917	**90**		712	96	99.3	1	**90**	91.5	2727	91	93.9	0	**90**		385	**90**		18	**90**		9
DSJC125.1g	23*	**23**	24.4	13	**23**	23.5	1203	24	24.6	68	**23**	23.1	1684	**23**	23.4	0	**23**		6	**23**		2	**23**		5
DSJC125.5gb	240	241	245.6	103	242	243.7	2562	**240**	240.9	757	**240**	240.2	1937	245	260.5	0	241	241.2	870	**240**		106	**240**		94
DSJC125.5g	71	**71**	72.2	1360	72		529	**71**	71.5	1661	**71**	71.8	1686	72		257	**71**		378	**71**		126	**71**		40
DSJC125.9gb	604*	**604**	610.5	4	**604**		778	**604**	609.7	2527	**604**	605.8	1901	**604**		1	**604**		1	**604**		167	**604**		66
DSJC125.9g	169*	170	174.1	2	**169**	169.1	1047	**169**	170.7	747	**169**	169.8	959	**169**		0	**169**		0	**169**		182	**169**		171
DSJC250.1	127	129	133.6	3294	133	133.8	1764	134	136.8	25	132	133.5	780	130	131.6	1	**127**	**127.5**	1785	**127**	127.8	1608	**127**	128	642
DSJC250.5	392	411	424.2	541	421	428.8	2586	397	406	2717	406	408.6	2349	398	401.2	103	396	397.9	3045	**393**	**397.6**	2567	397	399.8	1695
DSJC250.9	934*	949	976.1	232	962	978.8	1380	959	963.2	1766	950	956	2292	**934**	**935.6**	718	935	935.9	2406	936	942.1	3053	948	953.4	2604
DSJC500.1	184	198	201.5	1817	220	228.6	165	194	197	1526	201	202.1	1312	**187**	201.9	537	**187**	**187.8**	1764	188	188.8	1744	**187**	188.6	3168
DSJC500.5	685	762	778.1	1307	848	863.5	1474	733	741.3	3571	751	756	2233	**706**	**716.1**	2840	716	719.2	3234	724	735.5	1744	729	734.7	3410
DSJC500.9	1662	1744	1775.5	652	1909	1919.3	2002	1733	1755.1	228	1764	1775	1705	**1670**	**1675.1**	945	1687	1694.7	726	1720	1742	3039	1762	1775.7	77
DSJC1000.1	300	319	325	2182	396	401.9	357	313	316.2	2943	365	368.8	2604	305	307.2	1235	307	307.9	1799	305	307.4	1574	**303**	**304.9**	1176
DSJC1000.5	1185	1308	1330	3531	1463	1475	2226	1271	1282.3	3460	1304	1311.1	2562	**1198**	**1213.7**	2381	1244	1248.5	1197	1245	1269.2	231	1257	1274.2	1890
DSJC1000.9	2836	3066	3107	3601	3303	3325.1	2793	3034	3061.1	1381	3135	3152.9	546	**2840**	**2858.5**	2953	2974	2992.2	651	3026	3066.8	3580	3098	3123.7	3003
DSJR500.1	169*	**169**		54	**169**		157	171	178.2	21	**169**	169.2	1273	171	184.5	0	**169**		318	**169**		0	**169**		5
flat1000_50_0	924	1267	1293.3	2736	1421	1431.3	1512	1238	1247.5	3230	1271	1274.9	2194	**1155**	**1173.8**	3238	1199	1206.2	336	1222	1235	1712	1221	1231.3	42
flat1000_60_0	1162	1309	1323.2	3584	1465	1476.8	1575	1275	1288	2275	1305	1313.5	21	**1191**	**1205.7**	1080	1238	1244.5	1491	1250	1270	125	1259	1266.2	3360
flat1000_76_0	1165	1288	1304.4	3278	1442	1449.8	2341	1237	1263	2975	1285	1292.8	693	**1176**	**1194**	1107	1214	1222	1176	1232	1247.5	1198	1237	1246.5	1008
GEOM120a	105*	**105**	106.6	136	**105**	105.1	515	**105**	106.1	453	**105**		351	**105**	109.2	9	**105**		9	**105**		0	**105**		0
GEOM120b	35*	**35**	36.2	422	**35**	36	2557	**35**	35.3	1156	**35**	35.3	2051	**35**	35.5	0	**35**		0	**35**		0	**35**		0
GEOM120	72*	**72**	73	0	**72**		8	74	76.2	0	**72**	73	2097	**72**	75.2	0	**72**		13	**72**		0	**72**		0
latin_square_10	1480	1607	1652.4	2257	2037	2055.9	2394	1816	1855.2	45	1774	1789.8	247	**1505**	**1523**	2369	1533	1544.8	1938	1559	1581.2	1368	1572	1585.8	3458
le450_25a	306	311	316.3	2699	316	318.9	283	321	325.4	83	313	315.9	1456	**306**	307.4	503	**306**		131	**306**		174	**306**		736
le450_25b	307*	308	312.6	1891	308	310.1	2114	308	312.9	219	**307**	308.5	1491	**307**	313.4	56	**307**		86	**307**		10	**307**		18
le450_25c	342	355	364.4	3436	386	398.2	3375	361	366	677	371	373.6	1908	351	354.8	43	**348**	**349.4**	382	**348**	351.8	3207	351	353.1	1689
le450_25d	330	351	357.8	1630	383	391.8	1494	351	356.4	1955	363	366.3	3415	**332**	338.9	154	334	**335.9**	1435	339	342.5	1999	342	343.8	2283
queen10_10	162	165	166.9	524	**162**	164.1	2994	163	163.9	1629	**162**	162.3	1249	**162**	166.2	504	**162**		37	**162**		13	**162**		10
queen10_10gb	164	166	168.6	756	**164**	167	2046	166	167.4	1582	165	165.4	1846	165	166.2	284	**164**	164.8	1042	**164**		139	**164**		196
queen10_10g	43*	**43**	43.8	1405	**43**		1205	**43**		140	**43**		137	**43**	43.2	18	**43**		8	**43**		3	**43**		12
queen11_11	172	179	181.1	722	176	178.8	2859	**172**	172.9	1731	**172**	173.2	1071	174	177.1	761	**172**	173.2	1935	**172**	**172.6**	1820	**172**	172.8	1703
queen11_11gb	176	178	182.4	528	178	180.1	1998	178	178.8	556	177	177.8	1661	177	178.9	12	**176**		755	**176**		337	**176**		303
queen11_11g	47	48	49.1	292	48	48.8	1965	**47**	47.7	1570	**47**	47.9	2803	48	48.1	3	**47**		192	**47**		97	**47**		100
queen12_12	185	193	196.7	2062	191	193.9	3216	187	188.1	1881	187	188.4	2603	188	190.2	7	**185**	186.6	432	**185**	**186.1**	1261	186	186.4	1572
queen12_12gb	191	199	202.2	764	196	198.3	1663	198	200.5	1171	195	195.9	2362	193	195.3	1596	**191**	**191.3**	1442	**191**	**191.3**	1421	**191**	191.6	1502
queen12_12g	50	52	53.8	31	52	52.8	1823	51		271	51		886	**50**	51.9	1063	**50**		994	**50**		773	**50**	50.1	1005
queen13_13	194	203	206.7	288	202	205.2	1684	198	201.2	2205	199	200.5	2765	195	198.7	1528	**194**	**194.6**	1604	**194**	195.7	3475	195	196.6	2480
queen14_14	215	224	230.5	3113	226	229.2	184	217	218.4	1678	219	220.1	2014	217	222.4	18	**216**	**216.9**	918	**216**	217.3	2018	217	218.4	1514
queen15_15	223	236	240.2	1572	241	242.9	1745	233	236.8	2417	233	235.5	1835	227	229.7	1109	**225**	**226.2**	717	227	228.4	2167	227	229.2	1925
queen16_16	234	248	255.2	1055	255	258.2	3012	239	242.3	2745	243	244.8	1002	**237**	239.9	91	**237**	**237.7**	1812	238	240.2	2938	240	241.2	1204
p30	4891*	**4891**		0	**4891**		0	**4891**		0	**4891**		0	**4891**	4909.2	0	**4891**		3	**4891**		0	**4891**		0
p31	620*	**620**		0	**620**		0	**620**		0	**620**		0	**620**		0	**620**		0	**620**		0	**620**		0
p32	2480*	**2480**		0	**2480**		0	**2480**		0	**2480**		0	**2480**	2480.5	0	**2480**		0	**2480**		0	**2480**		0
r28	9407*	9415	9511.4	1819	9462	9508.6	3120	9460	9583.2	0	9411	9420.4	78	9410	9563	45	**9407**	9412	1573	**9407**		8	**9407**		5
r29	8693*	8715	8799	379	8816	8957.9	1554	8768	8894.9	0	8707	8743.4	2454	8696	8817.6	0	**8693**		539	**8693**		1	**8693**		1
r30	9816*	9826	9960	2378	9820	9840	504	9819	9871.4	0	**9816**	9816.8	1517	9836	9988.2	1	**9816**	9824	2570	**9816**		5	**9816**		7
#BKS			114			115			99			132			112			152			159			155	
#Best			114			115			99			132			129			159			165			157	
#Best Avg			45			97			53			98			44			145			160			152	

TABLE 6.3 – Résultats des MCTS+LS face aux recherches locales de l'état de l'art sur un ensemble d'instances variées pour le WVCP. En pied de tableau se trouvent les statistiques de nombre de BKS (*Best Known Score* - meilleurs scores connus dans l'état de l'art), nombre de fois où la méthode est la meilleure (#Best) ou obtient la meilleure moyenne (# Best Avg) sur l'ensemble des 188 instances de l'état de l'art.

6.5 Conclusion

Dans ce chapitre, un framework de type Monte Carlo Tree Search combiné avec une recherche locale a été présenté et testé sur le problème de coloration de graphe pondéré. Une nouvelle recherche locale, *TabuWeight*, a été introduite.

L'amélioration des résultats avec *AFISA*, *TabuWeight* et *RedLS* montre une réelle utilité à combiner l'approche du MCTS avec des recherches locales dans le cadre du WVCP.

Nous avons aussi testé l'algorithme sur le GCP avec les recherches locales *TabuCol* et *PartialCol* mais n'avons pas constaté d'intérêt de la combinaison avec le MCTS, voire même une dégradation des résultats. Nous pensons que ce phénomène est dû à la différence dans les espaces de recherches des deux problèmes (cf. Chapitre 1). En effet, nous avions vu que le GCP demande rapidement beaucoup d'intensification et qu'il est souvent plus aisé de naviguer dans son espace de recherche même lorsque le paysage de *fitness* est plus rugueux. À l'inverse, l'espace de recherche du WVCP étant composé de beaucoup de plateaux (de par le fait que seuls les sommets les plus lourds de chaque couleur ont un impact sur le score), la capacité de pouvoir complètement changer de zone de l'espace de recherche, apportée par le MCTS, devient alors intéressante.

Comme perspectives d'amélioration, nous pourrions envisager de ne pas lancer systématiquement la recherche locale à chaque itération de l'algorithme et mais plutôt nous fier à l'état de la solution après le glouton pour décider si oui ou non la recherche locale se révèlera utile. Nous avons exploré le fait d'utiliser des indicateurs sur la solution pour faire ce choix, comme le score de la solution, la distance entre la solution courante et les solutions obtenues précédemment après recherche locale, mais les premiers tests n'ont pas permis d'obtenir des résultats encourageants.

Par contre, utiliser un réseau de neurones préentraîné ou entraîné pendant la recherche pourrait permettre de prédire les résultats obtenus par les différentes recherches locales, avant éventuellement de les réaliser de manière effective, à la manière de celui utilisé dans [Goudet *et al.*, 2022] et similaire à celui présenté dans le chapitre suivant.

CHAPITRE 7

MCTS ET HYPERHEURISTIQUES

Ce chapitre présente la combinaison de l'algorithme MCTS avec des hyperheuristiques pour sélectionner des opérateurs de recherche locale pendant la recherche. Nous proposons un nouveau critère de sélection basé sur un réseau de neurones et comparons les performances de ce critère avec d'autres critères classiques basés sur le score des solutions. Les impacts de différents opérateurs de recherche locale et de différentes politiques de sélection, y compris le biais proportionnel, le bandit à un bras et un réseau de neurones, sont étudiés. Des expériences sur des instances de référence du problème de coloration pondérée sont menées pour mettre en évidence les avantages et les limites de chaque stratégie de sélection dynamique.

Ce chapitre est tiré de l'article publié à EVOCOP 2023 [Grelier *et al.*, 2023].

7.1 Introduction

Dans le chapitre précédent, nous avons étudié des combinaisons de différentes heuristiques de recherche locale pour le WVCP utilisées lors de la phase de simulation d'un MCTS. Nous avons montré que l'apprentissage de l'arbre permet alors de trouver continuellement de nouveaux bons points de départ pour les recherches locales. Cet avantage permet d'explorer plus efficacement l'espace de recherche constitué de beaucoup de plateaux dans le cas du WVCP. Cependant, nous avons observé que les résultats étaient très dépendants de l'algorithme de recherche locale utilisé pour chaque classe d'instances. Certaines procédures comme *RedLS* sont efficaces plutôt pour de grandes instances, tandis que *ILS-TS* est plus efficace pour les instances de plus petite taille. De façon à proposer un algorithme plus robuste et meilleur pour l'ensemble des instances des benchmarks considérés, nous proposons d'étudier dans ce chapitre l'apport d'une sélection dynamique et *online* de la procédure de recherche locale à appliquer lors de la phase de simulation du MCTS, au cours de la résolution d'une instance donnée.

Des hyperheuristiques ont été proposées dans la littérature à cette fin. Les hyperheuristiques sont des méthodes de recherche qui explorent l'espace composé par les heuristiques de bas niveau, au lieu de l'espace des solutions directement [Burke *et al.*, 2013]. Nous renvoyons

le lecteur à [Burke *et al.*, 2013, Drake *et al.*, 2020] pour un aperçu des hyperheuristiques existantes proposées dans la littérature pour résoudre divers problèmes d'optimisation combinatoire. Plus précisément, les hyperheuristiques ont déjà été utilisées pour résoudre des problèmes de partitionnement, avec des applications pour la planification et à la coloration de graphe [Sabar *et al.*, 2012, Elhag et Özcan, 2015], mais jamais pour le WVCP à notre connaissance.

La plupart des hyperheuristiques utilisées pour les problèmes de partitionnement appliquent une heuristique de bas niveau choisie pour la solution courante à chaque étape de la recherche avant de décider d'accepter ou de rejeter la nouvelle solution créée [Elhag et Özcan, 2015]. Cependant, ces hyperheuristiques itératives sont susceptibles de rester bloquées dans des optima locaux, en particulier lorsqu'elles s'attaquent à des problèmes de coloration de graphe qui sont généralement caractérisés par un paysage de fitness rugueux [Bouziri *et al.*, 2011]. Cela se produit lorsque l'hyperheuristique n'incorpore pas de composants de bas niveau efficaces qui peuvent être choisis pour diversifier la recherche dans une autre zone de l'espace de recherche. De plus, la plupart des hyperheuristiques proposées dans la littérature utilisent une stratégie de haut niveau prenant en compte le gain de fitness obtenu par les différents composants de bas niveau (par exemple [Goëffon *et al.*, 2016]) mais ne tiennent pas compte de l'état courant de la solution améliorée de manière itérative.

Pour surmonter ces deux limitations, nous proposons une approche hyperheuristique avec un apprentissage *online*, qui combine le framework de recherche d'arbre de Monte Carlo (MCTS) proposé dans le chapitre précédent avec un sélecteur de recherches locales qui tient compte de l'état brut de la solution actuelle à améliorer pendant la phase de simulation du MCTS. Ce framework MCTS peut apprendre en continu de nouveaux points de départ prometteurs pour une heuristique de recherche locale, ce qui pourrait aider une hyperheuristique itérative à éviter les pièges des minima locaux.

Deux travaux précédents [Sabar et Kendall, 2015, Asta *et al.*, 2016] ont également utilisé un MCTS dans une hyperheuristique pour les problèmes d'optimisation combinatoire. Dans [Sabar et Kendall, 2015], l'espace de recherche des heuristiques de bas niveau est exploré par un MCTS, qui recherche la meilleure séquence d'heuristiques de bas niveau à appliquer à une solution donnée, mais sans garanties d'optimalité. Contrairement à cette approche, le MCTS utilisé dans notre travail est directement construit pour explorer l'arbre des solutions légales. Par conséquent, il a la propriété de pouvoir fournir une garantie théorique qu'une solution optimale peut être trouvée si suffisamment de temps est donné à l'algorithme. Dans [Asta *et al.*, 2016], un MCTS a été utilisé pour créer des séquences initiales, qui sont améliorées dans une deuxième étape par une hyperheuristique utilisant une grande variété de mouvements locaux, sélectionnés

par une stratégie de haut niveau. Dans notre travail, nous visons à pousser plus loin le couplage entre la sélection par le MCTS et le choix de l'heuristique de bas niveau.

Nous résumons les contributions du chapitre comme suit. Nous proposons un nouveau *framework* de sélection pour choisir à la volée un opérateur parmi un ensemble de recherches locales en fonction de la solution courante à améliorer. Le score obtenu par une recherche locale donnée est utilisé pour mettre à jour à la fois la stratégie d'apprentissage du MCTS et le sélecteur d'opérateurs. Enfin, nous étudions l'impact des différents opérateurs de recherche locale et des différentes stratégies d'apprentissage en ligne, y compris un réseau de neurones prenant en compte non seulement les scores passés des composants de bas niveau comme dans [Dantas *et al.*, 2021], mais aussi l'état brut de la solution initiale à améliorer par une heuristique de recherche locale. Ce réseau neuronal est rendu invariant par permutation des groupes de couleurs en s'inspirant de l'architecture deep-set [Zaheer *et al.*, 2017], ce qui est une propriété intéressante pour les problèmes de coloration.

7.2 MCTS avec Stratégie de Simulation Adaptative

Cette section présente le framework général de la sélection adaptative des opérateurs de recherche locale combinée avec l'algorithme MCTS présenté dans [Grelier *et al.*, 2022]. Les différentes hyperheuristiques utilisées pour sélectionner les opérateurs de recherche locale sont ensuite présentées.

7.2.1 Framework Général

L'approche hyperheuristique proposée est détaillée dans l'Algorithme 9. Il prend en entrée un graphe pondéré $G = (V, E, w)$, un ensemble d'opérateurs de recherche locale $\mathcal{O} = \{o_1, \ldots, o_d\}$ et une stratégie de sélection de haut niveau $\pi_\theta : \Omega \to \mathcal{O}$ prenant en entrée une solution complète et légale S et donnant en sortie un opérateur de recherche locale $o \in \mathcal{O}$ à appliquer à S.

La stratégie de sélection π_θ est paramétrée par un vecteur de paramètres θ initialisé aléatoirement au début de la recherche. Ensuite, l'algorithme répète une boucle (itération) jusqu'à ce qu'une limite de temps soit atteinte ou qu'une solution avec un score égal à l'optimal soit atteint.

Algorithme 9 MCTS avec stratégie de simulation adaptative

Entrée: Graphe pondéré $G = (V, E, w)$, $\mathcal{O}$ un ensemble d'opérateurs de recherche locale et π_θ une stratégie de sélection.
Sortie: The best legal coloring S^* found

$S^* = \emptyset$ et $f(S^*) = MaxInt$
$D = \emptyset$.

tant que tant que la condition d'arrêt n'est pas satisfaite **faire**
    $C \leftarrow R$ ▷ noeud courant correspondant au noeud racine de l'arbre
    $S \leftarrow \{V_1, U\}$ avec $V_1 = \{v_1\}$ et $U = V \backslash V_1$ ▷ premier sommet dans le premier groupe de couleurs

    /* Étape 1 - Sélection */
    tant que C n'est pas une feuille **faire**
        $C \leftarrow$ select_best_child(C) avec une coloration légale $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$

    /* Étape 2 - Expansion */
    si C a un enfant potentiel, pas encore ouvert **alors**
        $C \leftarrow$ open_first_child_not_open(C) avec une coloration légale $< u, U, V_i >$
        $S \leftarrow S \oplus < u, U, V_i >$

    /* Étape 3 - stratégie de simulation adaptive */
    $S \leftarrow$ greedy(S) ▷ **compléter la solution courante avec un algorithme glouton**
    $o = \pi_\theta(S)$ ▷ **sélectionner un opérateur de recherche locale**
    $S' \leftarrow o(S)$ ▷ **améliorer la solution avec l'opérateur sélectionné**
    $D \leftarrow D \cup (S, o, -f(S'))$ ▷ **stocker un exemple d'apprentissage dans la base de données**
    $\theta \leftarrow learning(D, \theta)$ ▷ **apprentissage *online* de la stratégie de sélection adaptive**

    /* Étape 4 - Mise à jour */
    tant que $C \neq R$ **faire**
        update(C,f(S'))
        $C \leftarrow$ parent(C)
    si $f(S') < f(S^*)$ **alors**
        $S^* \leftarrow S'$

    /* Étape 5 - Élagage */
    application des règles d'élagage

retourne S^*

Nous rappelons ici que chaque itération du MCTS implique l'exécution de cinq étapes :

1. **Sélection** : à partir du nœud racine de l'arbre, les nœuds enfants les plus prometteurs sont sélectionnés de manière itérative jusqu'à ce qu'un nœud feuille soit atteint (un nœud sans tous les enfants ouverts). La sélection d'un nœud enfant à chaque niveau correspond au choix de la couleur pour le prochain sommet du graphe [1]. Le nœud le plus prometteur est sélectionné en fonction d'un compromis exploitation/exploration UCT (Upper Confidence bounds for Trees).
2. **Expansion** : l'arbre MCTS se développe en ajoutant un nouveau nœud enfant au nœud feuille atteint lors de la phase de sélection en respectant la borne supérieure sur le nombre de couleurs établie dans la Chapitre 4.
3. **simulation adaptive** : la solution partielle courante est complétée avec des mouvements légaux gloutons pour obtenir une solution complète S. Ensuite, l'opérateur sélectionné $o = \pi_\theta(S)$ est appliqué à la solution courante S pour l'améliorer pendant un temps fixe. Cela conduit à une nouvelle solution S' qui est la meilleure sur la trajectoire réalisée par la recherche locale. À partir de cette trajectoire, un nouvel exemple d'apprentissage (S, o, r) est construit, avec une récompense $r = -f(S')$. Cet exemple d'apprentissage est stocké dans une base de données D. Toutes les nb itérations, la stratégie de sélection est mise à jour sur cette base de données D.
4. **Mise à jour** : après la simulation, le score moyen et le nombre de visites de chaque nœud sur la branche explorée sont mis à jour.
5. **Élagage** : si un nouveau meilleur score est trouvé, certaines branches de l'arbre MCTS peuvent être élaguées s'il n'est pas possible d'améliorer le meilleur score actuel avec elles.

Dans ce qui suit, seule l'étape 3, avec la stratégie de simulation adaptative écrite en gras dans l'Algorithme 9, sera décrite en détail. Nous renvoyons le lecteur aux deux chapitres précédents pour plus de détail sur les autres phases.

7.2.2 Stratégie de Simulation Adaptative

Comme le montrent [Grelier *et al.*, 2022] et le chapitre précédent, aucune procédure de recherche locale ne domine les autres pour le WVCP, et certaines procédures sont plus performantes pour certains types d'instances que d'autres.

1. Les sommets sont considérés dans l'ordre décroissant de leur poids puis de leur degré.

Sélection dynamique d'opérateurs. Pour choisir le meilleur opérateur de recherche locale possible pendant la recherche, une stratégie de haut niveau π_θ sélectionne automatiquement un opérateur $o = \pi_\theta(S)$ dans $\mathcal{O}$ à appliquer à la solution courante S. Le choix de cet opérateur peut dépendre de la solution S à améliorer. L'ensemble $\mathcal{O}$ des composants heuristiques de bas niveau considérés pour la stratégie de simulation est constitué des quatre procédures de recherche locale dédiées à la résolution du WVCP présentées dans les Chapitres 2 et 6 : *TabuWeight* [Grelier *et al.*, 2022], *AFISA* [Sun *et al.*, 2018], *RedLS* [Wang *et al.*, 2020] et *ILS-TS* [Nogueira *et al.*, 2021]. Les différentes stratégies de haut niveau π_θ utilisées dans ce travail sont présentées dans la Section 7.3.

Collecte d'exemples d'apprentissage. Les opérateurs de recherche locale o sont appliqués à la solution courante S pendant un temps fixe. Peu importe l'espace de recherche utilisé, ils renvoient ensuite la meilleure solution légale trouvée, notée $S' = o(S)$. Un nouvel exemple d'apprentissage (S, o, r), associé à la récompense $r = -f(S')$, est alors stocké dans une base de données D. La récompense r est négative, car le WVCP est un problème de minimisation. Dans la littérature sur les hyperheuristiques, la récompense apportée par un opérateur donné est généralement proportionnelle à $f(S) - f(S')$ (pour un problème de minimisation), c'est-à-dire la différence entre le score avant et après l'application de l'opérateur [Goëffon *et al.*, 2016, Dantas *et al.*, 2021]. Cependant, le MCTS peut produire des solutions de qualité différente à chaque itération. Si l'on prenait en compte le score de la solution S à partir de laquelle la recherche locale commence dans l'évaluation de la récompense, les opérateurs commençant à partir d'une solution déjà bonne (faible valeur de $f(S)$) seraient désavantagés par rapport à ceux commençant à partir d'une mauvaise solution (valeur élevée de $f(S)$). En effet, il est généralement comparativement plus facile d'améliorer une solution de mauvaise qualité que d'améliorer une solution déjà de bonne qualité. Donc, dans cette étude, nous avons choisi d'attribuer une récompense qui dépend uniquement du meilleur score trouvé $f(S')$ au cours de la trajectoire réalisée dans l'espace de recherche par un opérateur donné.

Apprentissage *online* du sélecteur d'opérateur de haut niveau. Toutes les nb itérations du MCTS, la politique π_θ est entraînée sur la base de données D et l'ensemble de ses paramètres θ est mis à jour. La base de données D est modélisée comme une file d'attente de taille N correspondant aux N derniers exemples obtenus au cours des itérations précédentes. Cette file de taille limitée permet de mieux s'adapter aux variations potentielles des résultats des opérateurs, au cas où certains opérateurs sont meilleurs au début de la recherche qu'à la fin.

7.3 Sélecteurs d'Opérateurs

Nous présentons maintenant les différentes politiques de sélection d'opérateurs π_θ utilisées dans ce travail.

7.3.1 Sélecteur avec Réseau de Neurones

Nous proposons un nouveau sélecteur basé sur un réseau de neurones, voir Figure 7.1.

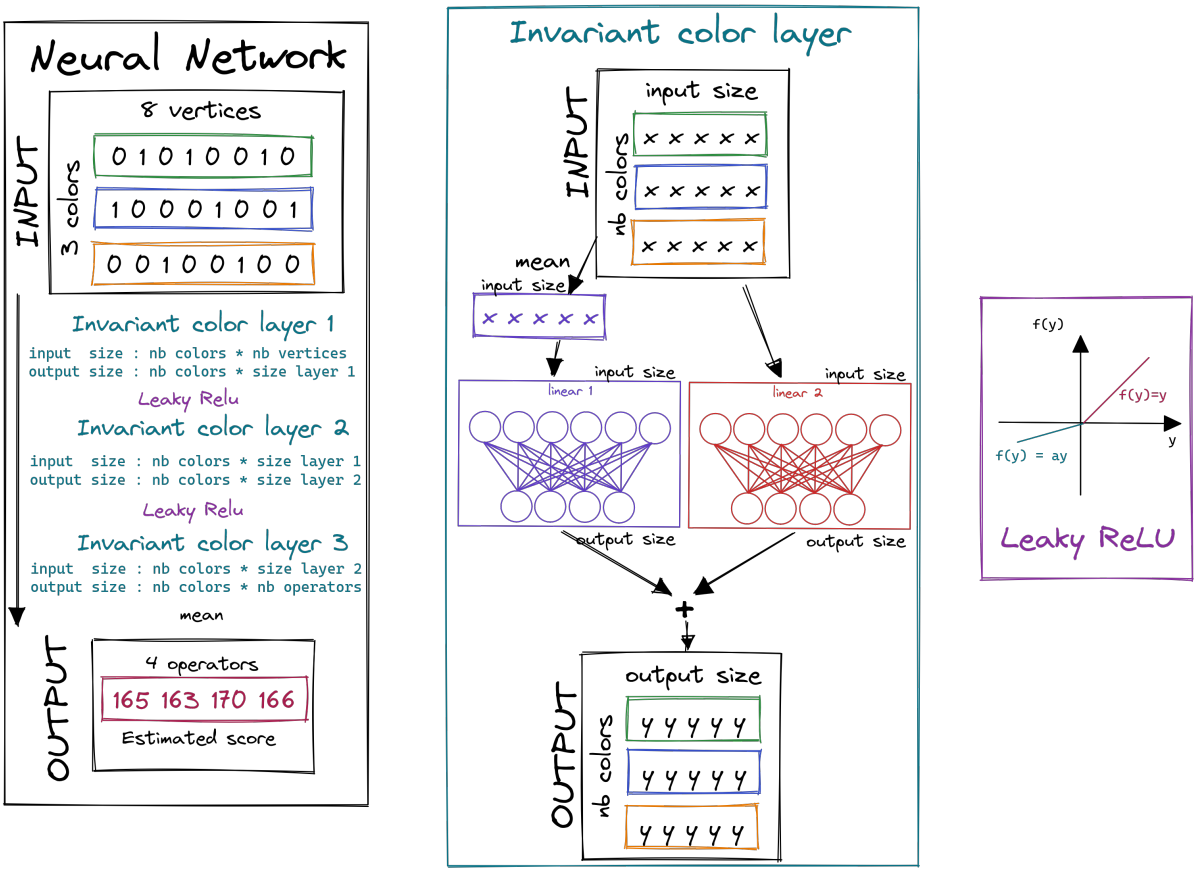


FIGURE 7.1 – À gauche, schéma du réseau de neurones utilisé. L'entrée (input) est une solution brute sous la forme d'une matrice de taille $k \times |V|$. Chaque ligne correspond à un groupe de couleurs et chaque colonne à un sommet. Un 1 est présent sur la ligne i et la colonne j si le sommet j appartient au groupe de couleurs i. La solution passe dans les différentes couches invariantes par permutation des couleurs avec des fonctions d'activation LeakyReLU. La sortie (output) est un vecteur de taille $|\mathcal{O}|$ avec le score espéré après application de chaque recherche locale. Au milieu, une représentation d'une couche invariante par permutation des couleurs. À droite, exemple de la fonction d'activation LeakyReLU.

NN (Neural Net, Figure 7.1) Sélecteur basé sur les recommandations d'un réseau de neurones pour une solution brute en entrée et donnant une estimation des résultats qu'il est possible d'obtenir après l'application de chaque procédure de recherche locale.

Dans ce cas, la fonction de sélection $\pi_\theta : \Omega \to \mathcal{O}$, utilise un réseau de neurones g_θ, paramétré par un vecteur θ (initialisé aléatoirement au début). Ce réseau de neurones g_θ prend en entrée une coloration S sous la forme d'un ensemble de k vecteurs binaires $\mathbf{v_j}$ de taille n, $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, où chaque $\mathbf{v_j}$ indique les sommets appartenant au groupe de couleurs j. À partir d'une telle entrée, le réseau de neurones produit un vecteur de $|\mathcal{O}|$ valeurs dans $\mathbb{R}$ correspondant à la récompense attendue de chaque opérateur $o \in \mathcal{O}$. Ensuite, avec une probabilité de 90%, l'opérateur associé à la récompense attendue maximale est sélectionné. Nous gardons toujours une probabilité de 10% de sélectionner un opérateur au hasard, pour assurer un minimum de diversité dans la sélection des opérateurs (exploration).

L'architecture *deep set* [Zaheer *et al.*, 2017, Lucas *et al.*, 2018, Goudet *et al.*, 2022] utilisée permet à la sortie du réseau d'être invariante par permutation des groupes de couleurs. Ainsi, le sélecteur avec réseau de neurones prendra la même décision pour deux couleurs d'entrée S et S_σ qui sont équivalentes à une permutation près des groupes de couleurs, notée σ. Plus précisément, pour toute permutation σ des groupes de couleurs d'entrée, nous avons :

$$g_\theta(\mathbf{v}_{\sigma(\mathbf{1})}, \ldots, \mathbf{v}_{\sigma(\mathbf{k})}) = g_\theta(\mathbf{v_1}, \ldots, \mathbf{v_k}). \tag{7.1}$$

Pour une coloration $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, le réseau de neurones invariant par permutation des groupes de couleurs g_θ est défini comme :

$$g_\theta(S) = \frac{1}{k} \sum_{i=1}^{k} (\phi_{\theta_P} \circ \phi_{\theta_{P-1}} \circ \cdots \circ \phi_{\theta_0}(S))_i, \tag{7.2}$$

où chaque ϕ_{θ_j} est une fonction invariante par permutation de $\mathbb{R}^{k \times l_{j-1}}$ à $\mathbb{R}^{k \times l_j}$ avec l_j étant la taille des couches. Notez que nous avons $l_{j-1} = |V|$ pour la première couche et $l_{j-1} = |V|$ et $l_j = |\mathcal{O}|$ pour la dernière couche. Voir [Lucas *et al.*, 2018] pour plus de détails sur la fonction invariante par permutation ϕ_{θ_j}.

Dans ce travail, nous utilisons un réseau de neurones avec deux couches cachées de taille $h1 = |V|$ et $h2 = \frac{|V|+|\mathcal{O}|}{2}$, et une fonction d'activation non linéaire définie comme $\mu(x) = \max(0.2 \times x, x)$ (LeakyReLU).

Pendant la phase d'entraînement, chaque exemple d'apprentissage (S, o, r) de la base de données D est converti en un exemple d'apprentissage supervisé (X, y), avec X une matrice d'entrée de taille $k \times |V|$ correspondant à l'ensemble des k vecteurs $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, et y est

un vecteur réel de taille $|\mathcal{O}|$ (nombre d'opérateurs candidats), de sorte que y est initialisé à la valeur $g_\theta(X)$ pour chaque opérateur, puis la valeur $y[o]$ pour l'opérateur choisi o est remplacée par la récompense attendue r : $y[o] = r$.

Toutes les $nb = 10$ itérations, le réseau de neurones est entraîné avec l'erreur quadratique moyenne sur les $|\mathcal{O}|$ sorties (apprentissage supervisé) sur l'ensemble de données D pendant 15 époques avec l'optimiseur Adam [Kingma et Ba, 2014] et un taux d'apprentissage de 0.001.

7.3.2 Sélecteurs Classiques Basés sur la Fitness

Nous comparons le sélecteur de réseau de neurones ci-dessus avec cinq sélecteurs, qui ne prennent pas en entrée la solution brute pour faire leur choix. Ces sélecteurs sont deux critères basés sur un biais proportionnel (*Roulette* et *Pursuit*), un critère basé sur la méthode du bandit à un bras (*UCB*), un purement aléatoire (*Random*) et un supprimant un à un les opérateurs les moins performants pour n'en garder qu'un (*Deleter*). *Roulette*, *Pursuit* et *UCB* ont été largement utilisés dans la littérature des hyperheuristiques (par exemple [Thierens, 2005, Hoos, 2012, Goëffon *et al.*, 2016]).

Pour ces trois opérateurs, la récompense r de chaque exemple d'apprentissage (S, o, r) est normalisée entre 0 et 1 sur la fenêtre glissante constituée des N derniers exemples stockés dans la base de données D, 0 étant la pire valeur obtenue au cours des N dernières itérations (correspondant à la plus haute valeur de la fitness $f(S')$) et 1 étant la meilleure valeur. Cette récompense moyenne normalisée calculée sur cette fenêtre glissante et associée à un opérateur o est écrite r_o. n_o désigne le nombre de fois où l'opérateur o a été sélectionné au cours des N dernières itérations.

Random Sélecteur aléatoire avec une sélection uniforme sur tous les opérateurs. Chaque opérateur a une chance $\frac{1}{|\mathcal{O}|}$ d'être sélectionné.

Deleter Sélecteur supprimant définitivement toutes les 5 itérations l'opérateur avec la plus mauvaise moyenne de résultats jusqu'à n'en garder qu'un (il y a $5 * |\mathcal{O}|$ itérations sans suppression en début de recherche).

Sélection avec Biais Proportionnel

Roulette et *Pursuit* sont deux stratégies basées sur des calculs de probabilité, inspirées de [Goëffon *et al.*, 2016]. Dans les deux cas, π_θ est une procédure aléatoire pilotée par un vecteur

de $|\mathcal{O}|$ paramètres $\theta = [p_1, \ldots, p_{|\mathcal{O}|}]$, de sorte que p_i correspond à la probabilité que l'opérateur i soit choisi ce tour. Nous avons $\sum_{i=1}^{|\mathcal{O}|} p_i = 1$. Au début, toutes les probabilités sont fixées à une valeur égale. Par conséquent, $p_i = \frac{1}{|\mathcal{O}|}$, pour $i = 1, \ldots, |\mathcal{O}|$.

Plus un opérateur o obtient de bonnes récompenses r_o, plus sa probabilité associée p_i augmente et plus il est choisi dans les itérations suivantes.

Roulette (ou Adaptive roulette wheel) Dans cette stratégie, les probabilités p_o pour $o = 1, \ldots, |\mathcal{O}|$ sont mises à jour à chaque itération selon la formule :

$$p_o = p_{min} + (1 - |\mathcal{O}| * p_{min}) * \frac{r_o}{\sum_{o'}^{|O|} r_{o'}},$$

où p_{min} est la probabilité de sélection minimale pour chaque opérateur (qui est fixée à une valeur strictement positive pour assurer un niveau minimum d'exploration). C'est un hyperparamètre de la méthode fixé à la valeur de $\frac{1}{5\times|\mathcal{O}|}$ dans ce travail.

Pursuit (ou Adaptive Pursuit) Dans cette stratégie, les probabilités sont mises à jour selon la formule :

$$\begin{cases} p_{o*} = p_{o*} + \beta(p_{max} - p_{o*}) \\ p_o = p_o + \beta(p_{min} - p_o), \end{cases} \tag{7.3}$$

où $o*$ est l'indice de l'opérateur le plus performant sur la fenêtre glissante, $p_{min} = \frac{1}{5\times|\mathcal{O}|}$, $p_{max} = 1 - (|O| - 1) * p_{min}$ et $\beta = 0.7$ est un coefficient introduit pour contrôler cette stratégie de type "le gagnant rafle tout" (*winner-take-all*).

Notez que contrairement à la stratégie de roulette adaptative *adaptive roulette*, qui donne des chances plus équilibrées à tous les opérateurs de bas niveau, la stratégie *Pursuit* est plus élitiste, car elle donne plus de chances à la meilleure stratégie appliquée dans les itérations précédentes.

UCB (ou bandit manchot) Cette politique π_θ est basée sur la méthode du bandit manchot. À chaque itération, selon la formule UCB (Upper Confidence Bound), l'opérateur avec le score le plus élevé $s_o = r_o + c * \sqrt{\frac{2*\log|D|}{n_o+1}}$ est sélectionné, où c est un hyperparamètre fixé à la valeur de 1. Cette formule correspond aussi à la stratégie utilisée lors de la phase de sélection dans le MCTS.

7.4 Expérimentations

Cette section présente les expériences menées sur les différents critères de sélection.

7.4.1 Paramètres Expérimentaux

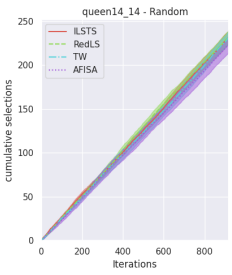
Nous considérons les 188 instances de référence WVCP de la littérature en version réduite. Pour les 142 instances dont le score optimal est connu, les méthodes chercheront à atteindre ce score et s'arrêteront. Pour les méthodes combinées, MCTS avec recherche locale, avec et sans sélection adaptative, une limite de temps d'une heure a été fixée pour la recherche locale, afin de comparer toutes les méthodes avec le même temps total passé dans les recherches locales. Le temps passé dans la recherche locale à chaque itération est fixé à $|V| * 0.02$ s.

Nous effectuons 20 exécutions indépendantes de chaque méthode sur chaque instance sur un ordinateur équipé d'un processeur Intel Xeon ES 2630, 2,66 GHz. Tous les algorithmes sont codés en C++, compilés et optimisés avec le compilateur g++ 12.1. Le code source et les résultats complets sur toutes les instances sont disponibles à l'adresse `https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts`. Des changements dans les résultats peuvent être observés par rapport à la publication [Grelier *et al.*, 2023] dus à des optimisations dans le code, l'utilisation du flag LTO à la compilation, ainsi qu'à une réduction plus complète des instances. Pour les réseaux de neurones, la librairie C++ de Pythorch 1.11 a été utilisée.

Dans ce qui suit, lorsqu'une méthode est dite meilleure qu'une autre sur une instance donnée, cela signifie que la différence entre les scores moyens calculés sur 20 exécutions est en faveur de la première méthode et que cette différence est significative (test de rang signé de Wilcoxon non paramétrique avec une valeur p $\leq 0,001$).

7.4.2 Sélection Adaptative d'Opérateurs pendant la Recherche

La Figure 7.2 montre la moyenne de la sélection cumulative de chaque opérateur avec des barres d'erreur pendant les 20 exécutions sur cinq instances difficiles pour chaque critère : *Deleter*, *Roulette*, *Pursuit*, *UCB* et *NN*. Le critère *Random* est le même sur toutes les instances et est affiché sur la figure à droite de ce texte. Les quatre algorithmes de recherche locale qui peuvent être sélectionnés à chaque itération du MCTS sont *ILS-TS* (ligne pleine rouge), *RedLS* (ligne avec tirets verts), *TabuWeight*/TW (ligne avec tirets et pointillés bleus) et *AFISA* (ligne en pointillés violet).



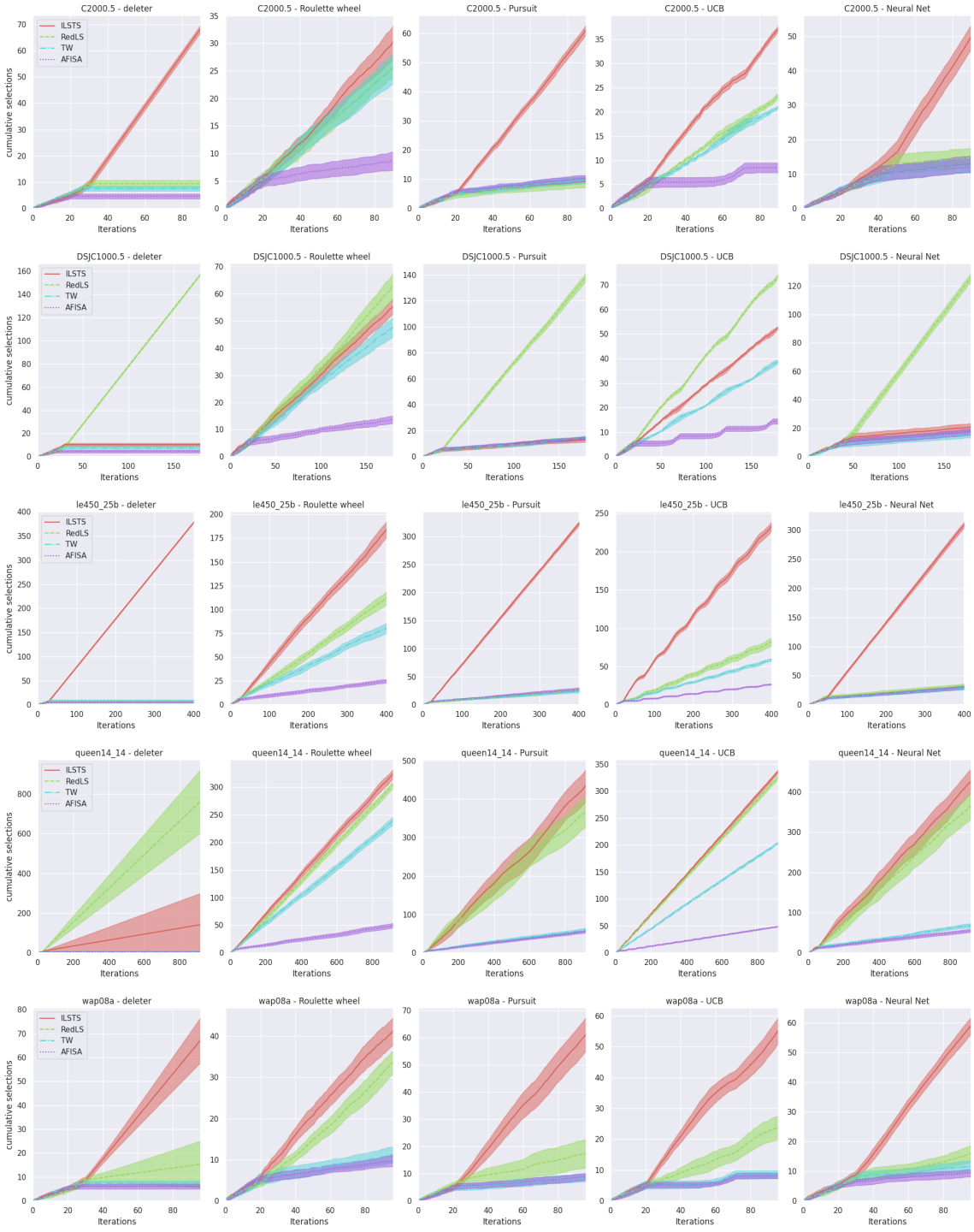


FIGURE 7.2 – Chaque graphique représente la moyenne sur 20 exécutions, pour une instance donnée, de la sélection cumulative des opérateurs effectuée par chaque politique de sélection, avec une barre d'erreur. Il y a une ligne par instance (5 instances différentes) et une colonne par méthode adaptative.

Nous observons d'abord que pendant les 20 premières itérations de l'algorithme, les choix effectués par les différentes stratégies de haut niveau sont aléatoires. Cela correspond au temps passé à collecter suffisamment d'exemples pour apprendre.

La Figure 7.2 montre que ce n'est pas toujours la même stratégie qui est choisie majoritairement pour les différentes instances. Les barres d'erreur peu larges montrent que la sélection est généralement cohérente sur les différentes exécutions, sauf pour wap08a ou queen14_14 pour laquelle les différentes stratégies oscillent beaucoup entre les méthodes *RedLS* et *ILS-TS* (qui sont les deux meilleures méthodes pour ces instances [Wang *et al.*, 2020, Nogueira *et al.*, 2021]) et donc pour cette instance la sélection cumulative peut varier beaucoup d'une exécution à l'autre. Ces courbes mettent en évidence que les différents sélecteurs d'opérateurs (à l'exception du sélecteur aléatoire) sont en général capables d'identifier la méthode la plus performante pour chaque instance donnée. *Roulette* illustre bien ce fait, car les courbes montrent une proportion logique entre la moyenne des résultats et le nombre de sélections de la manière la plus juste (avec un seuil minimum à $\frac{1}{5\times|\mathcal{O}|}$ soit 0.05 dans ce cas).

On peut observer différentes tendances dans la sélection des opérateurs en fonction de la politique. Tout d'abord, la sélection *Random* se déroule comme prévu, aléatoirement, nous observons que les nombres de sélections cumulés moyens sont presque égaux pour tous les opérateurs à différents pas de temps. *Roulette* conserve encore beaucoup de diversité dans la sélection des différents opérateurs, mais apprend un biais vers les meilleures méthodes pendant la recherche. La stratégie *UCB* se comporte en partie comme la stratégie *Roulette*. Notez qu'un coefficient d'exploration vs exploitation plus faible réduirait le nombre de fois où les pires opérateurs sont sélectionnés. Les stratégies *Deleter*, *Pursuit* et *NN* ont des comportements similaires : en général, les meilleurs opérateurs pour une instance donnée sont sélectionnés en priorité, les autres sont choisis au hasard ou jamais dans le cas de *Deleter*. Ces trois critères sont plus élitistes que les autres.

Nous remarquons aussi que les sélections sont cohérentes entre les différentes courbes, les opérateurs choisis sont les mêmes. À noter que sur l'ensemble des instances difficiles pour lesquelles suffisamment d'itérations sont réalisées avec les critères *UCB* et *Roulette* (qui sont les critères les plus équilibrés), *RedLS* et *ILS-TS* se partagent les deux premières places en termes de nombre de sélections avec un léger avantage pour *RedLS*. *TabuWeight* est sélectionné quelques fois en seconde place et très majoritairement en troisième place. *AFISA* est sélectionné très majoritairement en quatrième place.

7.4.3 Comparaisons à l'État de l'Art

	AFISA	MCTS+*AFISA*	*TabuWeight*	MCTS+TW	*RedLS*	MCTS+*RedLS*	*ILS-TS*	MCTS+*ILS-TS*	*Random*	*Deleter*	*Roulette*	*UCB*	*Pursuit*	*NN*
AFISA	-	35	**49**	17	**53**	0	0	0	0	0	0	0	0	0
MCTS+*AFISA*	**40**	-	**73**	10	**86**	0	1	0	0	0	0	0	0	0
TabuWeight	40	40	-	25	**48**	1	0	3	2	2	2	2	1	2
MCTS+TW	**74**	**73**	**78**	-	**101**	5	1	0	0	0	0	0	0	0
RedLS	41	35	47	29	-	11	14	15	12	11	12	13	11	11
MCTS+*RedLS*	**103**	**82**	**107**	**72**	**102**	-	**20**	**28**	5	5	4	3	2	4
ILS-TS	**104**	**82**	**106**	**75**	**96**	19	-	**18**	10	11	10	9	11	11
MCTS+*ILS-TS*	**102**	**82**	**103**	**73**	**92**	15	2	-	0	1	0	1	1	1
Random	**104**	**82**	**108**	**75**	**98**	**15**	**15**	**25**	-	2	0	0	**1**	1
Deleter	**104**	**82**	**108**	**75**	**99**	**14**	**17**	**23**	**5**	-	0	0	0	1
Roulette	**104**	**82**	**107**	**75**	**101**	**15**	**15**	**25**	0	**1**	-	0	1	**2**
UCB	**104**	**82**	**107**	**75**	**101**	**15**	**17**	**25**	0	**1**	0	-	**2**	**2**
Pursuit	**104**	**82**	**107**	**75**	**102**	**15**	**17**	**26**	0	**1**	1	1	-	**1**
NN	**104**	**82**	**107**	**75**	**101**	**15**	**18**	**26**	**4**	1	0	1	0	-

TABLE 7.1 – Comparaison de tous les solveurs de recherche locale et des variantes MCTS avec et sans sélection adaptative. Chaque valeur correspond au nombre d'instances où la méthode de la ligne est significativement meilleure que la méthode de la colonne sur les 188 instances de benchmark considérées (test de Wilcoxon signé avec une valeur $p \leq 0,001$). Un nombre est écrit en gras si le nombre d'instances est plus élevé pour la méthode de la ligne.

Le Tableau 7.1 affiche les comparaisons de performance entre chaque paire de méthodes sur les 188 instances de benchmark considérées dans ce travail. Les différents types de méthodes sont (i) les quatre solveurs de recherche locale seuls : *AFISA*, *TabuWeight*, *RedLS* et *ILS-TS* ; (ii) les versions MCTS avec un solveur de recherche locale fixe utilisé pour la simulation : MCTS+*AFISA*, MCTS+*TabuWeight*(TW), MCTS+*RedLS* et MCTS+*ILS-TS* ; (iii) les versions MCTS avec des stratégies de simulation adaptatives : *Random*, *Deleter*, *Roulette*, *UCB*, *Pursuit* et *NN*.

Le MCTS combiné à une recherche locale améliore les résultats de celle-ci à l'exception de *ILS-TS*. Parmi les différentes versions MCTS+LS, MCTS+*RedLS* est la plus efficace.

Contrairement aux résultats obtenus dans l'article [Grelier *et al.*, 2023], la stratégie *Random* n'est plus la pire parmi les stratégies adaptatives. Les optimisations dans le code permettant

de faire plus d'itérations dans la recherche locale ont réduit l'importance de l'apprentissage, sauf pour les stratégies *Deleter* et *NN* qui restent meilleures sur DSJC1000.5, DSJC500.9, DSJC250.9, latin_square_10 et queen15_15 pour le premier et DSJC1000.5, DSJC500.9, DSJC250.9 et flat1000_50_0 pour le second en étant moins bon sur queen10_10gb et wap06a. En revanche, les stratégies avec apprentissage sont plus souvent plus rapides pour trouver un même score.

Lorsque l'on compare les différentes stratégies adaptatives, on remarque assez peu de différences dans les résultats, tout comme dans les temps pour atteindre un même score. Contrairement au moment de la publication de l'article, ce sont les stratégies moins élitistes *Roulette* et *UCB* qui prennent très légèrement l'avantage sur quelques instances de quelques points sur la moyenne, la plus importante étant l'avantage de 19 points sur la moyenne entre *Roulette* et *NN*, pour les autres la différence à la moyenne est généralement inférieure à 2.

En général, bien que la sélection adaptative avec le réseau de neurones(*NN*) soit performante, l'avantage comparatif par rapport aux méthodes *Pursuit* ou *Deleter* plus basiques et élitistes n'apparaît pas dans ces expériences. Il ne semble pas y avoir d'avantage évident à choisir un opérateur en fonction de l'état brut de la solution à partir de laquelle la recherche commence. Une stratégie de sélection basée uniquement sur la fitness peut être tout aussi efficace dans ce contexte.

Le Tableau 7.2 affiche les résultats plus détaillés pour plusieurs méthodes testées dans ce travail. La colonne 1 indique le nom de l'instance. Pour limiter la taille du tableau, nous montrons ici les résultats pour un sous-ensemble d'instances relativement difficiles et de différents types. La colonne 2 montre le meilleur score connu (BKS) pour chaque instance donnée rapportée dans la littérature. Une étoile après ce score dans cette colonne indique que le score a été prouvé optimal. Notez que certains de ces BKS ont été obtenus dans des conditions spécifiques (temps de calcul de plus d'un jour et calcul parallèle sur un processeur GPU), tandis que les résultats rapportés ici sont atteints sur un seul CPU et avec une limite de temps d'une heure de recherche locale. Les colonnes 3-29 affichent les résultats pour une partie des différentes méthodes comparées dans ce travail. Pour des raisons d'espace, les résultats des versions *AFISA*, *TabuWeight*, MCTS+*AFISA*, MCTS+*TabuWeight* et MCTS+*ILS-TS* ne sont pas rapportés ici. Pour chaque méthode sont affichés les meilleurs et les scores moyens obtenus sur 20 exécutions, et le temps passé en secondes pour obtenir les meilleurs scores. Les scores en gras indiquent que la méthode a le meilleur score (best) ou la meilleure moyenne (avg) sur l'instance parmi les méthodes présentés ici. Les trois dernières lignes rappellent le nombre de meilleurs scores connus dans l'état de l'art trouvés par chaque méthode (#Best),

instance	BKS	*RedLS*			MCTS+*RedLS*			*ILS-TS*			*Random*			*Deleter*			*Roulette*			*UCB*			*Pursuit*			*NN*		
		best	avg	time	best	avg	time	best	avg	time	best	avg	time	best	avg	time	best	avg	time	best	avg	time	best	avg	time	best	avg	time
C2000.5	2144	**2167**	**2193.8**	2403	2354	2369	2091	2237	2266.4	3498	2336	2347.1	2870	2331	2342.2	2952	2330	2347.3	1845	2335	2349.8	1558	2332	2340.8	1271	2337	2346.2	6924
C2000.9	5477	**5502**	**5528.1**	3303	6060	6093.6	1107	5910	5969.9	3587	6035	6110.6	1968	6067	6105.4	492	6073	6109.4	2542	6077	6111.4	2952	6069	6103.4	205	6109	6128.5	7363
DSJC125.1gb	90*	91	93.9	0	**90**	**90**	385	**90**	**90**	18	**90**	**90**	63	**90**	**90**	49	**90**	**90**	46	**90**	**90**	53	**90**	**90**	43	**90**	**90**	52
DSJC125.1g	23*	**23**	23.4	0	**23**	**23**	7	**23**	**23**	3	**23**	**23**	11	**23**	**23**	13	**23**	**23**	10	**23**	**23**	12	**23**	**23**	9	**23**	**23**	9
DSJC125.5gb	240	245	260.5	0	241	241.2	870	**240**	**240**	107	**240**	**240**	332	**240**	**240**	177	**240**	**240**	294	**240**	**240**	148	**240**	**240**	164	**240**	**240**	274
DSJC125.5g	71	72	72	258	**71**	**71**	379	**71**	**71**	126	**71**	**71**	85	**71**	**71**	366	**71**	**71**	106	**71**	**71**	138	**71**	**71**	79	**71**	**71**	141
DSJC125.9gb	604*	**604**	**604**	2	**604**	**604**	2	**604**	**604**	168	**604**	**604**	15	**604**	**604**	8	**604**	**604**	13	**604**	**604**	9	**604**	**604**	16	**604**	**604**	15
DSJC125.9g	169*	**169**	**169**	0	**169**	**169**	0	**169**	**169**	183	**169**	**169**	9	**169**	**169**	7	**169**	**169**	9	**169**	**169**	8	**169**	**169**	12	**169**	**169**	10
DSJC250.1	127	130	131.6	1	**127**	**127.5**	1785	**127**	127.8	1609	**127**	128.1	1604	**127**	127.7	2271	**127**	127.8	2593	**127**	127.6	2125	**127**	127.8	1712	**127**	127.6	2094
DSJC250.5	392	398	401.2	103	396	397.9	3045	**393**	397.6	2567	397	399.1	1242	**393**	**397.5**	3558	397	398.6	1971	394	398.4	1746	396	397.8	1143	396	398.2	2701
DSJC250.9	934*	**934**	**935.6**	719	935	935.9	2406	936	942.1	3053	936	936.8	1593	935	936	2643	935	936.2	1802	935	936.1	1856	935	936.2	3054	**934**	**935.6**	2451
DSJC500.1	184	187	201.9	537	187	**187.8**	1765	188	188.8	1744	**186**	188.4	2079	**186**	188.2	3773	187	188.4	2503	187	188.4	3861	187	188.5	2079	187	188.2	3043
DSJC500.5	685	**706**	**716.1**	2840	716	719.2	3234	724	735.5	1744	715	720.2	964	715	719	3234	715	720.7	1045	712	720.6	880	715	719.8	715	713	718.9	631
DSJC500.9	1662	**1670**	**1675.1**	946	1687	1694.7	726	1720	1742	3039	1694	1697.5	2229	1685	1692.9	847	1689	1694.5	2622	1691	1695.4	1881	1690	1694.3	1227	1688	1693.5	3134
DSJC1000.1	300	305	307.2	1235	307	307.9	1799	305	307.4	1575	304	306.2	3570	304	305.8	1190	**302**	305.6	588	304	305.9	2835	305	305.4	2151	303	**305.1**	4617
DSJC1000.5	1185	**1198**	**1213.7**	2381	1244	1248.5	1197	1245	1269.2	231	1245	1251.4	756	1238	1246.6	1428	1241	1251.4	2142	1242	1250	1722	1239	1247.8	1554	1240	1247.6	3159
DSJC1000.9	2836	**2840**	**2858.5**	2953	2974	2992.2	651	3026	3066.8	3580	2982	2995.9	1890	2983	2992.5	3486	2984	2995.2	2982	2983	2995.1	504	2983	2995.2	2058	2976	2993.2	1499
DSJR500.1	169*	171	184.5	0	**169**	**169**	318	**169**	**169**	0	**169**	**169**	14	**169**	**169**	19	**169**	**169**	15	**169**	**169**	20	**169**	**169**	16	**169**	**169**	16
flat1000_50_0	924	**1155**	**1173.8**	3238	1199	1206.2	336	1222	1235	1713	1203	1210.7	3045	1191	1206.2	630	1201	1208.1	945	1205	1210.6	756	1198	1207.8	3654	1196	1204.6	4370
flat1000_60_0	1162	**1191**	**1205.7**	1080	1238	1244.5	1491	1250	1270	125	1235	1249.5	3045	1242	1247.5	525	1241	1248.2	1995	1232	1247.8	882	1239	1245.3	1701	1238	1245.1	1393
flat1000_76_0	1165	**1176**	**1194**	1107	1214	1222	1176	1232	1247.5	1198	1210	1224.9	2751	1217	1222.1	1470	1211	1223.8	168	1218	1226.2	504	1214	1223	2121	1211	1222.5	4602
GEOM110a	97*	**97**	106.2	0	**97**	**97**	16	**97**	**97**	0	**97**	**97**	7	**97**	**97**	8	**97**	**97**	8	**97**	**97**	8	**97**	**97**	7	**97**	**97**	7
GEOM110b	37*	**37**	38.4	0	**37**	**37**	3	**37**	**37**	0	**37**	**37**	2	**37**	**37**	3	**37**	**37**	2	**37**	**37**	3	**37**	**37**	4	**37**	**37**	2
GEOM110	68*	**68**	70.8	0	**68**	**68**	3	**68**	**68**	0	**68**	**68**	4	**68**	**68**	3	**68**	**68**	3	**68**	**68**	3	**68**	**68**	3	**68**	**68**	3
GEOM120a	105*	**105**	109.2	9	**105**	**105**	9	**105**	**105**	0	**105**	**105**	9	**105**	**105**	6	**105**	**105**	7	**105**	**105**	8	**105**	**105**	7	**105**	**105**	8
GEOM120b	35*	**35**	35.5	0	**35**	**35**	1	**35**	**35**	0	**35**	**35**	4	**35**	**35**	5	**35**	**35**	4	**35**	**35**	5	**35**	**35**	7	**35**	**35**	3
GEOM120	72*	**72**	75.2	0	**72**	**72**	14	**72**	**72**	0	**72**	**72**	3	**72**	**72**	8	**72**	**72**	5	**72**	**72**	4	**72**	**72**	3	**72**	**72**	2
inithx.i.1	569*	**569**	**569**	12	**569**	**569**	6	**569**	**569**	16	**569**	**569**	9	**569**	**569**	8	**569**	**569**	7	**569**	**569**	10	**569**	**569**	7	**569**	**569**	7
inithx.i.2	329*	**329**	329.1	312	**329**	**329**	52	**329**	**329**	9	**329**	**329**	49	**329**	**329**	33	**329**	**329**	40	**329**	**329**	32	**329**	**329**	36	**329**	**329**	45
inithx.i.3	337*	**337**	341.6	75	**337**	**337**	135	**337**	**337**	29	**337**	**337**	145	**337**	**337**	99	**337**	**337**	83	**337**	**337**	70	**337**	**337**	89	**337**	**337**	114
latin_square_10	1480	**1505**	**1523**	2369	1533	1544.8	1938	1559	1581.2	1369	1546	1552.9	1406	1533	1544.2	1606	1540	1550.6	3268	1522	1546.8	456	1529	1547.2	798	1538	1546.2	4558
le450_15a	212	214	236.3	218	**212**	**213.7**	684	213	214.7	2429	213	214.2	1554	213	215.2	3141	213	214	1022	213	214.2	873	213	213.8	1866	214	214.2	1530
le450_15b	216	218	225.6	39	217	**217.6**	1115	**216**	217.8	2536	217	217.9	968	217	218.4	783	217	217.8	1319	217	217.7	1930	217	217.8	2594	217	217.8	2540
le450_15c	275	281	285.8	34	279	**280.8**	1415	280	283.7	1668	**278**	281.6	1910	279	**280.8**	790	279	281.7	750	280	281.2	1066	279	281.2	3670	279	281.4	2630
le450_15d	272	278	281	268	276	**277.1**	2153	277	279.9	1445	276	278.3	2590	276	277.4	1480	276	277.9	1620	277	278	1842	**275**	**277.1**	870	**275**	277.2	1146
le450_25a	306	**306**	307.4	504	**306**	**306**	132	**306**	**306**	175	**306**	**306**	453	**306**	**306**	330	**306**	**306**	349	**306**	**306**	406	**306**	**306**	292	**306**	**306**	424
le450_25b	307*	**307**	313.4	56	**307**	**307**	87	**307**	**307**	10	**307**	**307**	63	**307**	**307**	75	**307**	**307**	59	**307**	**307**	52	**307**	**307**	61	**307**	**307**	47
le450_25c	342	351	354.8	44	348	**349.4**	383	348	351.8	3207	349	350.2	945	**347**	**349.4**	2147	350	350.3	1543	348	349.8	2052	**347**	349.5	2889	**347**	**349.4**	3388
le450_25d	330	**332**	338.9	154	334	**335.9**	1436	339	342.5	1999	336	336.8	1403	334	336.1	1935	335	336.7	1512	333	336	2394	335	**335.9**	1386	335	336	2054
queen10_10	162	**162**	166.2	505	**162**	**162**	38	**162**	**162**	14	**162**	**162**	52	**162**	**162**	51	**162**	**162**	36	**162**	**162**	37	**162**	**162**	31	**162**	**162**	34
queen10_10gb	164	165	166.2	285	**164**	164.8	1042	**164**	**164**	140	**164**	**164**	724	**164**	164.3	1243	**164**	**164**	566	**164**	**164**	513	**164**	164.1	771	**164**	164.1	534
queen10_10g	43*	**43**	43.2	19	**43**	**43**	9	**43**	**43**	4	**43**	**43**	16	**43**	**43**	15	**43**	**43**	19	**43**	**43**	20	**43**	**43**	17	**43**	**43**	14
queen11_11	172	174	177.1	762	**172**	173.2	1935	**172**	**172.6**	1821	**172**	172.9	2011	**172**	172.8	1952	**172**	172.8	1960	**172**	172.8	1293	**172**	172.8	1370	**172**	172.7	2632
queen11_11g	176	177	178.9	12	**176**	**176**	756	**176**	**176**	338	**176**	**176**	949	**176**	**176**	464	**176**	**176**	437	**176**	**176**	684	**176**	**176**	634	**176**	**176**	560
queen11_11g	47	48	48.1	4	**47**	**47**	192	**47**	**47**	97	**47**	**47**	249	**47**	**47**	141	**47**	**47**	291	**47**	**47**	202	**47**	**47**	201	**47**	**47**	197
queen12_12	185	188	190.2	8	**185**	186.6	432	**185**	**186.1**	1261	186	186.7	1379	186	186.3	1506	186	186.5	1723	186	186.6	2051	186	186.4	1502	186	186.4	1369
queen12_12gb	191	193	195.3	1596	**191**	**191.3**	1443	**191**	**191.3**	1421	**191**	191.8	1737	**191**	**191.3**	1186	**191**	191.7	1527	**191**	191.4	1446	**191**	191.4	1672	**191**	191.8	1925
queen12_12g	50	**50**	51.9	1063	**50**	**50**	995	**50**	**50**	773	**50**	50.5	1514	**50**	50.1	1431	**50**	50.2	1491	**50**	**50**	1288	**50**	50.3	1360	**50**	50.2	1181
queen13_13	194	195	198.7	1528	**194**	194.6	1605	**194**	195.7	3476	**194**	194.9	2997	**194**	194.5	1668	**194**	194.8	2707	**194**	194.5	2734	**194**	194.6	1352	**194**	**194.3**	2140
queen14_14	215	217	222.4	18	**216**	**216.9**	918	**216**	217.3	2018	217	217.3	1408	**216**	**216.9**	2173	**216**	217.2	1868	**216**	217.3	3268	**216**	217.3	1168	217	217.3	1669
queen15_15	223	227	229.7	1110	225	226.2	718	227	228.4	2167	226	226.9	2274	226	226.2	1392	226	226.6	2088	225	226.4	2408	**224**	**226.1**	2590	225	226.2	2647
queen16_16	234	237	239.9	91	237	237.7	1813	238	240.2	2938	237	238.3	1998	**236**	**237.4**	2627	237	237.7	2104	**236**	237.8	3720	**236**	237.6	3726	**236**	237.7	1544
R75_1gb	70*	**70**	78.3	0	**70**	**70**	7	**70**	**70**	0	**70**	**70**	5	**70**	**70**	4	**70**	**70**	3	**70**	**70**	3	**70**	**70**	3	**70**	**70**	4
R75_1g	18*	**18**	19.1	0	**18**	**18**	4	**18**	**18**	0	**18**	**18**	3	**18**	**18**	5	**18**	**18**	3	**18**	**18**	5	**18**	**18**	5	**18**	**18**	5
R75_5gb	186*	**186**	192.1	0	**186**	**186**	9	**186**	**186**	1	**186**	**186**	8	**186**	**186**	7	**186**	**186**	6	**186**	**186**	8	**186**	**186**	6	**186**	**186**	6
R75_5g	51*	**51**	51.3	109	**51**	**51**	6	**51**	**51**	0	**51**	**51**	1	**51**	**51**	2	**51**	**51**	1	**51**	**51**	1	**51**	**51**	1	**51**	**51**	1
R75_9gb	396*	**396**	**396**	0	**396**	**396**	0	**396**	**396**	10	**396**	**396**	6	**396**	**396**	9	**396**	**396**	5	**396**	**396**	6	**396**	**396**	5	**396**	**396**	6
R75_9g	110*	**110**	**110**	0	**110**	**110**	0	**110**	**110**	0	**110**	**110**	1	**110**	**110**	1	**110**	**110**	1	**110**	**110**	0	**110**	**110**	1	**110**	**110**	1
R100_1gb	81*	82	84.5	0	**81**	**81**	59	**81**	**81**	2	**81**	**81**	13	**81**	**81**	10	**81**	**81**	10	**81**	**81**	10	**81**	**81**	12	**81**	**81**	11
R100_1g	21*	**21**	22.1	0	**21**	**21**	53	**21**	**21**	8	**21**	**21**	19	**21**	**21**	24	**21**	**21**	21	**21**	**21**	25	**21**	**21**	26	**21**	**21**	21
R100_5gb	220	**220**	220.7	87	**220**	**220**	33	**220**	**220**	6	**220**	**220**	19	**220**	**220**	19	**220**	**220**	19	**220**	**220**	16	**220**	**220**	19	**220**	**220**	17
R100_5g	59	**59**	**59**	17	**59**	**59**	6	**59**	**59**	0	**59**	**59**	5	**59**	**59**	5	**59**	**59**	4	**59**	**59**	5	**59**	**59**	4	**59**	**59**	4
R100_9gb	518*	**518**	**518**	0	**518**	**518**	1	**518**	518.8	1853	**518**	**518**	13	**518**	**518**	8	**518**	**518**	15	**518**	**518**	10	**518**	**518**	9	**518**	**518**	10
R100_9g	141*	**141**	**141**	0	**141**	**141**	0	**141**	**141**	608	**141**	**141**	8	**141**	**141**	11	**141**	**141**	12	**141**	**141**	13	**141**	**141**	13	**141**	**141**	18
wap01a	545	563	688.4	252	561	586.6	2860	**548**	**552**	3263	560	590.3	2596	587	591.6	2420	567	589.6	352	570	590.1	3080	586	591.5	1672	567	589.5	440
wap02a	538	552	586.5	589	551	556.6	90	**540**	**542.7**	2658	552	560.2	1215	554	558	945	551	559.8	945	549	558.5	2520	548	556.5	3105	551	559.1	5815
wap03a	562	**569**	**573.1**	2893	587	588.8	2085	576	578.6	3102	587	589.5	1656	587	589.5	3404	587	588.9	2024	587	589.3	3266	586	588.9	2300	588	589.9	8953
wap04a	563	**564**	**572.5**	3280	581	583.9	1287	569	574.6	3529	583	585.2	1906	583	584.8	2138	582	584	2525	582	584.1	2624	581	584.1	2277	582	585.1	1685
wap05a	541	543	544.5	970	543	544.8	1326	**541**	**543.4**	3306	544	545.8	1139	544	545.7	1397	545	545.6	1929	545	546	1649	543	545.4	578	543	545.3	1571
wap06a	516	**518**	590.4	1005	521	523.5	1332	519	**522.4**	1123	521	526.1	1188	526	529.7	270	523	527.1	756	522	526.6	972	524	528.3	2196	521	527.4	252
wap07a	555	729	745.7	0	566	632.6	770	**564**	**569.5**	2680	567	595.9	210	605	610.6	945	569	608.5	140	604	610.2	3290	567	606.4	70	568	606	2904
wap08a	529	**536**	613.9	3035	543	**545.8**	144	543	549.5	3585	544	546.9	2844	544	551	288	545	546.9	1836	544	547.4	2736	544	547.1	828	545	547.9	1215
p41	2688*	2718	2787.3	0	**2688**	**2688**	530	**2688**	**2688**	0	**2688**	**2688**	9	**2688**	**2688**	9	**2688**	**2688**	10	**2688**	**2688**	7	**2688**	**2688**	10	**2688**	**2688**	8
p42	2466*	**2466**	2522.5	0	**2466**	**2466**	109	**2466**	**2466**	4	**2466**	**2466**	18	**2466**	**2466**	19	**2466**	**2466**	23	**2466**	**2466**	11	**2466**	**2466**	16	**2466**	**2466**	18
r29	8693*	8696	8817.6	0	**8693**	**8693**	539	**8693**	**8693**	1	**8693**	**8693**	23	**8693**	**8693**	14	**8693**	**8693**	18	**8693**	**8693**	13	**8693**	**8693**	17	**8693**	**8693**	21
r30	9816*	9836	9988.2	1	**9816**	9824	2570	**9816**	**9816**	6	**9816**	**9816**	47	**9816**	**9816**	30	**9816**	**9816**	35	**9816**	**9816**	35	**9816**	**9816**	27	**9816**	**9816**	20
#BKS /188			112			152			159			156			156			156			156			156			157		
#Best Score /188			127			153			164			158			161			158			158			161			160		
#Best Avg /188			44			142			159			151			156			151			152			153			154		

TABLE 7.2 – Résultats détaillés pour quelques instances sur les meilleures méthodes.

le nombre de fois où la méthode obtient le meilleur score par rapport aux autres méthodes (#Best Score) et le nombre de fois où la méthode obtient le meilleur score moyen par rapport aux autres méthodes (#Best Avg). Les résultats détaillés pour l'ensemble des 188 instances et pour toutes les méthodes sont disponibles avec le code source à l'adresse `https://github.com/Cyril-Grelier/gc_wvcp_adaptive_mcts`.

Nous observons d'abord que les deux meilleurs solveurs de recherche locale *RedLS* et *ILS-*

TS sont très performants, mais pas pour le même type d'instance. Plus précisément, l'algorithme *RedLS* est meilleur pour les grandes instances telles que latin_square_10, les flat, les plus grands DSJC et les C2000, tandis que *ILS-TS* est meilleur pour les plus petites instances telles que les DSJC250.5 ou moins, les GEOM, plusieurs le450, les queens, les R75, les pxx et rxx mais aussi sur plusieurs plus grandes parmi les wap. Cela montre que ces méthodes peuvent être complémentaires et qu'il peut être bénéfique de choisir la bonne en cours de recherche pour une instance donnée.

ILS-TS garde le meilleur ratio en nombre de meilleurs scores trouvés et de meilleures moyennes même face aux versions MCTS+hyperheuristiques (MCTS+HH). Cet algorithme, comme pour *RedLS*, reste très bon sur les wap ou les C2000 face aux MCTS+HH qui prennent l'avantage sur quelques autres instances difficiles comme latin_square_10, les flat et les DSJC.

7.5 Conclusion

Un framework de Monte Carlo Tree Search avec une stratégie de simulation adaptative a été présenté et testé sur le problème de coloration de graphe pondéré. Différents sélecteurs d'opérateurs de haut niveau ont été introduits dans ce travail.

Les résultats montrent que les versions MCTS avec sélection adaptative d'opérateurs atteignent un grand nombre de meilleurs scores pour l'ensemble des 188 instances de benchmark de la littérature. Cependant, *RedLS* et *ILS-TS* gardent l'avantage sur une dizaine d'instances de grande taille où le MCTS n'a pas le temps d'intensifier suffisamment.

L'analyse des sélections d'opérateurs pendant la recherche pour chaque instance particulière montre qu'en général le choix des opérateurs ne change pas pendant la recherche. Une fois que le meilleur solveur pour chaque instance a été identifié, il est généralement encore choisi pour le reste de la recherche. Ce manque de variation pendant la recherche peut s'expliquer par le fait que pour une instance donnée, il y a généralement un solveur dominant et nous n'observons pas de complémentarité dans l'utilisation des différents opérateurs pendant la recherche pour ce problème. Cela peut expliquer pourquoi le sélecteur avec réseau de neurones, *NN*, prenant en compte l'état brut de la solution actuelle n'apporte pas de meilleurs résultats que d'autres sélecteurs plus basiques basés sur la fitness uniquement.

Dans le chapitre suivant, nous étudions l'introduction des différentes stratégies de sélection d'opérateurs présentées dans ce chapitre, mais utilisées cette fois-ci dans le cadre d'un algorithme mémétique.

CHAPITRE 8

ALGORITHMES MÉMÉTIQUES ET HYPERHEURISTIQUES

Le chapitre précédent présentait une étude de l'apport d'une sélection *online* de procédures de recherche locale pour améliorer des solutions instanciées par un algorithme MCTS.

Dans ce chapitre, nous proposons d'étudier l'impact de cette sélection automatique d'opérateurs dans un cadre différent et très populaire pour résoudre les problèmes de coloration de graphe, à savoir les algorithmes mémétiques.

Dans ce contexte, l'algorithme choisira dynamiquement quelle recherche locale employer à chaque étape de la résolution, mais aussi quelle procédure de croisement utiliser pour recombiner les individus entre eux et générer de nouveaux descendants.

Cette approche originale combinant hyperheuristique et algorithme mémétique sera appliquée à la fois pour le GCP et pour le WVCP. Nous pourrons ainsi discuter des avantages et limites de cette approche pour ces deux problèmes qui ont été abordés avec des méthodes différentes dans la littérature.

8.1 Introduction

Comme vu dans les chapitres précédents, différents algorithmes constructifs et de recherche locale ont été utilisés dans la littérature pour résoudre le GCP et WVCP. Pour le GCP, on peut citer deux recherches locales particulièrement intéressantes, *TabuCol* [Hertz et Werra, 1987] et *PartialCol* [Blöchliger et Zufferey, 2008], qui ont été proposées de longue date dans la littérature. Pour le WVCP, ces méthodes sont beaucoup plus récentes : *AFISA* [Sun *et al.*, 2018], *RedLS* [Wang *et al.*, 2020], *ILS-TS* [Nogueira *et al.*, 2021] et *TabuWeight* [Grelier *et al.*, 2022]. Nous avons vu dans les chapitres précédents qu'aucune de ces méthodes ne domine vraiment les autres pour toutes les instances de référence, et qu'il y a donc un intérêt à choisir une recherche locale adaptée pour résoudre chaque type d'instance.

Cependant, même en admettant qu'une procédure de recherche locale bien adaptée soit

choisie pour résoudre une instance spécifique, il est souvent difficile pour celle-ci d'atteindre une solution de très bonne qualité par elle-même pour les instances de grands graphes qui comportent de nombreux optima locaux.

Pour échapper à ces optima locaux, nous avons vu dans la Section 2.2.2 du Chapitre 2 que les algorithmes de recherche locale intègrent généralement des mécanismes dédiés tels qu'une liste tabou [Hertz et Werra, 1987, Blöchliger et Zufferey, 2008] ou des stratégies de perturbation [Nogueira *et al.*, 2021]. Cependant, pour les cas très difficiles de coloration de graphe, l'approche de recherche locale à trajectoire unique n'est souvent pas assez puissante pour localiser des solutions de très haute qualité, principalement en raison de sa capacité de diversification limitée. Pour surmonter cette difficulté, des algorithmes hybrides ont été proposés, en particulier en s'appuyant sur le cadre des algorithmes mémétiques qui comprend une population d'individus et qui combine des recherches locales avec des croisements (cf. Section 2.2.3 du Chapitre 2). Ces algorithmes hybrides combinent les avantages de la recherche locale pour l'intensification avec une population de solutions de haute qualité offrant des possibilités de diversification.

Ces différents algorithmes mémétiques ont principalement été utilisés à ce jour pour résoudre le GCP. L'algorithme HEA (*Hybrid Evolutionary Algorithm*) [Galinier et Hao, 1999] a introduit l'opérateur de croisement très performant *GPX* et utilise une recherche locale tabou inspirée de *TabuCol*. Evo-Div [Porumbel *et al.*, 2010a] et MACOL [Lü et Hao, 2010] utilisent tous deux des stratégies de croisement avec plusieurs parents et une gestion de la distance entre les solutions. Plus récemment l'algorithme HEAD (*HEA in Duet*) [Moalic et Gondran, 2018] propose l'utilisation d'uniquement deux individus dans la population et un système de réintégration d'individus de bonne qualité qui avaient été trouvés plus tôt lors de la recherche. HEAD utilise aussi le croisement *GPX* et une amélioration de l'algorithme original *TabuCol*. Cet algorithme est actuellement un des solveurs les plus efficaces pour le GCP.

Pour le WVCP, il n'existe à notre connaissance qu'un seul algorithme mémétique à ce jour dans la littérature : DLMCOL que nous avons récemment proposé dans [Goudet *et al.*, 2022] (non présenté dans cette thèse). Sa particularité est d'utiliser une très grande population d'individus (plus de 20000), dont les recherches locales sont calculées en parallèle sur un GPU, et d'utiliser un réseau de neurones pour prédire quel appariement entre les individus de la population permettra d'effectuer un croisement efficace, afin d'obtenir les meilleurs résultats possibles avec la recherche locale *AFISA*.

Dans HEAD [Moalic et Gondran, 2018] appliqué pour le GCP, les auteurs ont remarqué que la qualité d'une solution enfant (mesurée en termes de *fitness*) générée avec un croisement *GPX* à partir de deux parents dépend fortement de la distance entre ces deux parents (au sens de

la distance de partitionnement définie dans la Section 1.4.2 du Chapitre 2). Plus précisément, les auteurs ont montré expérimentalement qu'il existe généralement une corrélation négative entre la distance entre les deux parents et la qualité de la solution enfant. Ceci se comprend intuitivement par le fait que la recombinaison de solutions de colorations très différentes (et donc "incompatibles") a de forte chance de produire une solution enfant qui ne contient plus les grands ensembles de sommets indépendants qui étaient contenus dans l'un ou l'autre de ses parents, et donc est de mauvaise qualité.

En partant de ce constat, les auteurs ont proposé d'utiliser des variantes de l'opérateur de croisement *GPX*, notamment des crossovers asymétriques plus *conservatifs*, dans le sens où l'enfant reçoit une plus grande partie des groupes de couleurs d'un seul des deux parents et moins du second parent, ce qui permet de diversifier la recherche tout en conservant plus de *structure* dans la solution descendante.

Les auteurs ont montré que ces variantes de l'opérateur de croisement GPX pouvaient mener à de meilleurs résultats pour certains types d'instances, notamment pour des instances de graphes géométriques (comme les instances DSJR) pour lesquels les minima locaux de bonne qualité (parents) sont très distants, à la différence des graphes aléatoires (comme les instances DSJC) pour lesquels on retrouve paradoxalement plus de similitudes dans les bonnes solutions rencontrées et donc des distances réduites entre les individus de la population.

Dans le cadre d'un algorithme mémétique, il semble donc pertinent de choisir la bonne recherche locale, mais aussi le bon opérateur de croisement pour réaliser une recherche efficace.

Dans ce chapitre, les contributions proposées sont les suivantes :

- de nouveaux algorithmes mémétiques reprenant le *framework* HEAD, proposé par [Moalic et Gondran, 2018], couplé à de nouvelles procédures de recherches locales, sont mis en œuvre pour le GCP et le WVCP. De tels algorithmes mémétiques sont nouveaux notamment pour le WVCP pour lequel il existe à ce jour très peu d'algorithmes évolutionnaires pour le résoudre.
- nous intégrons une nouvelle procédure de recherche, nommée *TabuEdge*. Inspirée de *RedLS* [Wang *et al.*, 2020] pour le WVCP, elle s'est montrée très efficace pour le GCP surtout pour des instances que les algorithmes *TabuCol* et *PartialCol* ont du mal à résoudre.
- Enfin, nous proposons d'étudier les capacités des hyperheuristiques à choisir des procédures de recherche locale, mais aussi de croisement, adaptées à la résolution d'une instance donnée au cours de la recherche.

Ce chapitre présentera d'abord le *framework* général de l'algorithme avec ses différentes

étapes. Nous présenterons en parallèle les différentes subtilités en fonction du problème étudié. Les différents opérateurs de croisement et recherche locale seront introduits ainsi qu'un rappel sur les différents critères de sélection vu dans le chapitre précédent. Enfin, nous étudierons les impacts de nos algorithmes sur les instances de référence de l'état de l'art.

8.2 Adaptive HEAD

Nous présentons dans cette section le framework général d'algorithme mémétique adaptatif mis au point pour les deux problèmes GCP et WVCP, ainsi que les différents opérateurs utilisés.

8.2.1 Framework Général

L'architecture générale de l'algorithme proposé, AHEAD (*Adaptive HEAD*), est décrite dans la Figure 8.1 et l'Algorithme 10. Elle reprend celle de HEAD [Moalic et Gondran, 2018] avec, en plus, des phases de sélection d'opérateurs.

Nous avons choisi comme point de départ le framework HEAD, car il obtient actuellement les meilleurs résultats de l'état de l'art pour le problème de coloration standard. De plus, il présente l'avantage d'être très simple, car il y a seulement deux individus dans la population, ce qui simplifie au maximum les phases d'appariement et d'insertion qui apparaissent typiquement dans la plupart des autres algorithmes de la littérature [Galinier et Hao, 1999, Lü et Hao, 2010, Porumbel *et al.*, 2010a, Goudet *et al.*, 2022].

Une version avec 10 individus et un fonctionnement plus proche des algorithmes mémétiques proposés par [Galinier et Hao, 1999, Lü et Hao, 2010, Porumbel *et al.*, 2010a] a aussi été testée. Nous ne la présentons pas ici, car elle obtenait de moins bons résultats.

L'algorithme que nous proposons dans ce travail prend en entrée un graphe pondéré $G = (V, E, w)$, un ensemble d'opérateurs de recherche locale $\mathcal{O}^l = \{o^l_1, \ldots, o^l_m\}$, un ensemble d'opérateurs de croisement $\mathcal{O}^x = \{o^x_1, \ldots, o^x_n\}$ et une stratégie de sélection de haut niveau π_θ caractérisée par un vecteur de paramètre θ. Cette stratégie π_θ permettra de sélectionner deux paires d'opérateurs $< o^l_i, o^x_j >$ avec $o^l \in \mathcal{O}^l$ et $o^x \in \mathcal{O}^x$ à appliquer à chaque génération de façon à générer deux nouveaux individus qui remplaceront leurs parents dans la population pour la génération suivante. Dans le cas du GCP, cet algorithme prend en entrée aussi le nombre maximum k de couleurs qu'il est possible d'utiliser.

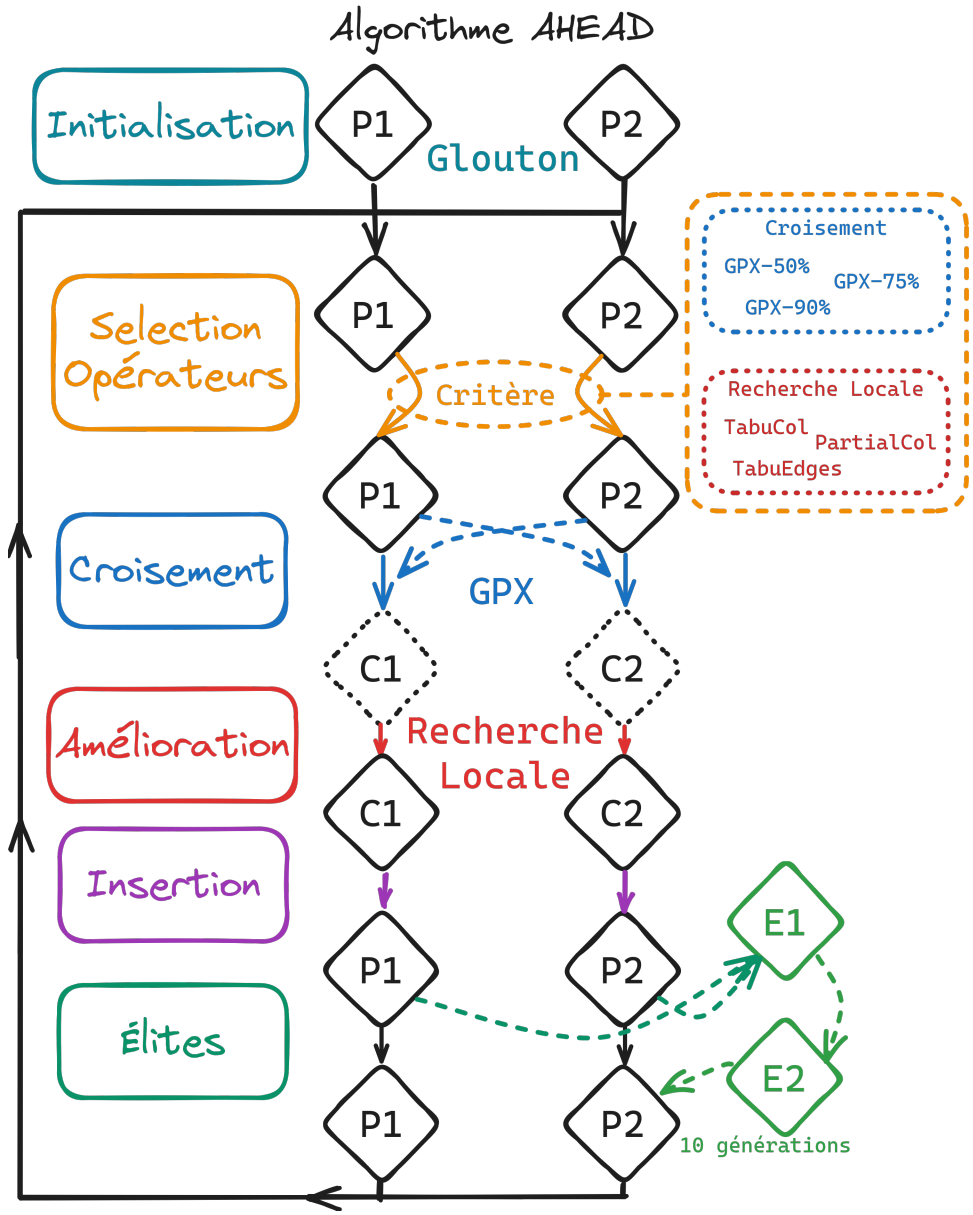


FIGURE 8.1 – Architecture générale de l'algorithme mémétique adaptatif, Adaptive HEAD (AHEAD). Après initialisation aléatoire des deux individus de la population, deux opérateurs de croisement et deux opérateurs de recherche locale sont sélectionnés à chaque génération. Les deux individus parents $P1$ et $P2$ sont croisés avec les deux opérateurs de croisement sélectionnés pour générer deux nouveaux individus $C1$ et $C2$. Ces deux nouveaux individus sont ensuite améliorés par les deux opérateurs de recherche locale sélectionnés. Les deux individus améliorés sont ensuite insérés dans la population pour la génération suivante. Enfin, le meilleur individu de la génération courante est sauvegardé pour être réintroduit dans la population 10 à 20 générations plus tard en tant qu'élite.

La population est tout d'abord initialisée aléatoirement avec deux solutions de coloration S_1 et S_2, ainsi que deux solutions "élites", $elite_1$ et $elite_2$ dans l'espace de recherche Ω. Ces deux solutions élites permettent de réintroduire de la diversité dans le duo de solution S_1 et S_2 comme dans l'algorithme HEAD original [Moalic et Gondran, 2018].

Ensuite, à chaque génération, l'algorithme alterne cinq étapes jusqu'à ce que la condition d'arrêt soit rencontrée :

1. Une phase de sélection de deux paires d'opérateurs <croisement, recherche locale> à appliquer : $< o_1^x, o_1^l >$ et $< o_2^x, o_2^l >$. Cette sélection, détaillée dans la Section 8.2.2, se fait selon la fonction π_θ qui prend en entrée les deux individus de la population, avec S_1 en premier parent pour la sélection de $< o_1^x, o_1^l >$, puis S_2 en premier parent pour la sélection de $< o_2^x, o_2^l >$.
2. Une phase de croisement des deux parents S_1 et S_2 durant laquelle les deux croisements qui viennent d'être sélectionnés, o_1^x, respectivement o_2^x, sont appliqués avec S_1 en premier parent, respectivement S_2 en deuxième parent. Chaque phase de croisement permet de générer deux nouveaux individus C_1 et C_2 (cf. Section 8.2.3).
3. Une phase d'intensification, durant laquelle les deux individus de la population C_1, et C_2 sont améliorés respectivement par les procédures de recherche locale o_1^l et o_2^l pendant un temps limité de T_{RL} secondes (Section 8.2.4). Ces deux procédures de recherches locales sont indépendantes et sont lancées en parallèle sur deux CPU différents. Chacune de ces procédures de recherche locale renvoie le meilleur individu obtenu pendant la recherche. Les deux individus ainsi obtenus, C_1' et C_2' sont automatiquement conservés et deviennent les nouveaux parents pour la phase de croisement de la génération suivante.
4. Une phase de stockage d'exemples d'apprentissages constitués à partir des opérateurs sélectionnés et des résultats qu'ils ont permis d'obtenir, suivie d'une phase d'apprentissage de la politique de sélection π_θ à partir des exemples collectés lors des dernières générations. Cette phase est décrite dans la Section 8.2.5.
5. Enfin, comme dans l'algorithme HEAD original [Moalic et Gondran, 2018], pour réintroduire continuellement de la diversité dans la population et éviter d'être bloqué dans des minima locaux, une génération de l'algorithme se termine par une phase de stockage du meilleur individu du cycle (individu *élite*) qui sera réintroduit dans la population deux cycles plus tard (Section 8.2.6).

Algorithme 10 Algorithme AHEAD

Entrée: $G = (V, E, w)$ un graphe pondéré ; $\mathcal{O}^l$ un ensemble de d'opérateurs de recherche locale ; $\mathcal{O}^x$ un ensemble de d'opérateurs de croisement ; π_θ une stratégie de sélection ; $Iter_{cycle} = 10$, le nombre de générations avant de réintroduire les solutions élites ; T_{RL} le temps passé dans chaque recherche locale.

Sortie: S^* la meilleure solution trouvée

Initialisation aléatoire de la population :
S_1, S_2, $elite_1$, $elite_2$, $S^* \leftarrow$ init()
$D = \emptyset$.
$generation \leftarrow 0$
tant que condition d'arrêt non atteinte **faire**
    # Sélection de paires d'opérateurs <croisement,recherche locale> (Section 8.2.2) :
    $< o_1^x, o_1^l >= \pi_\theta(S_1, S_2)$
    $< o_2^x, o_2^l >= \pi_\theta(S_2, S_1)$
    # Application des opérateurs de croisements sélectionnés (Section 8.2.3) :
    $C_1 \leftarrow o_1^x(S_1, S_2)$
    $C_2 \leftarrow o_2^x(S_2, S_1)$
    # Application des procédures de recherche locale sélectionnées (Section 8.2.4) :
    $C'_1 \leftarrow o_1^l(C_1, T_{RL})$
    $C'_2 \leftarrow o_2^l(C_2, T_{RL})$
    # Insertion automatique des nouveaux individus pour remplacer les parents :
    $S_1 \leftarrow C'_1$
    $S_2 \leftarrow C'_2$
    # Stockage de deux nouveaux exemples d'apprentissage dans la base de données :
    $D \leftarrow D \cup (o_1^x, o_1^l, C_1, -f(C'_1)) \cup (o_2^x, o_2^l, C_2, -f(C'_2))$
    # Mise à jour de la stratégie de sélection adaptative (Section 8.2.5) :
    $\theta \leftarrow learning(D, \theta)$
    # Sauvegarde du meilleur individu du cycle courant :
    $elite_1 \leftarrow saveBest(S_1, S_2, elite_1)$
    # Sauvegarde de la meilleure solution obtenue depuis le début :
    $S^* \leftarrow saveBest(elite_1, best)$
    # Ré-insertion d'un individu élite à la fin de chaque cycle (Section 8.2.6) :
    si $generation \% Iter_{cycle} = 0$ **alors**
        $S_1 \leftarrow elite_2$
        $elite_2 \leftarrow elite_1$
    $generation++$
retourne S^*

8.2.2 Sélection Automatique d'Opérateurs

Pour sélectionner les opérateurs de croisement et de recherche locale dans l'algorithme mémétique, nous reprenons le même framework d'apprentissage que celui présenté dans le Chapitre 7, avec les différences suivantes :

- Au lieu de sélectionner un seul opérateur de recherche locale comme dans le Chapitre 7, cette fois-ci π_θ sélectionne un couple d'opérateurs $< o^x, o^l > \in \mathcal{O}^x \times \mathcal{O}^l$. Il y a donc $|\mathcal{O}^x| \times |\mathcal{O}^l|$ différents couples <croisement,recherche locale> possibles. Nous considérons que la sélection se fait comme si chaque couple était un méta-opérateur indépendant. Cette sélection se fait donc sans considérer d'interaction particulière entre deux couples différents, mais qui contiendraient par exemple la même recherche locale.
- Dans le cas général, la fonction π_θ ne prend plus en entrée une solution brute S comme dans le Chapitre 7, mais un couple de parents. Par exemple, pour (S_1, S_2), π_θ choisit un couple d'opérateurs (o^x, o^l), avec un opérateur de croisement o^x à calculer avec S_1 en premier parent et S_2 en second parent. o^x donnera un enfant C_1 qui sera amélioré par la procédure de recherche locale o^l pour obtenir un nouvel individu C'_1.
- Les scores de récompense associés à chaque choix de couple d'opérateurs (o^x_1, o^l_1) et (o^x_2, o^l_2), sont les scores obtenus après les recherches locales, à savoir $r = -f(C'_1)$, et $r = -f(C'_2)$, avec f la fonction de fitness du problème GCP ou WVCP.

Sélecteur avec Réseau de Neurones

Pour la sélection avec le réseau de neurones, nous reprenons la même architecture de réseau de neurones que celle utilisée dans le Chapitre 7. Elle prend en entrée une solution brute et donne en sortie une estimation des résultats obtenus après l'application de chaque opérateur de recherche locale.

Nous utilisons ce réseau de neurones g_θ pour sélectionner un couple d'opérateurs $(o^x, o^l) \in |\mathcal{O}^x| \times |\mathcal{O}^l|$, à partir d'un couple de parents (S_1, S_2) de la façon suivante :

1. pour chaque opérateur de croisement $o^x \in \mathcal{O}^x$, un individu $C^x = o^x(S_1, S_2)$ est généré.
2. chaque solution de coloration brute C^x, obtenue après croisement, est passée en entrée du réseau de neurones g_θ, pour obtenir un vecteur de $|\mathcal{O}^l|$ valeurs dans $\mathbb{R}$ correspondant à la récompense estimée après l'application de chaque recherche locale $o^l \in \mathcal{O}$ à C^x.
3. le couple (o^x, o^l) correspondant à la valeur la plus élevée en sortie du réseau de neurones (pour les $|\mathcal{O}^x|$ évaluations données par g_θ) est sélectionné.

Dans ce travail, comme dans le Chapitre 7, nous utilisons un réseau de neurones avec deux couches cachées de taille $h1 = |V|$ et $h2 = \frac{|V|+|\mathcal{O}|}{2}$, et une fonction d'activation non linéaire définie comme $\mu(x) = \max(0.2 \times x, x)$ (LeakyReLU).

Sélecteurs Classiques Basés sur la Fitness

Comme dans le Chapitre 8, nous comparons le sélecteur avec réseau de neurones décrit ci-dessus avec cinq sélecteurs, qui ne prennent pas en entrée le couple de parents (S_1, S_2) pour faire leur choix. Ces cinq sélecteurs sont deux critères basés sur un biais proportionnel (*Roulette* et *Pursuit*), un critère basé sur la méthode du bandit à un bras (*UCB*), un aléatoire (*Random*) et un supprimant un à un les opérateurs les moins performants pour n'en garder qu'un (*Deleter*).

8.2.3 Application des Opérateurs de Croisement

Lors de cette étape les solutions parents S_1 et S_2 sont fusionnées pour créer deux solutions enfant $C_1 = o_1^x(S_1, S_2)$ et $C_2 = o_2^x(S_2, S_1)$ à partir des opérateurs o_1^x et o_2^x choisis lors de la phase de sélection détaillée dans la section précédente.

Notons que les opérateurs de croisement que nous considérons sont asymétriques, donc appliquer un même opérateur de croisement avec le couple de parents (S_1, S_2) en entrée ne produit pas la même solution enfant que si le couple (S_2, S_1) était passé en entrée.

La Figure 8.2 présente l'application de l'opérateur de croisement *GPX* sur deux solutions parents P_1 et P_2 pour créer une solution enfant $C1$. À chaque étape de l'algorithme, un parent est sélectionné pour donner sa couleur contenant le plus de sommets. Les sommets de cette couleur sont alors ajoutés à la solution enfant et les sommets sont retirés des deux parents. Pour finir, les sommets restants sont colorés dans les couleurs minimisant le nombre de conflits pour le GCP, ou dans la première couleur disponible pour le WVCP, avec possibilité d'ouvrir de nouvelles couleurs.

Dans ce travail, nous considérons les opérateurs de croisement suivants :

1. Le premier opérateur est l'opérateur très populaire *GPX* (*Greedy Partition Crossover*) présenté dans la Section 2.2.3 de l'état de l'art et utilisé dans l'algorithme HEAD original [Moalic et Gondran, 2018]. Il consiste à prendre alternativement le plus grand groupe de couleurs de chaque parent - dans le cas du GCP - jusqu'à atteindre le nombre de couleurs recherché, les sommets non colorés seront alors ajoutés dans des couleurs minimisant le nombre de conflits, et - dans le cas du WVCP - jusqu'au nombre de cou-

leurs maximum entre les deux parents, les sommets restants seront colorés dans la première couleur disponible avec la possibilité d'ajouter de nouvelles couleurs.

2. Une variante du crossover *GPX*, notée *GPX*-75% pour lequel on prend alternativement 3 groupes du premier parent, puis 1 groupe du second parent.
3. Une variante du crossover *GPX*, notée *GPX*-90% pour lequel on prend alternativement 9 groupes du premier parent, puis 1 groupe du second parent.

Ces deux dernières variantes du crossover *GPX* sont beaucoup plus conservatrices, car une plus grande partie du premier parent est transmise à l'enfant. Ce qui permet généralement, en pratique, de conserver un enfant de meilleure fitness (mais moins diversifié) même quand les deux parents sont très distants dans l'espace de recherche.

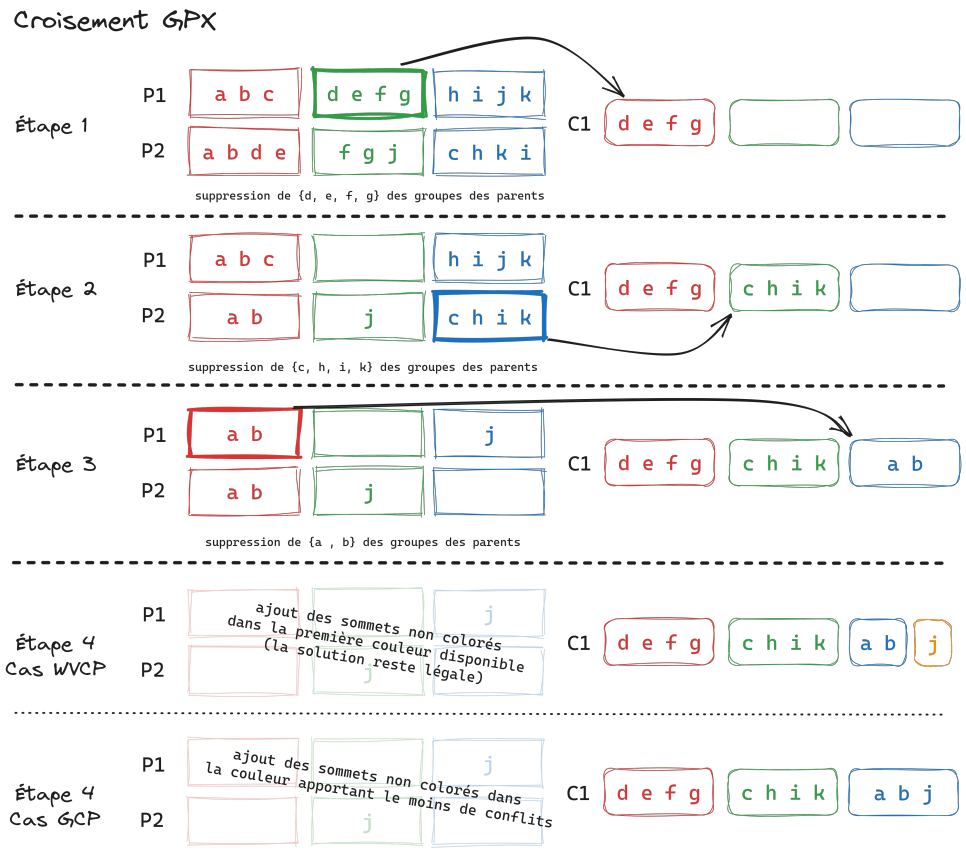


FIGURE 8.2 – Application de l'opérateur *GPX*. L'étape 4 présente les deux différents cas possibles pour le GCP et le WVCP. Pour le WVCP, les sommets restants sont colorés dans la première couleur disponible avec possibilité d'ouvrir de nouvelles couleurs. Pour le GCP, il n'est pas possible d'ouvrir de nouvelles couleurs, donc les sommets restants sont colorés dans les couleurs minimisant le nombre de conflits.

Cette phase de croisement permet de générer deux individus enfants C_1 et C_2 qui seront améliorés dans la phase de recherche locale détaillée dans la sous-section suivante.

8.2.4 Recherches Locales

Lors de cette étape, les nouveaux individus C_1 et C_2 sont améliorés avec l'aide des recherches locales o_1^l et o_2^l sélectionnées. Ces deux recherches locales indépendantes sont lancées en parallèle sur deux CPU différents. Leur temps d'exécution est limité à $T_{RL} = 0.04|V|$ pour le WVCP et $T_{RL} = 0.001|V|$ pour le GCP, avec $|V|$ le nombre de sommets du graphe.

Nous disposons de plusieurs opérateurs de recherches locales qui sont adaptés à chaque problème particulier, GCP ou WVCP.

Opérateurs de Recherche Locale pour le GCP

Pour le GCP, nous utilisons ces deux recherches locales populaires dans l'état de l'art :

- *TabuCol* [Hertz et Werra, 1987] recherche explorant l'espace illégal des solutions avec un opérateur de voisinage *one-move*
- *PartialCol* [Blöchliger et Zufferey, 2008] recherche explorant l'espace partiellement légal des solutions avec un opérateur de voisinage *grenade*

Nous proposons aussi un nouvel opérateur pour le GCP :

TabuEdges (Algorithme 11) inspiré de *RedLS* de [Wang *et al.*, 2020]. Cet opérateur explore l'espace illégal des solutions, mais demande de partir d'une solution légale sans limite sur le nombre de couleurs à utiliser pour obtenir de meilleurs résultats. Lorsque la solution est légale, les sommets d'un groupe de couleurs sont déplacés vers la couleur apportant le moins de conflits. Tant qu'un *one-move* améliorant non tabou est possible, alors il est appliqué jusqu'à atteindre un minimum local. Les sommets sont tabou jusqu'à ce qu'une solution légale soit trouvée ou jusqu'à ce qu'un voisin du sommet devienne tabou. Lorsque la solution est dans un minimum local, si aucun déplacement non tabou ne permet de résoudre un conflit en ajoutant une couleur, alors le poids de toutes les arêtes en conflit est incrémenté et un sommet en conflit est déplacé dans une couleur minimisant le nombre de conflits. [Morris, 1993] proposait déjà un système similaire pour échapper à des minima locaux. Les poids sur les arêtes sont utilisés pour calculer la pénalité et les différences de score lors de la recherche. Ces poids permettent de transmettre de l'apprentissage lors de la recherche. Lorsque les poids sont élevés pour une arête, cela signifie pour l'algorithme qu'il est difficile de résoudre le conflit si les deux sommets

Algorithme 11 *TabuEdges*

Entrée: $G = (V, E)$ un graphe, S une solution légale
Sortie: S^* la meilleure solution trouvée

$S^* \leftarrow S$
$tabou \leftarrow$ liste de $false$ de taille $|V|$
pénalité sur les arcs en conflit initialisée à 1
tant que condition d'arrêt non atteinte **faire**
    si S est légale **alors**
        $S^* \leftarrow S$
        reset $tabou$ à false pour tous les sommets
        Déplace tous les sommets d'une couleur à une autre en minimisant les conflits
        si S est légale **alors**
            continue
    si $\exists N_{o-m}(S)$ améliorant la pénalité et non tabou **alors**
        appliquer le déplacement $N_{o-m}(S)$
        le sommet déplacé devient tabou, ses voisins sont libérés de la liste tabou
    sinon
        incrémenter la pénalité des arcs en conflits
        déplacer un sommet en conflit aléatoire en minimisant les conflits
        le sommet déplacé devient tabou
retourne S^*

se retrouvent dans la même couleur. C'est pourquoi il est préférable de partir d'une solution légale, car lorsque la recherche a beaucoup diminué le nombre de couleurs, sur le chemin elle a aussi appris quelles paires de sommets ne pas mettre dans la même couleur, alors qu'en partant d'une solution avec beaucoup de sommets en conflit, aucun apprentissage sur ce point n'est fait.

Dans le cas du GCP, la recherche locale prend en entrée une solution illégale pour un nombre de couleurs et doit retourner une solution légale ou illégale pour ce même nombre de couleurs. Si la recherche locale travaille dans l'espace partiellement légal, comme *PartialCol* celle-ci à la charge de transformer les solutions d'un espace à l'autre.

Opérateurs de Recherche Locale pour le WVCP

Pour le WVCP, la recherche locale prend en entrée une solution légale et retourne la meilleure solution légale trouvée.

Les opérateurs de recherche locale possibles sont les mêmes que dans le chapitre précédent, *AFISA* [Sun *et al.*, 2018], *TabuWeight* [Grelier *et al.*, 2022], *RedLS* [Wang *et al.*, 2020] et *ILS-TS* [Nogueira *et al.*, 2021].

Cependant, comme remarqué dans le chapitre précédent, *AFISA* n'était jamais sélectionné. Pour les instances pour lesquelles *TabuWeight* était meilleur face à *ILS-TS*, il restait moins bon face à *RedLS* et inversement sur les instances où il était meilleur face à *ILS-TS*, il restait moins bon face à *RedLS*. Nous avons donc décidé de ne garder que *RedLS* et *ILS-TS* pour cette étude. À noter que cette décision avantage le sélecteur *Random* qui a plus de chance de sélectionner un opérateur performant.

8.2.5 Mise à Jour de la Stratégie d'Apprentissage

Toutes les nb itérations, la politique π_θ est entraînée sur la base de données D et l'ensemble de ses paramètres θ est mis à jour.

La base de données D est modélisée comme une file d'attente de taille N correspondant aux N derniers exemples obtenus au cours des itérations précédentes. Cette file de taille limitée permet de mieux s'adapter aux variations potentielles des résultats des opérateurs, au cas où certains opérateurs seraient meilleurs au début de la recherche qu'à la fin.

Pour le réseau de neurones, pendant la phase d'entraînement, chaque exemple d'apprentissage (o^x, o^l, C, r) de la base de données D est converti en un exemple d'apprentissage supervisé (X, y), avec X une matrice d'entrée de taille $k \times |V|$ correspondant à l'ensemble des k vecteurs $C = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, et y est un vecteur réel de taille $|\mathcal{O}^l|$ (nombre d'opérateurs de recherche locale), de sorte que y est initialisé avec $g_\theta(C)$, le vecteur de sortie du réseau de neurones prenant en entrée C, puis sa valeur $y[o^l]$ pour l'opérateur choisi o^l est remplacée par la récompense attendue r : $y[o] = r$.

Toutes les $nb = 10$ générations, ce réseau de neurones est entraîné avec l'erreur quadratique moyenne sur les $|\mathcal{O}|$ sorties (apprentissage supervisé) sur l'ensemble de données D pendant 15 époques avec l'optimiseur Adam [Kingma et Ba, 2014] et un taux d'apprentissage de 0.001.

8.2.6 Insertion et Solutions Élites

Le principe de l'insertion est le même que pour HEAD de [Moalic et Gondran, 2018] : les enfants remplacent systématiquement les parents et il y a une phase de gestion des élites. Les élites sont des solutions rencontrées lors de générations antérieures qui sont ajoutées à la population après un certain nombre d'itérations pour apporter de la diversité et éviter d'obtenir des solutions trop proches au fil des itérations. Ainsi toutes les $Iter_{cycle} = 10$ générations, le meilleur individu rencontré des 10 à 20 générations précédentes est réinséré dans la population.

8.3 Expérimentations

Cette section présente les expériences menées pour évaluer l'impact des différents opérateurs de croisement et de recherche locale, ainsi que des différents critères de sélection, dans le cadre du framework AHEAD d'algorithme mémétique.

8.3.1 Paramètres Expérimentaux

Pour le WVCP, nous considérons les 48 instances de référence les plus difficiles de la littérature en version réduite. Pour les 5 instances dont le score optimal est connu, les différentes méthodes cherchent à atteindre ce score et s'arrêtent une fois qu'il est trouvé. Pour le GCP, 31 instances parmi les plus difficiles ont été retenues en version réduite. Pour ces instances, 20 exécutions indépendantes par méthode sont réalisées pendant une heure pour chaque nombre de couleurs k fixé.

Le temps passé dans la recherche locale lors de chaque génération est de $0.04*|V|$ dans le cas du WVCP et $0.001*|V|$ dans le cas du GCP. Ces valeurs ont été choisies suite à des tests sur une plage de valeurs pour chaque problème qui ne seront pas présentées ici. À noter que les résultats obtenus par l'algorithme HEAD dans la version d'origine ne correspondent pas forcément aux résultats obtenus par notre version de HEAD avec les mêmes opérateurs *TabuCol* et *GPX*, car contrairement à l'article d'origine, nous n'avons pas effectué de *fine-tuning* spécifique du temps passé dans la recherche locale pour chaque instance donnée (contrairement à ce qui avait été fait dans [Moalic et Gondran, 2018]).

Par ailleurs, le temps d'entraînement du réseau de neurones n'est pas optimisé (il pourrait être moins fréquent). Toutefois, de façon à gommer l'impact du temps d'entraînement, pour les versions avec hyperheuristiques, les méthodes disposent d'autant d'itérations dans l'algorithme mémétique que les versions sans hyperheuristiques sur une heure de calcul.

Les expériences sont réalisées sur un ordinateur équipé d'un processeur Intel Xeon ES 2630, 2,66 GHz. Tous les algorithmes sont codés en C++, compilés et optimisés avec le compilateur g++ 12.1. Le code source sera disponible lors de la publication du travail dans le futur. Pour l'implémentation du réseau de neurones, la librairie C++ de Pytorch 1.13 a été utilisée.

Dans ce qui suit, lorsqu'une méthode est dite meilleure qu'une autre sur une instance donnée, cela signifie que la différence entre les scores moyens calculés sur 20 exécutions est en faveur de la première méthode et que cette différence est significative (test de rang signé de Wilcoxon non paramétrique avec une valeur p $\leq 0,001$).

8.3.2 Résultats Expérimentaux pour le WVCP

Dans cette section nous analysons tout d'abord les résultats obtenus pour le WVCP, puis nous étudions les sélections des différents opérateurs dans AHEAD.

Comparaison des Différentes Méthodes

Nous proposons tout d'abord une analyse comparative de l'algorithme HEAD en combinaison avec les différentes recherches locales, mais sans sélection automatique des opérateurs. Dans ces versions, le croisement utilisé est toujours le croisement *GPX*. Nous analysons notamment si une combinaison de HEAD avec une recherche locale RL donnée (version nommée HEAD + RL) donne de meilleurs résultats que cette recherche locale lancée seule avec le même temps total d'exécution (soit 2h, car HEAD est lancé pendant 1h sur 2 CPU). Nous avons aussi comparé HEAD avec l'algorithme MCTS présenté dans le chapitre précédent. La version du MCTS choisie pour la comparaison est celle avec l'hyperheuristique de sélection *UCB* et un choix *online* entre les opérateurs de recherche locale *RedLS* et *ILS-TS*. Nous analysons aussi l'impact d'une stratégie automatique de sélection des opérateurs dans AHEAD, en comparant les différentes stratégies de sélection proposées dans la Section 8.2.2. Ces versions sont nommées "AHEAD + le nom de la stratégie de sélection".

Le Tableau 8.2 compare chaque méthode deux à deux pour le WVCP et compte le nombre d'instances où la méthode en ligne est significativement meilleure que la méthode en colonne.

Lorsque nous analysons l'impact de HEAD en combinaison avec des recherches locales, nous pouvons constater que HEAD améliore par exemple les résultats de *RedLS* sur 28 instances, mais la recherche locale *RedLS* seule reste meilleure sur 10 instances. Les instances où *RedLS* reste meilleur sont les C2000, DSJC1000.5/9, DSJC500.9, flat, latin_square et wap03a, soit les plus grandes instances qui demandent beaucoup d'intensification, à l'inverse, les instances plus petites ou peu denses sont mieux résolues par HEAD : le450, queen, plusieurs wap et les DSJC de faible densité. Pour *ILS-TS*, l'impact de HEAD est positif sur les flat et DSJC1000.1/5, instances pour lesquelles *ILS-TS* est moins performant à l'inverse de presque toutes les instances de plus petite taille (à l'exception de C2000.9) où il garde l'avantage seul.

Ces analyses sont présentées dans le Tableau 8.1, qui résume les meilleurs scores et les moyennes obtenus, ainsi que le temps pour atteindre le meilleur score. Les valeurs en gras correspondent au meilleur score sur la ligne. Avec plus de temps d'exécution (4h), nous avons trouvé trois nouveaux meilleurs scores pour le WVCP. Ainsi AHEAD+*Deleter* trouve un score de 211 pour le450_15a, 215 pour le450_15b et 214 pour queen14_14.

instance	BKS	MCTS+*UCB*			*RedLS*			*ILS-TS*			HEAD+*RedLS*			HEAD+*ILS-TS*			AHEAD+*Random*			AHEAD+*Roulette*			AHEAD+*Deleter*			AHEAD+*UCB*			AHEAD+*Pursuit*			AHEAD+*NN*		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	2337	2349.6	3567	**2144**	**2183.3**	5499	2237	2265.9	3801	2254	2280.9	810	2256	2271.4	2592	2213	2246.4	1944	2231	2254.2	3726	2228	2251.8	972	2221	2248.8	2430	2230	2250.1	3564	2227	2259.2	892
C2000.9	5477	6107	6139.1	615	**5477**	**5500.6**	6834	5873	5944	5963	5754	5780.9	3407	6004	6038.4	3569	5782	5815.6	1298	5783	5817.8	3731	5765	5805.3	1785	5786	5819.9	3731	5793	5818.1	3731	5782	5814	1829
DSJC1000.1	300	303	304.4	3700	305	306.9	2611	304	306.8	4747	305	306.5	2214	303	304.8	2255	303	304.8	3608	302	304.6	1804	**300**	**304**	2788	302	304.2	3198	302	304.4	3034	303	305.2	948
DSJC1000.5	1185	1242	1246.2	5439	**1198**	**1209.8**	2418	1245	1267.8	2.50	1226	1234	533	1238	1256	574	1226	1233.7	2022	1221	1233.2	3444	1224	1231.8	410	1225	1233.3	492	1214	1232.2	2952	1227	1232.7	1076
DSJC1000.9	2836	2979	2991.7	3717	**2836**	**2852.9**	6830	3022	3060.2	6600	2925	2937.3	2215	3066	3081.8	902	2929	2938.6	1969	2909	2939.8	1066	2918	2935	2379	2922	2934.9	3445	2915	2934.8	1476	2930	2939.9	924
DSJC500.1	184	186	187.9	4928	186	201.7	6451	187	188.2	4140	187	187.4	1788	186	188.5	2079	**185**	186.5	1904	**185**	186.8	2772	**185**	**186.4**	1932	**185**	186.5	2856	186	187.1	882	186	187.1	898
DSJC500.5	685	712	717.8	1870	**706**	**714.6**	2992	719	730.4	5350	709	716.2	2730	720	728.1	1764	709	715.7	1827	709	715.6	2814	711	716.2	2142	710	716.1	756	708	**714.6**	1974	712	716.5	888
DSJC500.9	1662	1687	1693	847	**1668**	**1672.8**	6407	1717	1736	5663	1678	1686	3024	1730	1747.2	2982	1679	1687.2	2268	1682	1686.5	2667	1680	1685.7	1491	1681	1686.7	1260	1681	1687.4	462	1679	1687.5	1329
DSJC250.1	127	**127**	**127.3**	2224	130	131.6	1	**127**	127.6	3607	**127**	127.7	1789	**127**	127.7	1829	**127**	**127.3**	1744	**127**	127.4	1970	**127**	127.5	1796	**127**	127.5	1284	**127**	127.7	1041	**127**	127.5	837
DSJC250.5	392	395	397.4	870	398	401.2	97	**392**	**395.2**	3916	396	397.9	2076	395	398.1	2552	394	396.9	418	395	396.8	2948	395	397.4	1730	393	397.2	2926	394	397.4	2816	393	397	805
DSJC250.9	934*	935	935.8	3186	**934**	**935.5**	757	**934**	939.6	5832	**934**	**935.5**	2549	**934**	943.2	1364	**934**	935.6	2352	**934**	**935.5**	1262	**934**	935.8	480	**934**	**935.5**	2603	**934**	935.6	1062	**934**	935.6	323
DSJC125.5gb	240	**240**		118	245	260.5	0	**240**		107	**240**	240.9	1088	**240**		100	**240**		234	**240**		161	**240**		195	**240**		245	**240**		131	**240**		160
DSJC125.5g	71	**71**		83	72		244	**71**		126	**71**	71.2	1001	**71**		61	**71**		114	**71**		158	**71**		70	**71**		85	**71**		141	**71**		51
DSJC125.9gb	604*	**604**		7	**604**		1	**604**		159	**604**		3	**604**		87	**604**		13	**604**		14	**604**		9	**604**		11	**604**		11	**604**		6
DSJC125.9g	169*	**169**		3	**169**		0	**169**		180	**169**		0	**169**		134	**169**		12	**169**		8	**169**		9	**169**		7	**169**		7	**169**		4
flat1000_50_0	924	1197	1203.5	3181	**1155**	**1173.8**	3208	1222	1234	1810	1184	1192.8	1886	1206	1215.2	410	1183	1192.8	1640	1170	1189.5	1722	1185	1192	2542	1182	1190.5	1804	1182	1190.8	2542	1183	1193.5	953
flat1000_60_0	1162	1236	1244.4	5124	**1191**	**1205.7**	983	1249	1268.9	6134	1223	1230.8	1394	1244	1251.6	2706	1219	1230.5	2214	1214	1229	1968	1223	1229.8	984	1219	1228.8	2214	1223	1231	2706	1222	1232	994
flat1000_76_0	1165	1212	1219.2	2667	**1176**	**1193.2**	995	1232	1246.6	1245	1195	1208.5	1804	1222	1229.5	492	1204	1208.4	1749	1198	1207.9	820	1199	1208.5	3608	1193	1208	1968	1196	1206.4	2460	1200	1209.8	123
latin_square_10	1480	1533	1543.3	323	**1505**	**1520.3**	3786	1558	1579.1	7001	1520	1535.8	1184	1558	1570.6	2664	1523	1532	3404	1525	1533.7	1998	1525	1534.7	1258	1519	1534.1	888	1527	1533.7	1332	1521	1534.6	74
le450_15a	212	213	213.7	3399	214	236.3	240	213	214.3	4089	214	213.4	238	214	215.7	1734	**212**	213.3	1643	**212**	**213.2**	1487	**212**	213.3	2629	**212**	**213.2**	2414	213	213.5	1783	213	213.6	680
le450_15b	216	217	217.8	4245	218	225.6	42	**216**	217.4	2534	217	217.5	1281	217	218.6	1368	**212**	217.6	1462	217	217.7	2288	**216**	217.2	2700	**216**	**217.1**	2736	**216**	217.3	3384	217	217.7	649
le450_15c	275	279	280.6	2850	281	285.8	33	280	283.2	1832	277	280.1	2204	282	284.6	1064	**276**	280.1	2736	278	280.2	2052	278	280.1	2793	278	280.2	1178	278	280.4	1558	277	**279.9**	133
le450_15d	272	275	277	4580	278	281	273	277	279.1	2115	276	277.1	1550	278	280.8	2242	275	276.7	1691	275	276.5	1985	**274**	**276.1**	3534	275	276.6	1387	275	276.6	2147	275	276.7	725
le450_25a	306	**306**		195	**306**	307.2	1317	**306**		174	**306**		163	**306**		200	**306**		140	**306**		138	**306**		107	**306**		135	**306**		177	**306**		65
le450_25b	307*	**307**		20	**307**	313.4	57	**307**		10	**307**		100	**307**		24	**307**		27	**307**		32	**307**		32	**307**		32	**307**		44	**307**		17
le450_25c	342	347	348.9	6435	350	354.4	5133	348	350.6	4038	347	348.9	1872	350	352.1	1512	348	349	1272	347	348.8	3240	**346**	**348.4**	1656	**346**	348.6	3492	**346**	**348.4**	2520	347	348.9	180
le450_25d	330	334	335.6	6165	**332**	338.9	154	339	341.1	2955	334	335.1	1908	341	343.5	1836	**332**	335.1	648	333	335.2	2952	333	**334.8**	1188	333	335.4	1350	333	335.1	378	334	335.5	602
queen10_10	162	**162**		25	**162**	166.2	465	**162**		14	**162**		76	**162**		18	**162**		27	**162**		31	**162**		25	**162**		27	**162**		22			16
queen10_10gb	164	**164**		613	165	166.2	277	**164**		135	**164**	164.7	2031	**164**		244	**164**		230	**164**		272	**164**		354	**164**		286	**164**		338	**164**		261
queen10_10g	43*	**43**		7	**43**	43.2	18	**43**		3	**43**		13	**43**		11	**43**		7	**43**		17	**43**		11	**43**		11	**43**		8	**43**		5
queen11_11	172	**172**	172.8	2843	174	177.1	760	**172**	172.4	2299	**172**	173.1	1812	**172**	172.6	2085	**172**	172.7	1812	**172**	172.6	2395	**172**	**172.2**	1254	**172**	172.6	1820	**172**	172.5	1659	**172**	172.8	908
queen11_11gb	176	**176**		386	177	178.9	12	**176**		3.19	**176**	176.1	1248	**176**		335	**176**		399	**176**		447	**176**		299	**176**		479	**176**		213	**176**		193
queen11_11g	47	**47**		124	48	48.1	4	**47**		96	**47**		219	**47**		112	**47**		197	**47**		176	**47**		163	**47**		146	**47**		137	**47**		68
queen12_12	185	**185**	185.9	6117	188	190.2	8	**185**	**185.7**	3339	186	186.8	900	**185**	185.8	2136	**185**	185.9	924	186	186.1	2092	**185**	186.1	2121	**185**	186.1	1680	**185**	185.8	1782	**185**	186.2	1578
queen12_12gb	191	**191**	191.2	3495	193	195.3	1475	**191**	**191.1**	2683	**191**	191.7	1738	**191**	191.4	1786	**191**	**191.1**	1052	**191**	**191.1**	1040	**191**	191.2	1178	**191**	191.2	1144	**191**	191.2	822	**191**	191.3	619
queen12_12g	50	**50**		1701	**50**	51.9	1083	**50**		835	**50**		1193	**50**		946	**50**		830	**50**		973	**50**		684	**50**		994	**50**		941	**50**		605
queen13_13	194	**194**	194.2	3490	195	198.7	1592	**194**	195.6	3437	**194**	194.7	1757	195	195.8	1486	**194**	**194.1**	1370	**194**	**194.1**	1255	**194**	194.2	936	**194**	**194.1**	1180	**194**	194.2	1218	**194**	194.2	782
queen14_14	215	216	216.8	1728	217	222.4	18	216	216.9	3670	216	217	1877	216	217.7	2976	216	216.8	1840	216	216.8	1640	**215**	**216.4**	2560	216	216.8	1318	**215**	216.7	2128	216	217.1	1189
queen15_15	223	225	225.8	3031	227	229.7	1126	227	228.1	3026	225	226.4	1230	227	228.2	2492	224	226.1	260	224	225.7	1620	224	226	160	**223**	**225.6**	1700	224	225.9	1880	225	226.2	212
queen16_16	234	236	237.6	5904	237	239.9	92	237	239.4	6267	236	237.4	1760	239	240.4	1485	**234**	236.8	2046	235	236.8	1892	235	236.9	374	235	236.8	3608	235	**236.4**	1529	235	236.8	253
wap01a	545	579	588.5	7128	563	688.4	247	**547**	**551.1**	5920	558	563.5	1392	550	554.2	1566	551	556.5	2871	549	556.7	2610	552	556.1	2784	549	556.2	1740	552	557.6	3480	555	559	928
wap02a	538	549	554	4230	552	586.4	591	**540**	**542.3**	2669	554	560.7	3382	543	547.3	1958	543	549.2	3026	546	550.6	2670	547	550.9	3085	544	551.2	3382	545	550.5	1958	542	551.2	1604
wap03a	562	585	588	6256	**567**	**571.9**	6057	574	577.1	4867	576	580.2	3312	591	643.5	3312	574	577.8	1840	574	579.6	3680	575	579.2	2392	575	578.5	1840	572	578.5	1472	574	579.2	1658
wap04a	563	582	584	1287	**564**	**572**	3013	567	572.5	4893	574	577	1970	597	645.2	3546	573	575.5	2758	572	575.1	3152	572	574.9	3152	571	575.1	2758	572	575.4	2167	571	576.8	1578
wap05a	541	544	545	2062	542	544	6847	**541**	**543.2**	3302	542	543.7	2838	543	545.1	1188	543	544.1	2926	543	543.9	1881	542	543.8	1815	543	544	1874	542	543.5	2013	542	543.9	1179
wap06a	516	525	527.6	5895	**518**	590.4	919	519	**521.1**	4619	520	522.1	2636	521	523.5	2975	519	522.5	2940	519	522	2310	520	522.5	1750	521	522.5	1785	**518**	521.8	840	521	522	677
wap07a	555	572	607.5	2100	729	745.7	0	564	569	2742	**558**	**560.7**	2800	572	575.4	2706	559	561.5	2030	560	562.1	1120	560	561.7	1050	559	561.1	2604	559	561.7	2940	**558**	561.6	735
wap08a	529	546	547.6	2856	**535**	613.7	3828	543	548.6	3649	540	**542.4**	3024	549	552.4	2304	540	543	1440	539	542.8	3744	541	543	1224	541	542.8	2928	541	543.1	432	540	543	1675
#BKS				17/48			11/48			21/48			18/48			16/48			20/48			18/48			22/48			21/48			20/48			18/48	
#Best				17/48			23/48			23/48			19/48			16/48			23/48			19/48			25/48			23/48			22/48			19/48	
#Best Avg				13/48			15/48			19/48			11/48			12/48			15/48			16/48			19/48			17/48			15/48			13/48	

TABLE 8.1 – Résultats sur les 48 instances les plus difficiles du WVCP pour les différentes méthodes.

	MCTS+*UCB*	*RedLS*	*ILS-TS*	HEAD+*RedLS*	HEAD+*ILS-TS*	AHEAD+*Random*	AHEAD+*Roulette*	AHEAD+*Deleter*	AHEAD+*UCB*	AHEAD+*Pursuit*	AHEAD+*NN*
MCTS+*UCB*	-	**27**	**16**	5	**22**	0	0	0	0	0	0
RedLS	11	-	12	10	14	10	10	10	10	10	10
ILS-TS	9	**23**	-	10	**16**	3	2	2	2	4	3
HEAD+*RedLS*	**16**	**28**	**19**	-	**24**	1	1	1	1	1	1
HEAD+*ILS-TS*	6	**21**	4	6	-	0	0	0	1	1	1
AHEAD+*Random*	**17**	**29**	**20**	**7**	**25**	-	0	0	0	0	0
AHEAD+*Roulette*	**16**	**29**	**22**	**9**	**25**	0	-	0	0	0	0
AHEAD+*Deleter*	**17**	**29**	**22**	**7**	**26**	0	0	-	0	0	**1**
AHEAD+*UCB*	**18**	**29**	**22**	**7**	**25**	0	0	0	-	0	0
AHEAD+*Pursuit*	**18**	**29**	**21**	**7**	**27**	0	0	0	0	-	0
AHEAD+*NN*	**16**	**29**	**19**	**6**	**24**	0	0	0	0	0	-

TABLE 8.2 – Comparaison des recherches locales *RedLS*, *ILS-TS*, et du MCTS avec l'hyperheuristique *UCB* lancés pendant 2 heures face aux variantes de HEAD lancées 1 heure avec 2 CPU. Chaque valeur correspond au nombre d'instances où la méthode de la ligne est significativement meilleure que la méthode de la colonne sur les 48 instances les plus difficiles du WVCP. Un nombre est écrit en gras si le nombre d'instances est plus élevé pour la méthode de la ligne.

Malgré l'absence de différences significatives entre les méthodes deux à deux, la stratégie *Random* obtient légèrement moins de meilleurs scores trouvés, ce qui montre l'impact positif de l'apprentissage.

Par ailleurs, nous avons aussi constaté qu'un point faible de l'algorithme mémétique pour le WVCP est le fait de ne garder que la dernière solution légale rencontrée lors de la recherche, ce qui signifie que tout le temps passé à explorer entre cette solution et la fin du temps allouée à la recherche locale pour l'itération courante est totalement perdu. Ce fait explique aussi la raison pour laquelle nous avons choisi un temps de recherche locale pour le WVCP ($0.04 * |V|$ secondes) beaucoup plus grand que dans le cas du GCP ($0.001 * |V|$ secondes). L'algorithme HEAD pour le WVCP fonctionne mieux avec plus de temps dans la recherche locale, car il a besoin de beaucoup parcourir l'espace de recherche pendant la recherche locale pour trouver de bons résultats (à cause des grands plateaux). On a remarqué aussi qu'un algorithme mémétique comme HEAD peut avoir des difficultés pour le WVCP, car les bonnes solutions du WVCP peuvent être très distantes dans l'espace de recherche, ce qui rend leur croisement moins efficace que pour d'autres problèmes comme le GCP (cf. section suivante).

Analyse des Fréquences de Sélection

Les graphiques de la Figure 8.3 présentent les sélections des opérateurs au fil des générations par les différentes stratégies de sélection pour le WVCP, Une ligne par instance (3 différentes) et une colonne par stratégie. Les courbes en nuance de rouge correspondent à *RedLS* et celles en bleu à *ILS-TS*. Plus la couleur est intense, plus le croisement *GPX* est conservateur.

Nous observons trois *patterns* récurrents ici :

- les cas où *ILS-TS* est favori pour les instances telles que queen14_14, DSJC1000.1 ou les queen10/11, les croisements plus conservateurs sont alors préférés ;
- les cas où *RedLS* est préféré, suivi de près par *ILS-TS* avec un croisement conservateur pour les instances telles que DSJC250.1, DSJC500.1, le450_15, queen12/13 ;
- les cas où *RedLS* est favori pour les instances telles que C2000.9, DSJC500.5/9.

Nous remarquons que *RedLS* n'est que très peu affecté par le type de croisement *GPX* utilisé, conservateur ou non, alors qu'*ILS-TS* est plutôt sélectionné avec le croisement *GPX*-90%. Ceci s'explique par le fait que la recherche locale *ILS-TS* intègre déjà un fort mécanisme de perturbation et donc a moins besoin d'un croisement très disruptif.

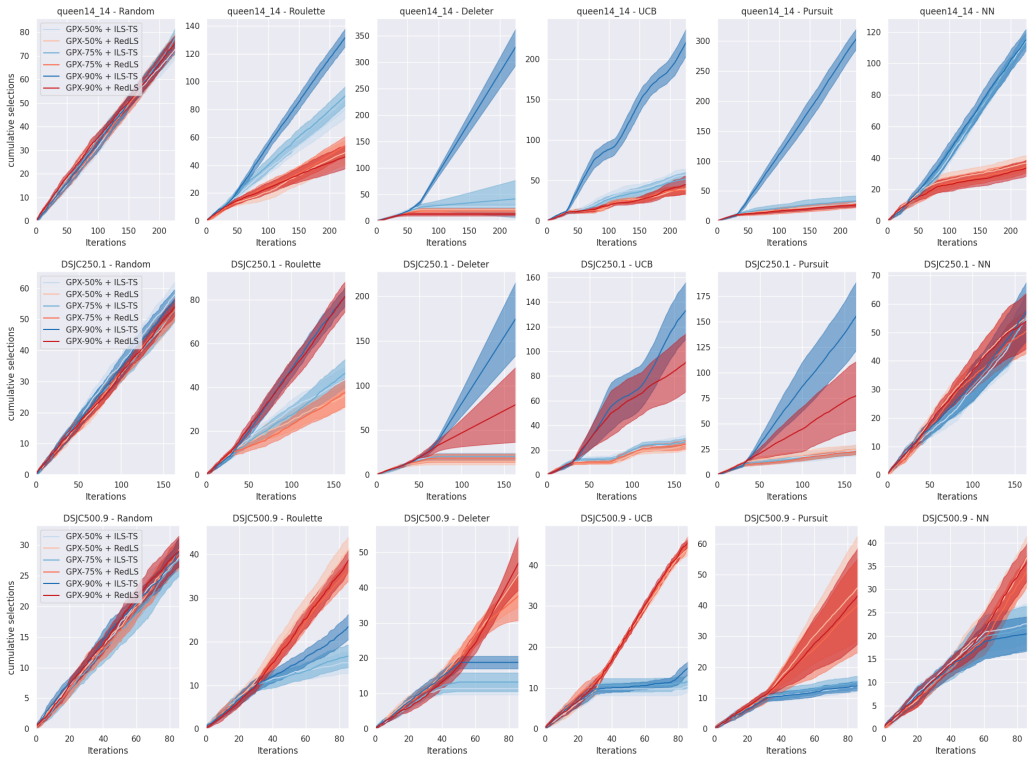


FIGURE 8.3 – Chaque graphique représente la moyenne de la sélection cumulative des opérateurs pour un critère basé sur les 20 exécutions sur une instance, avec barre d'erreur pour le WVCP.

8.3.3 Résultats Expérimentaux sur le GCP

Nous analysons maintenant les résultats obtenus par les différentes méthodes sur le GCP.

Comparaison des Différentes Méthodes

Le Tableau 8.3 étudie les différences des méthodes deux à deux et les Tableaux 8.4 et 8.5 affichent les résultats pour toutes les méthodes sur les 31 instances difficiles étudiées.

En analysant tout d'abord les résultats de HEAD combiné à une seule recherche locale (*PartialCol*, *TabuCol* ou *TabuEdges*), nous observons que *PartialCol* et *TabuCol* sont les procédures de recherche locales qui bénéficient le plus d'une intégration dans le framework HEAD. Au contraire de notre nouvelle procédure *TabuEdges* qui est plus souvent meilleure seule. Nous pouvons aussi constater que dans le cas du GCP, l'intérêt de l'apprentissage face à une sélection aléatoire est globalement plus visible comparé au WVCP.

	PartialCol	HEAD+*PartialCol*	*TabuCol*	HEAD+*TabuCol*	*TabuEdges*	HEAD+*TabuEdges*	AHEAD+*Random*	AHEAD+*Deleter*	AHEAD+*Roulette*	AHEAD+*UCB*	AHEAD+*Pursuit*	AHEAD+*NN*
PartialCol	-	3	2	2	5	8	1	2	1	2	2	2
HEAD+*PartialCol*	**8**	-	6	1	7	10	0	0	1	0	0	1
TabuCol	**14**	**11**	-	2	8	**11**	2	2	0	2	1	2
HEAD+*TabuCol*	**18**	**20**	**12**	-	9	**14**	**5**	3	1	3	3	2
TabuEdges	**17**	**16**	**11**	**11**	-	**12**	8	8	7	8	7	7
HEAD+*TabuEdges*	**15**	**12**	5	3	5	-	0	0	0	0	0	0
AHEAD+*Random*	**18**	**21**	**11**	4	**9**	**13**	-	0	1	0	0	0
AHEAD+*Roulette*	**18**	**21**	**11**	**5**	**9**	**14**	**2**	-	1	0	0	0
AHEAD+*Deleter*	**19**	**22**	**15**	**8**	**11**	**17**	**9**	**6**	-	**6**	**3**	**2**
AHEAD+*UCB*	**18**	**21**	**12**	**4**	8	**14**	**1**	0	0	-	0	0
AHEAD+*Pursuit*	**19**	**21**	**13**	**7**	**10**	**16**	**5**	**3**	0	**2**	-	0
AHEAD+*NN*	**19**	**21**	**12**	**4**	**9**	**15**	**5**	**1**	0	**1**	**1**	-

TABLE 8.3 – Comparaison deux à deux de toutes les méthodes. Un point en plus pour une méthode en ligne signifie que sur une instance de plus la moyenne des scores obtenus est meilleure que celle des scores obtenus par la méthode en colonne et que cette différence est significative selon un test de Wilcoxon où $pvalue < 0.001$.

Les Tableaux 8.4 et 8.5 présentent les résultats des différentes méthodes sur les 31 instances les plus difficiles du GCP. *TabuEdges* trouve avec beaucoup de facilité le meilleur score connu de l'état de l'art à 234 en une trentaine de secondes pour la grande instance r1000.5. On notera que ce score de 234 est très loin d'être atteint même par les meilleures métaheuristiques de la littérature comme [Lü et Hao, 2010, Titiloye et Crispin, 2011, Moalic et Gondran, 2018]. Notre nouvelle recherche locale *TabuEdges* permet aussi de trouver deux nouveaux meilleurs scores pour wap03a (42) et wap07a (40) qui n'avaient pas été trouvés plus tôt dans l'état de l'art. On pourra souligner que le score de 40 pour wap07a est dorénavant prouvé optimal, car les auteurs de [Heule *et al.*, 2022] ont récemment trouvé une borne inférieure à 40 pour ce graphe.

instance	BKS	*PartialCol*			HEAD+*PartialCol*			*TabuCol*			HEAD+*TabuCol*			*TabuEdges*			HEAD+*TabuEdges*		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	145	164	165.2	5313	153	154.2	2779	162	162.8	4628	**148**	**149.2**	3330	167	167.4	4360	160	162.7	2991
C2000.9	408	420	420.8	5171	419	421.1	2731	411	412.5	4786	**405**	406.4	2328	418	419.2	5963	412	415.5	3085
C4000.5	259	304	305.6	6690	288	289.1	3084	303	304.2	5567	**278**	**279.6**	3580	310	310.9	5863	304	307	3232
DSJC500.1	12	**12**		128	**12**		163	**12**		75	**12**		86	13		0	13		0
DSJC500.5	47	50	50.1	2227	**48**	48.6	1443	49		460	**48**		819	50	50.2	1350	52	52.7	1611
DSJC500.9	126	128		975	127	127.2	959	**126**	126.3	2988	**126**		1027	**126**	126.7	996	130	130.4	1407
DSJC1000.1	20	21		1	21		2	21		0	21		0	21	21.1	731	22		0
DSJC1000.5	82	90	90.5	3516	84	84.9	1978	88		1760	**83**	**83.3**	2290	90	90.8	5236	86	87.5	2915
DSJC1000.9	222	227	228.4	3630	226	227.4	3346	224	224.9	3345	**223**	224	1616	**223**	224.6	5089	226	227.3	1112
DSJR500.5	122*	125	126.2	1666	124	125.6	1507	124	127	1155	123	124	1766	**122**		0	**122**		0
flat300_28_0	28*	**28**		896	**28**	28.6	1213	**28**	29.5	3220	30	30.8	1916	31	31.9	1758	32		321
flat1000_50_0	50*	**50**		44	**50**		6	**50**		69	**50**		28	**50**		14	**50**		9
flat1000_60_0	60*	**60**		213	**60**		19	**60**		233	**60**		54	**60**		3205	**60**		68
flat1000_76_0	76*	89	89.1	2845	83	83.8	2534	86	87	3096	**82**	**82.3**	1905	89	89.9	3046	85	86.2	2991
latin_square_10	97	107	110.2	4875	108	109	2699	100	100.8	4377	102	103.7	93	101	101.6	1828	103	103.6	1288
le450_25c	25*	27		69	27		74	26		0	26		0	26		0	26		0
le450_25d	25*	27		50	27		42	26		0	26		0	26		0	26		0
queen11_11	11*	**11**	**11.9**	1303	**11**	**11.9**	909	12		0	12		0	**11**	**11.9**	5418	**11**	**11.9**	701
queen12_12	12*	**13**		4	**13**		4	**13**		0	**13**		0	**13**		3	**13**		0
queen13_13	13*	**14**		20	**14**		19	**14**		0	**14**		1	**14**		4	**14**		1
queen14_14	14*	**15**		585	**15**		513	**15**		20	**15**		9	**15**	15.2	128	**15**		6
r250.5	65*	67		134	**65**	66	2984	66	67.2	462	**65**	66	3378	**65**		0	**65**		0
r1000.1c	98	141	149.1	61	102	104.3	3259	134	155.2	77	100	101.6	264	**98**		8	**98**		4
r1000.5	234	247	248.1	5638	250	252	2657	244	245.6	3622	246	247.6	1479	**234**		30	249	249.2	1474
wap01a	41*	42		1088	42	42.5	1427	42	43	2160	42		137	**41**		5	42		1
wap02a	40*	41	41.7	4275	42		2	**40**	41.1	6499	41		15	**40**		6	41		1
wap03a	43	44		91	45	45.6	513	44	45.9	4342	45		261	**42**	**42.6**	4122	46		752
wap04a	41	43		61	44		123	42	43.1	4869	43		880	**41**	**41.8**	2800	45		41
wap06a	40*	41		98	41		91	**40**	41.3	4248	**40**		909	**40**		0	**40**		0
wap07a	41	44		41	44		136	**41**	42.3	5046	42	42.1	1771	**40**	**40.9**	1242	42	42.5	1767
wap08a	40*	43	43.2	2750	43	43.6	1229	41	41.5	2967	42		48	**40**		185	42		11
#BKS			5/31			6/31			8/31			7/31			15/31			7/31	
#Best			8/31			10/31			10/31			15/31			19/31			10/31	
#Best Avg			8/31			7/31			6/31			13/31			16/31			10/31	

TABLE 8.4 – Résultats sur les instances difficiles du GCP, 1/2.

Pour AHEAD, on remarque que la version *Deleter*, permet d'obtenir globalement le plus grand nombre de meilleurs scores, même si l'avantage reste faible par rapport aux autres stratégies de sélection comme *Pursuit*.

On notera qu'une nouvelle couleur à 404 (contre 408 avant) a aussi été trouvée au cours de ces expérimentations pour l'instance C2000.9 avec les méthodes AHEAD+*Deleter*/*UCB*/*Pursuit*.

instance	BKS	AHEAD+*Random*			AHEAD+*Roulette*			AHEAD+*Deleter*			AHEAD+*UCB*			AHEAD+*Pursuit*			AHEAD+*NN*		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	145	150	151.3	2992	150	150.7	3141	149	149.9	3174	149	150.8	3457	149	150.4	3328	150	150.3	4482
C2000.9	408	**407**	407.5	2445	**405**	406.8	2767	**404**	405.7	3450	**404**	406.4	1946	**404**	**405.4**	3002	**405**	406	5310
C4000.5	259	282	283.6	3545	281	282.1	3450	**278**	280.6	3787	281	282.4	3542	281	282.9	3671	280	281.6	9899
DSJC500.1	12	**12**		162	**12**		95	**12**		60	**12**		127	**12**		79	**12**		107
DSJC500.5	47	**48**		890	**48**	48.1	1067	**48**	48.1	950	**48**		892	**48**		1239	**48**		806
DSJC500.9	126	**126**	126.2	1447	**126**		850	**126**		439	**126**	126.1	1393	**126**		540	**126**		910
DSJC1000.1	20	21		2	21		3	**20**	**20.9**	3149	21		2	**20**	**20.9**	2507	**20**	**20.9**	812
DSJC1000.5	82	**83**	83.8	3196	**83**	83.8	2815	**83**	83.9	1955	**83**	83.8	2189	**83**	83.7	2573	**83**	83.5	3123
DSJC1000.9	222	**223**	224.2	2595	**223**	224.3	2220	**223**	**223.8**	1768	**223**	224.2	2257	**223**	224.1	2606	**223**	224	1939
DSJR500.5	122*	**122**		1	**122**		3	**122**		1	**122**		1	**122**		1	**122**		1
flat300_28_0	28*	**28**	28.8	1784	**28**	28.9	2163	29	30.6	68	**28**	29.6	3246	**28**	29.6	1167	**28**	30.2	2454
flat1000_50_0	50*	**50**		10	**50**		8	**50**		9	**50**		9	**50**		9	**50**		9
flat1000_60_0	60*	**60**		30	**60**		36	**60**		38	**60**		35	**60**		36	**60**		41
flat1000_76_0	76*	**82**	82.8	2578	**82**	82.8	2420	**82**	82.8	2043	**82**	82.7	2648	**82**	82.9	2803	**82**	82.6	2979
latin_square_10	97	103	103.2	1448	103	103.1	1452	**99**	**100.5**	1823	103	103.2	1515	**99**	101.7	3267	100	102.3	2489
le450_25c	25*	26		1	**25**	25.9	2588	**25**	**25.2**	1418	**25**	25.9	2419	**25**	25.4	1710	**25**	25.6	2314
le450_25d	25*	26		1	**25**	25.9	3513	**25**	**25.2**	1320	26		0	**25**	25.6	1907	**25**	25.8	1649
queen11_11	11*	12		0	12		0	12		0	**11**	**11.9**	510	**11**	**11.9**	202	12		0
queen12_12	12*	**13**		0	**13**		0	**13**		0	**13**		0	**13**		0	**13**		0
queen13_13	13*	**14**		1	**14**		2	**14**		1	**14**		1	**14**		1	**14**		1
queen14_14	14*	**15**		13	**15**		17	**15**		16	**15**		7	**15**		15	**15**		13
r250.5	65*	**65**		1	**65**		1	**65**		1	**65**		1	**65**		1	**65**		1
r1000.1c	98	**98**		19	**98**		15	**98**		11	**98**		14	**98**		12	**98**		17
r1000.5	234	248	248.8	1503	246	247.9	2721	243	244.6	1859	247	247.7	1728	246	247.2	2075	246	247.2	5517
wap01a	41*	42		4	42		4	**41**	41.8	1916	42		5	42		3	42		5
wap02a	40*	41		4	41		4	**40**	40.9	2170	41		4	41		4	41		4
wap03a	43	45		211	45		206	44	44.2	1761	45		196	45		161	45		237
wap04a	41	43	43.2	1705	43		1619	42	43	3538	43		1550	43		396	43		987
wap06a	40*	**40**		1	**40**		1	**40**		1	**40**		1	**40**		1	**40**		1
wap07a	41	42	43	30	42	43	2195	42		556	42	42.9	1680	42		1465	42		1878
wap08a	40*	42		64	42		26	42		40	42		40	42		30	42		37
#BKS			10/31			12/31			14/31			12/31			14/31			13/31	
#Best			16/31			18/31			23/31			19/31			22/31			19/31	
#Best Avg			11/31			11/31			16/31			12/31			15/31			13/31	

TABLE 8.5 – Résultats sur les instances difficiles du GCP, 2/2.

Analyse des Fréquences de Sélection

Les graphiques de la Figure 8.4 représentent les sélections cumulées des différents opérateurs de croisement et de recherche locale. Les nuances de rouge correspondent à *TabuCol*, celles en bleu à *PartialCol* et en vert à *TabuEdges*. Plus la couleur est intense, plus le croisement *GPX* utilisé en combinaison est conservateur.

Nous n'affichons que deux graphiques, car les sélections sont très peu différentes, *TabuCol* est systématiquement favori suivi par les deux autres recherches locales dont le choix varie en fonction des instances. On observe que le croisement *GPX*-90%, plus conservateur, est souvent préféré.

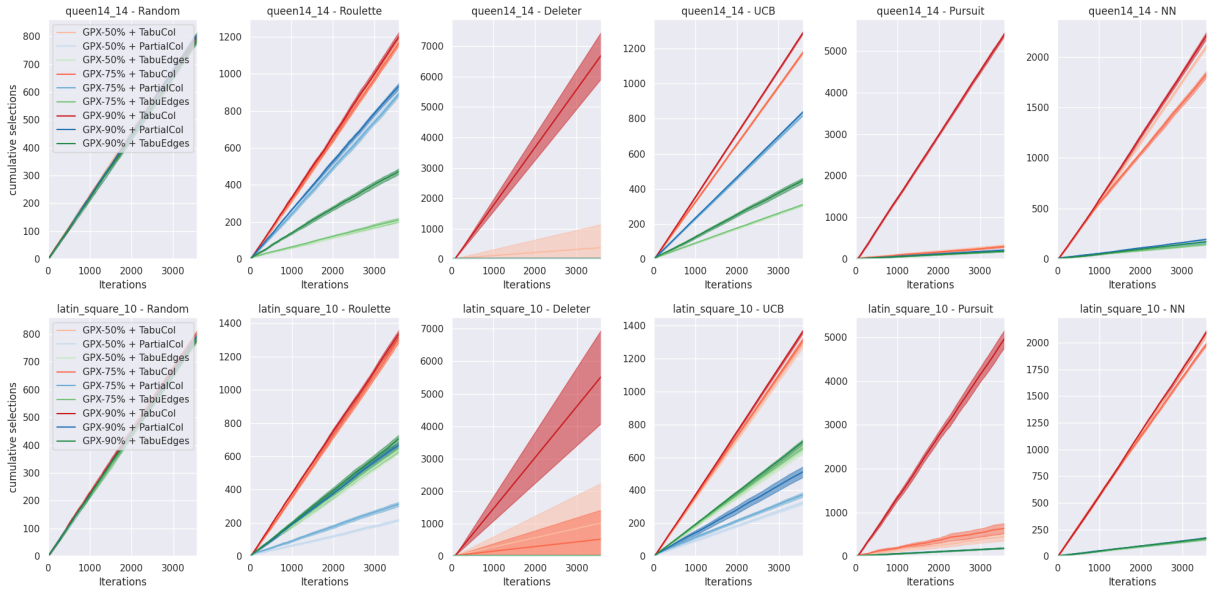


FIGURE 8.4 – Chaque graphique représente la moyenne de la sélection cumulative des opérateurs pour un critère basé sur 20 exécutions pour chaque instance, avec des barres d'erreur, pour le GCP. Il y a une ligne par instance (2 instances différentes), une colonne par méthode adaptative.

8.4 Conclusion

Nous avons proposé AHEAD, un algorithme mémétique basé sur le fonctionnement de HEAD de [Moalic et Gondran, 2018] en y ajoutant des hyperheuristiques pour les problèmes GCP et WVCP. Une population avec deux individus est améliorée en parallèle par des opérateurs de croisement et de recherche locale sélectionnés pendant la recherche par différentes méthodes d'apprentissage. Nous avons aussi proposé dans ce chapitre une nouvelle recherche locale pour le GCP, nommée *TabuEdges*.

Pour les deux problèmes, notre algorithme mémétique adaptatif montre globalement des avantages comparé aux versions sans sélection automatique des opérateurs de bas niveau. La stratégie de sélection qui utilise un réseau de neurones pour faire ses choix est celle qui obtient des résultats légèrement meilleurs en termes de nombre de meilleurs scores obtenus.

Au cours de ces expérimentations, nous avons obtenu trois nouveaux meilleurs scores pour le WVCP (qui n'avaient notamment pas été obtenus après de nombreuses heures de calcul sur un GPU par [Goudet *et al.*, 2022]), ainsi que de nouvelles colorations qui n'avaient jamais été obtenues auparavant dans la littérature pour le GCP. Des variantes de AHEAD ont permis de trouver une nouvelle coloration pour l'instance C2000.9, et notre nouvelle recherche locale *TabuEdges*, utilisée seule, a permis de retrouver très rapidement le score très difficile de 234 pour l'instance r1000.5 et d'obtenir deux nouvelles colorations pour de très grandes instances wap. Ces résultats sont plutôt une surprise, ce qui suggère de creuser plus en détail cette nouvelle procédure de recherche locale pour le GCP.

Ainsi, à l'avenir, il pourrait être intéressant notamment d'étudier un algorithme mémétique utilisant uniquement cette recherche locale *TabuEdges*, mais en transmettant, lors du croisement, non seulement les couleurs de chaque sommet, mais aussi les poids des arêtes appris par cette procédure. Ces poids représentent pour l'algorithme des arêtes dont les sommets ne doivent pas se retrouver dans la même couleur, car leur résolution est difficile. Une version optimisée de cette procédure de recherche locale serait aussi envisageable à la manière de l'optimisation de *TabuCol* réalisé par [Moalic et Gondran, 2018], utilisée aussi dans nos recherches.

CONCLUSION ET PERSPECTIVES

Cette thèse étudie deux problèmes de coloration de graphe, la coloration standard (GCP - *Graph Coloring Problem*) et la coloration pondérée (WVCP - *Weighted Vertex Coloring Problem*). Comme vu dans le Chapitre 1, ces problèmes ont de nombreuses applications réelles, généralement des problèmes d'affectation de ressources. Ces deux problèmes ont été étudiés depuis de nombreuses années, en particulier pour le GCP. Le WVCP a connu un regain d'intérêt assez récemment dans la littérature, avec plusieurs travaux publiés lors des cinq dernières années comme vu dans le Chapitre 2. La littérature sur les problèmes de coloration de graphe est très vaste et beaucoup de méthodes ont été abordées pour résoudre ces problèmes. Les métaheuristiques étant particulièrement efficaces pour les résoudre, guider ces méthodes avec de l'apprentissage peut permettre d'améliorer l'exploration de l'espace de recherche pour atteindre d'excellents résultats.

Nos contributions exposées dans les Chapitres 3 à 8 sont résumées ici.

Dans le Chapitre 3, nous avons étudié dans un premier temps la possibilité de retirer des sommets du graphe pour le WVCP et le GCP. Comme la complexité des algorithmes et le temps passé dans la recherche sont souvent dépendants du nombre de sommets dans le graphe, en retirer semble intéressant tant pour des méthodes exactes que pour des méthodes approchées. Nous avons donc proposé différentes règles de réduction basées sur des améliorations de règles existantes dans l'état de l'art. Ces règles sont ensuite appliquées itérativement sur chaque sommet jusqu'à atteindre un point fixe pour lequel plus aucun sommet ne peut être supprimé. Suite à cette réduction, nous avons observé deux cas de figure très différents en fonction des instances considérées : pour les graphes aléatoires, en général, aucun sommet n'est retiré, alors que pour les graphes avec plus de structure, il est parfois possible de complètement résoudre une instance après la phase de réduction (lorsque le graphe finit par se résumer à une unique clique).

Le Chapitre 4 étudie des bornes supérieures et inférieures sur le nombre de couleurs et le score que peut avoir une solution optimale pour le WVCP. Ces bornes permettent, comme dans le cas de la réduction, de réduire l'espace de recherche ou de limiter l'exploration de celui-ci pour rester dans des zones plus proches de solutions optimales. Ces bornes originalement proposées pour un modèle exacte de programmation par contrainte peuvent aussi être utilisées pour un algorithme heuristique de type MCTS comme celui étudié dans le chapitre suivant.

Dans le Chapitre 5, nous proposons l'utilisation d'un arbre de recherche de Monte Carlo (MCTS - *Monte Carlo Tree Search*) pour le GCP et le WVCP. Nous avons tenu compte de plusieurs particularités pour ce problème, comme l'ordre des sommets à colorer, des limites sur le nombre de couleurs à utiliser à chaque étage de l'arbre (en partie en utilisant les bornes que nous avons trouvées pour le WVCP), des règles d'élagage spécifiques et enfin l'utilisation de différents algorithmes gloutons adaptés au problème lors de la phase de simulation. Nous avons comparé alors cet algorithme à l'état de l'art et nous avons obtenu de bons résultats face à certaines recherches locales dans le cas du WVCP. Pour le GCP en revanche, les performances de l'algorithme n'égalent pas l'état de l'art.

Au vu des performances du MCTS avec un algorithme glouton comme procédure de simulation, dans le Chapitre 6, nous avons étudié cette fois l'utilisation d'une recherche locale lors de la phase de simulation. Nous avons comparé l'utilisation du MCTS en combinaison avec différentes recherches locales de l'état de l'art. Le MCTS peut alors être vu comme un moyen pour initialiser des solutions de bonne qualité pour les recherches locales. Il donnait alors de bons résultats pour le WVCP avec des variations sur la meilleure recherche locale à utiliser en fonction des instances.

Ainsi, nous avons remarqué que certaines recherches locales sont plus performantes pour certaines instances que pour d'autres, ce qui nous a amenés à utiliser des hyperheuristiques dans le Chapitre 7. Ces hyperheuristiques de sélection ont pour objectif de choisir des opérateurs de recherche locale dynamiquement pendant la recherche. Pour faire ce choix, elles utilisent les résultats de chaque opérateur au cours des précédentes itérations, pour mettre à jour leurs critères de sélection. Dans ce cadre, nous avons proposé une stratégie de sélection originale utilisant un réseau de neurones, qui prend en entrée une solution brute et donne en sortie une estimation du score que chaque recherche locale atteint après son utilisation. L'utilisation de plusieurs opérateurs de recherches locales choisies par des sélecteurs automatiques nous a permis d'améliorer en moyenne les résultats des heuristiques de base de la littérature.

Le dernier Chapitre 8 étudie l'utilisation d'algorithmes mémétiques adaptatifs pour les problèmes de coloration de graphe. Nous proposons un algorithme mémétique dont le croisement et la recherche locale sont sélectionnés de façon dynamique pendant la recherche. Dans ce chapitre, nous avons aussi introduit une nouvelle recherche locale pour le GCP permettant d'atteindre de nouveaux meilleurs résultats.

Perspectives

Plusieurs travaux sont possibles pour la suite, comme envisager une amélioration des algorithmes étudiés, ou en proposer de nouveaux.

Concernant la réduction étudiée dans le Chapitre 3, hormis imaginer ou concevoir une nouvelle règle de réduction, il pourrait être possible de chercher à mieux réduire les graphes en améliorant la procédure itérative. Les règles s'appliquent à la manière d'un glouton, si un sommet rencontré peut être supprimé alors on le supprime, mais il est possible que sa suppression bloque celle d'un autre sommet qui aurait éventuellement pu conduire à plus de sommets supprimés.

La réduction pourrait aussi avoir lieu en cours de recherche lors de l'exploration du MCTS étudié dans le Chapitre 5. Après avoir ouvert un nombre de couleurs important, il est possible que dans la branche étudiée, plusieurs sommets puissent être supprimés pour raccourcir la taille de la branche et pour élaguer plus tôt. Le prochain sommet à colorer lors de l'expansion pourrait aussi être choisi avec des critères proches de l'algorithme *DSatur*, plutôt que d'utiliser un tri prédéterminé sur le poids et le degré des sommets. Pour les grandes instances, il serait aussi intéressant d'étudier l'impact d'un algorithme de recherche en faisceau (*beam search*) pour réduire la taille de l'arbre et se concentrer plus rapidement sur des zones de l'espace de recherche plus prometteuses. Utiliser un réseau de neurones préentraîné pour ce choix pourrait être une bonne option.

Concernant les Chapitres 6 et 7, la recherche locale est lancée systématiquement à chaque itération. Celle-ci consomme beaucoup de temps lors de la recherche et l'arbre est bien moins exploré. Nous avons essayé alors plusieurs stratégies pour limiter la fréquence d'utilisation des recherches locales sans succès important. Un réseau de neurones, similaire à celui utilisé avec les hyperheuristiques, pourrait permettre de nous indiquer quel score nous pouvons espérer obtenir en utilisant une recherche locale avec une solution donnée et choisir d'utiliser la recherche locale ou non. Un autre point à étudier avec la combinaison du MCTS et des recherches locales serait la mise à jour de l'arbre par rapport à la coloration finale de la solution. En effet, la recherche locale peut modifier le choix des couleurs des sommets colorés par le MCTS, la branche empruntée n'est donc théoriquement plus la même. Il serait intéressant de voir si retracer la véritable branche de la solution donnée par la recherche locale à la fin de la simulation améliore les résultats. L'algorithme ne chercherait alors plus à trouver de bonnes solutions de départ pour la recherche locale mais à donner des solutions plus proches de ce que la recherche locale a pu atteindre précédemment.

Le Chapitre 8 n'est pas publié, c'est donc le prochain objectif pour ce travail. Pour la suite, nous aimerions travailler aussi sur la recherche locale *TabuEdges* qui obtient de bons résultats, en particulier utilisée seule. Le fait que cette recherche locale soit actuellement efficace seule, mais plus lorsqu'elle est utilisée avec des croisements doit être dû au fait que dans le cadre de l'algorithme mémétique, entre deux itérations, les poids que la recherche locale a attribués aux arêtes souvent en conflit se retrouvent remis à zéro. La procédure *TabuEdges* perd donc des informations importantes qu'elle avait rassemblées pendant la recherche qui aurait pu la guider pendant la suite. Imaginer un algorithme mémétique combiné à cette recherche locale transmettant les poids des arêtes avec les couleurs des sommets pourrait idéalement guider vers de meilleurs résultats. De même, cette attribution de poids sur les arêtes pourrait être utilisée dans le cadre d'autres recherches locales de l'état de l'art pour mieux nuancer le voisinage des solutions en particulier pour le WVCP.

Le travail sur les modèles de programmation par contraintes non présenté dans la thèse dans [Goudet *et al.*, 2023] a aussi pour objectif d'être poursuivi dans le cadre d'une LNS (*Large Neighborhood Search*). En partant d'une solution initiale, alterner des phases d'optimisation avec une recherche locale et l'utilisation d'un modèle CP dont une partie des sommets sont bloqués pourrait mener à de bons résultats sur le WVCP. En analysant pendant la recherche quels sont les sommets les plus importants dans la solution, nous pourrions identifier quels sommets bloquer ou non pour le modèle CP et explorer rapidement des zones de l'espace de recherche plus prometteuses.

BIBLIOGRAPHIE

[Asta *et al.*, 2016] Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J Parkes. Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Information Sciences*, 373 :476–498, 2016. 40, 112

[Blöchliger et Zufferey, 2008] Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3) :960–975, 2008. 32, 43, 129, 130, 139

[Boudhar et Finke, 2000] Mourad Boudhar and Gerd Finke. Scheduling on a batch machine with job compatibilities. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 40(1-2) :69–80, 2000. 17

[Bouziri *et al.*, 2011] Hend Bouziri, Khaled Mellouli, and El-Ghazali Talbi. The k-coloring fitness landscape. *Journal of Combinatorial Optimization*, 21(3) :306–329, 2011. 112

[Brélaz, 1979] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256, 1979. 30, 75, 77, 81, 97

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1) :1–43, 2012. 73

[Burke *et al.*, 2010a] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. *Handbook of metaheuristics*, pages 449–468, 2010. 40

[Burke *et al.*, 2010b] Edmund K Burke, Jakub Mareček, Andrew J Parkes, and Hana Rudová. A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research*, 179 :105–130, 2010. 28

[Burke *et al.*, 2013] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics : A survey of the state of the art. *Journal of the Operational Research Society*, 64 :1695–1724, 2013. 40, 111, 112

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10) :1672–1696, 2011. 33

[Cai et Lin, 2016] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *IJCAI*, pages 568–574, 2016. 50, 51, 52, 169

[Cai, 2003] Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3) :415–429, 2003. 45

[Cazenave *et al.*, 2021] Tristan Cazenave, Benjamin Negrevergne, and Florian Sikora. Monte carlo graph coloring. In *Monte Carlo Search : First Workshop, MCS 2020, Held in Conjunction with IJCAI 2020, Virtual Event, January 7, 2021, Proceedings 1*, pages 100–115. Springer, 2021. 39, 43, 73, 85, 93, 94, 97

[Cazenave, 2009] Tristan Cazenave. Nested monte-carlo search. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009. 39

[Cazenave, 2017] Tristan Cazenave. Nested rollout policy adaptation with selective policies. In *Computer Games : 5th Workshop on Computer Games, CGW 2016, and 5th Workshop on General Intelligence in Game-Playing Agents, GIGA 2016, Held in Conjunction with the 25th International Conference on Artificial Intelligence, IJCAI 2016, New York, USA, July 9-10, 2016, Revised Selected Papers 5*, pages 44–56. Springer, 2017. 39

[Cheeseman *et al.*, 1991] Peter C Cheeseman, Bob Kanefsky, William M Taylor, et al. Where the really hard problems are. In *IJCAI*, volume 91, pages 331–337, 1991. 45, 46, 49

[Chiarandini *et al.*, 2002] Marco Chiarandini, Thomas Stützle, et al. An application of iterated local search to graph coloring problem. In *Proceedings of the computational symposium on graph coloring and its generalizations*, pages 112–125. Ithaca New York (USA), 2002. 34

[Cornaz *et al.*, 2017] Denis Cornaz, Fabio Furini, and Enrico Malaguti. Solving vertex coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 217 :151–162, 2017. 29, 60, 70, 86, 99

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006. 39, 73

[Culberson, 2001] Joseph Culberson. Hidden solutions, tell-tales, heuristics and anti-heuristics. In *The IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence, edited by H. Hoos and T. Stuëtzle*, volume 9, page 14. Citeseer, 2001. 18

[Dantas *et al.*, 2021] Augusto Dantas, Alexander Fiabane do Rego, and Aurora Pozo. Using deep Q-network for selection hyper-heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1488–1492, 2021. 113, 116

[Demange *et al.*, 2007] Marc Demange, D de Werra, Jérôme Monnot, and V Th Paschos. Time slot scheduling of compatible jobs. *Journal of Scheduling*, 10(2) :111–127, 2007. 16, 61, 67, 68, 69, 76

[Dorne et Hao, 1998] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In *Parallel Problem Solving from Nature—PPSN V : 5th International Conference Amsterdam, The Netherlands September 27–30, 1998 Proceedings 5*, pages 745–754. Springer, 1998. 36, 43

[Dorne et Hao, 1999] Raphaël Dorne and Jin-Kao Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In *Meta-heuristics : Advances and trends in local search paradigms for optimization*, pages 77–92. Springer, 1999. 32

[Dowsland et Thompson, 2008] Kathryn A Dowsland and Jonathan M Thompson. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156(3) :313–324, 2008. 38, 43

[Drake *et al.*, 2020] John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2) :405–428, 2020. 41, 112

[Edelkamp et Greulich, 2014] Stefan Edelkamp and Christoph Greulich. Solving physical traveling salesman problems with policy adaptation. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014. 73

[Elhag et Özcan, 2015] Anas Elhag and Ender Özcan. A grouping hyper-heuristic framework : Application on graph colouring. *Expert Systems with Applications*, 42(13) :5491–5507, 2015. 40, 112

[Formanowicz et Tanaś, 2012] Piotr Formanowicz and Krzysztof Tanaś. A survey of graph coloring-its types, methods and applications. *Foundations of Computing and Decision Sciences*, 37(3) :223–238, 2012. 16

[Furini et Malaguti, 2012] Fabio Furini and Enrico Malaguti. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2) :130–136, 2012. 17, 29

[Galinier *et al.*, 2013] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. Recent advances in graph vertex coloring. *Handbook of Optimization : From Classical to Modern Approach*, pages 505–528, 2013. 29

[Galinier et Hao, 1999] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3 :379–397, 1999. 32, 36, 43, 130, 132, 169

[Garey et Johnson, 1979] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979. 14

[Gelly *et al.*, 2006] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. *Modification of UCT with patterns in Monte-Carlo Go*. PhD thesis, INRIA, 2006. 73

[Glass et Prügel-Bennett, 2003] Celia A Glass and Adam Prügel-Bennett. Genetic algorithm for graph coloring : exploration of Galinier and Hao's algorithm. *Journal of Combinatorial Optimization*, 7 :229–236, 2003. 34

[Goëffon *et al.*, 2016] Adrien Goëffon, Frédéric Lardeux, and Frédéric Saubion. Simulating non-stationary operators in search algorithms. *Applied Soft Computing*, 38 :257–268, 2016. 41, 112, 116, 119

[Gomes et Shmoys, 2002] Carla Gomes and David Shmoys. Completing quasigroups or latin squares : A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, pages 22–39, 2002. 18

[Gondran et Moalic, 2019] Alexandre Gondran and Laurent Moalic. Optimality clue for graph coloring problem. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research : 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings 16*, pages 337–354. Springer, 2019. 21

[Goudet *et al.*, 2021] Olivier Goudet, Béatrice Duval, and Jin-Kao Hao. Population-based gradient descent weight learning for graph coloring problems. *Knowledge-Based Systems*, 212 :106581, 2021. 38, 43

[Goudet *et al.*, 2022] Olivier Goudet, Cyril Grelier, and Jin-Kao Hao. A deep learning guided memetic framework for graph coloring problems. *Knowledge-Based Systems*, 258 :109986, 2022. 11, 37, 40, 42, 43, 70, 88, 99, 109, 118, 130, 132, 151

[Goudet *et al.*, 2023] Olivier Goudet, Cyril Grelier, and David Lesaint. New bounds and constraint programming models for the weighted vertex coloring problem. *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 1927–1934, 2023. 11, 29, 42, 45, 55, 59, 61, 70, 86, 99, 156

[Grelier *et al.*, 2022] Cyril Grelier, Olivier Goudet, and Jin-Kao Hao. On monte carlo tree search for weighted vertex coloring. In *Evolutionary Computation in Combinatorial Optimi-*

zation : 22nd European Conference, EvoCOP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings, pages 1–16. Springer, 2022. 10, 41, 73, 85, 97, 99, 101, 102, 104, 113, 115, 116, 129, 140

[Grelier *et al.*, 2023] Cyril Grelier, Olivier Goudet, and Jin-Kao Hao. Monte carlo tree search with adaptive simulation : A case study on weighted vertex coloring. In *Evolutionary Computation in Combinatorial Optimization : 23rd European Conference, EvoCOP 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings*, pages 98–113. Springer, 2023. 11, 41, 111, 121, 124

[Gualandi et Malucelli, 2012] Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1) :81–100, 2012. 28

[Halldórsson et Shachnai, 2008] Magnús M Halldórsson and Hadas Shachnai. Batch coloring flat graphs and thin. In *Algorithm Theory–SWAT 2008 : 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008. Proceedings 11*, pages 198–209. Springer, 2008. 17

[Hébrard et Katsirelos, 2020] Emmanuel Hébrard and George Katsirelos. Constraint and satisfiability reasoning for graph coloring. *Journal of Artificial Intelligence Research*, 69 :33–65, 2020. 28

[Held *et al.*, 2012] Stephan Held, William Cook, and Edward C Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4) :363–381, 2012. 28

[Hertz *et al.*, 2008] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13) :2551–2560, 2008. 32

[Hertz et Werra, 1987] Alain Hertz and D de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4) :345–351, 1987. 32, 43, 67, 94, 101, 129, 130, 139

[Heule *et al.*, 2022] Marijn JH Heule, Anthony Karahalios, and Willem-Jan van Hoeve. From cliques to colorings and back again. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022. 28, 148

[Holland, 1992] John H Holland. Genetic algorithms. *Scientific American*, 267(1) :66–73, 1992. 34

[Hoos, 2012] Holger H Hoos. Automated algorithm configuration and parameter tuning. *Autonomous search*, pages 37–71, 2012. 41, 119

[Huang *et al.*, 2019] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with alphagozero. *arXiv preprint arXiv :1902.10162*, 2019. 39, 43

[Ignatiev *et al.*, 2017] Alexey Ignatiev, Antonio Morgado, Joao Marques-Silva, and ISDCT SB RAS. Cardinality encodings for graph optimization problems. In *IJCAI*, pages 652–658, 2017. 28

[Jansen et Kratsch, 2013] Bart MP Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Information and Computation*, 231 :70–88, 2013. 45

[Johnson *et al.*, 1991] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing : an experimental evaluation ; part ii, graph coloring and number partitioning. *Operations research*, 39(3) :378–406, 1991. 18

[Johnson et Trick, 1996] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability : second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996. 18, 19

[Jooken *et al.*, 2023] Jorik Jooken, Pieter Leyman, Tony Wauters, and Patrick De Causmaecker. Exploring search space trees using an adapted version of monte carlo tree search for combinatorial optimization problems. *Computers & Operations Research*, 150 :106070, 2023. 39, 73, 79

[Junker *et al.*, 1999] Ulrich Junker, Stefan E Karisch, Niklas Kohl, Bo Vaaben, Torsten Fahle, and Meinolf Sellmann. A framework for constraint programming based column generation. *CP*, 99 :261–274, 1999. 28

[Khalil *et al.*, 2017] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017. 39

[Kingma et Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014. 119, 141

[Koster, 2005] AMCA Koster. Wavelength assignment in multifiber wdm networks. *Proceedings of INOC 2005*, pages 60–66, 2005. 19

[Kubale et Jackowski, 1985] Marek Kubale and Boguslaw Jackowski. A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, 28(4) :412–418, 1985. 28, 75

[Lai *et al.*, 1985] Tze Leung Lai, Herbert Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1) :4–22, 1985. 39, 79

[Lee, 2002] Jon Lee. All-different polytopes. *Journal of Combinatorial Optimization*, 6 :335–352, 2002. 28

[Leighton, 1979] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6) :489, 1979. 18, 30, 81

[Lewandowski et Condon, 1993] Gary Lewandowski and Anne Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. *Cliques, Coloring, and Satisfiability*, 26, 1993. 18

[Lewis, 2015] Rhyd Lewis. *A guide to graph colouring*, volume 7. Springer, 2015. 15, 16, 75

[Li *et al.*, 2018] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018. 39

[Lin *et al.*, 2017] Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In *IJCAI*, pages 517–523, 2017. 45

[Liu *et al.*, 2006] Huadong Liu, Micah Beck, and Jian Huang. Dynamic co-scheduling of distributed computation and replication. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 9–pp. IEEE, 2006. 17

[Lü et Hao, 2010] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1) :241–250, 2010. 36, 43, 130, 132, 148

[Lucas *et al.*, 2018] Thomas Lucas, Corentin Tallec, Yann Ollivier, and Jakob Verbeek. Mixed batches and symmetric discriminators for GAN training. In *International Conference on Machine Learning*, pages 2844–2853, 2018. 118

[Malaguti *et al.*, 2009] Enrico Malaguti, Michele Monaci, and Paolo Toth. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15 :503–526, 2009. 29, 60, 99

[Malaguti *et al.*, 2011] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2) :174–190, 2011. 28, 67

[Malaguti et Toth, 2010] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International transactions in operational research*, 17(1) :1–34, 2010. 28, 29

[Mehrotra et Trick, 1996] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4) :344–354, 1996. 28

[Mısır *et al.*, 2013] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A new hyper-heuristic as a general problem solver : an implementation in hyflex. *Journal of Scheduling*, 16 :291–311, 2013. 40

[Moalic et Gondran, 2018] Laurent Moalic and Alexandre Gondran. Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics*, 24 :1–24, 2018. 32, 36, 43, 67, 130, 131, 132, 134, 137, 141, 142, 148, 151

[Morris, 1993] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the eleventh national conference on Artificial intelligence*, pages 40–45, 1993. 139

[Mostafaie *et al.*, 2020] Taha Mostafaie, Farzin Modarres Khiyabani, and Nima Jafari Navimipour. A systematic study on meta-heuristic approaches for solving the graph coloring problem. *Computers & Operations Research*, 120 :104850, 2020. 38

[Nogueira *et al.*, 2021] Bruno Nogueira, Eduardo Tavares, and Paulo Maciel. Iterated local search with tabu search for the weighted vertex coloring problem. *Computers & Operations Research*, 125 :105087, 2021. 31, 34, 43, 70, 77, 86, 88, 99, 100, 102, 116, 123, 129, 130, 140

[Porumbel *et al.*, 2010a] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37(10) :1822–1832, 2010. 37, 43, 130, 132

[Porumbel *et al.*, 2010b] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. A search space "cartography" for guiding graph coloring heuristics. *Computers & Operations Research*, 37(4) :769–778, 2010. 24, 37

[Porumbel *et al.*, 2011] Daniel Cosmin Porumbel, Jin Kao Hao, and Pascale Kuntz. An efficient algorithm for computing the distance between close partitions. *Discrete Applied Mathematics*, 159(1) :53–59, 2011. 23

[Prais et Ribeiro, 2000] Marcelo Prais and Celso C Ribeiro. Reactive grasp : An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12(3) :164–176, 2000. 17, 19, 30, 43, 52, 86, 99

[Previti *et al.*, 2011] Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte-carlo style uct search for boolean satisfiability. In *Congress of the Italian Association for Artificial Intelligence*, pages 177–188. Springer, 2011. 39

[Régin, 2003] Jean-Charles Régin. Using constraint programming to solve the maximum clique problem. In *Principles and Practice of Constraint Programming–CP 2003 : 9th International Conference, CP 2003, Kinsale, Ireland, September 29–October 3, 2003. Proceedings 9*, pages 634–648. Springer, 2003. 28

[Ribeiro *et al.*, 1989] Celso Carneiro Ribeiro, Michel Minoux, and Manoel Camillo Penna. An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operational Research*, 41(2) :232–239, 1989. 17, 29

[Sabar *et al.*, 2012] Nasser R Sabar, Masri Ayob, Rong Qu, and Graham Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1) :1–11, 2012. 41, 112

[Sabar et Kendall, 2015] Nasser R Sabar and Graham Kendall. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences*, 314 :225–239, 2015. 40, 112

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587) :484–489, 2016. 38, 73

[Silver *et al.*, 2017a] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv :1712.01815*, 2017. 38

[Silver *et al.*, 2017b] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676) :354–359, 2017. 38

[Sun *et al.*, 2018] Wen Sun, Jin-Kao Hao, Xiangjing Lai, and Qinghua Wu. Adaptive feasible and infeasible tabu search for weighted vertex coloring. *Information Sciences*, 466 :203–219, 2018. 19, 31, 33, 43, 52, 88, 99, 100, 102, 116, 129, 140

[Thierens, 2005] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pages 1539–1546, 2005. 41, 119

[Titiloye et Crispin, 2011] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2) :376–384, 2011. 38, 43, 148

[Trick et Yildiz, 2007] Michael A Trick and Hakan Yildiz. A large neighborhood search heuristic for graph coloring. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems : 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007. Proceedings 4*, pages 346–360. Springer, 2007. 32

[Wang *et al.*, 2020] Yiyuan Wang, Shaowei Cai, Shiwei Pan, Ximing Li, and Monghao Yin. Reduction and local search for weighted graph coloring problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2433–2441, 2020. 31, 33, 43, 45, 46, 54, 55, 56, 59, 60, 69, 88, 99, 100, 102, 116, 123, 129, 131, 139, 140

[Welsh et Powell, 1967] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1) :85–86, 1967. 30, 81

[Wu et Hao, 2012] Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2) :283–290, 2012. 38, 43

[Zaheer *et al.*, 2017] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. 113, 118

[Zhou *et al.*, 2016] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Reinforcement learning based local search for grouping problems : A case study on graph coloring. *Expert Systems with Applications*, 64 :412–422, 2016. 40

[Zhou *et al.*, 2018] Yangming Zhou, Béatrice Duval, and Jin-Kao Hao. Improving probability learning based local search for graph coloring. *Applied Soft Computing*, 65 :542–553, 2018. 40, 43

[Zhou *et al.*, 2020] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Frequent pattern-based search : A case study on the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 52(3) :1503–1515, 2020. 40, 43

LISTE DES PUBLICATIONS

Articles de conférence

- New Bounds and Constraint Programming Models for the Weighted Vertex Coloring Problem (O. Goudet, **C. Grelier**, D. Lesaint). 2023. *Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI 2023, Macao, SAR, China*
- Monte Carlo Tree Search with Adaptive Simulation : A Case Study on Weighted Vertex Coloring (**C. Grelier**, O. Goudet, J-K Hao). 2023. *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar). Springer, Cham. Brno, Czech Republic.*
- On Monte Carlo Tree Search for Weighted Vertex Coloring (**C. Grelier**, O. Goudet, J-K Hao). 2022. *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar). Springer, Cham.*

Articles de revue

- A deep learning guided memetic framework for graph coloring problems. (O Goudet, **C. Grelier** and J-K Hao). 2022. *Journal Knowledge-Based System.*
- Combining Monte Carlo Tree Search and Heuristic Search for Weighted Vertex Coloring (**C. Grelier**, O. Goudet, J-K Hao). **En cours de relecture pour la revue *SN Computer Science*. Prépublication arXiv :2304.12146v.**

Communication dans des conférences nationales

- Algorithme mémétique guidé par l'apprentissage profond pour des problèmes de coloration de graphe. (O. Goudet, **C. Grelier**, J-K Hao). 2023. *24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2023), 2023, Rennes, France.*
- Sélection automatique d'opérateurs dans un arbre de recherche de Monte-Carlo pour la coloration de graphe pondéré. (**C. Grelier**, O. Goudet, J-K Hao). 2023. *24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2023), 2023, Rennes, France.*

- Recherche arborescente Monte-Carlo pour la coloration de graphe pondéré. (**C Grelier**, O Goudet and J-K Hao). 2022. *23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision.*

Travail en cours

- Sélection automatique d'opérateurs de croisement et de recherche locale dans le cadre d'un algorithme mémétique pour le GCP et le WVCP . (**C Grelier**, O Goudet and J-K Hao). (non publié).

LISTE DES ALGORITHMES

LISTE DES FIGURES

LISTE DES TABLEAUX

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

université angers

Titre : Métaheuristiques Guidées par l'Apprentissage pour la Coloration de Graphe

Mot clés : Coloration de graphe, Arbre de Recherche de Monte Carlo, Recherche Locale, Algorithmes Mémétiques, Apprentissage Automatique, Hyperheuristiques

Résumé : Dans cette thèse, nous nous intéressons à la résolution de problèmes de coloration de graphe. Tout d'abord, nous étudions des propriétés théoriques de ces problèmes, qui permettent de réduire en pratique le nombre de variables et l'espace de recherche pour des instances difficiles de graphe. Nous présentons ensuite des méthodes de type Monte Carlo Tree Search (MCTS), qui mettent en jeu une recherche dans l'arbre des solutions légales guidée par de l'apprentissage. Pour en améliorer les performances, nous avons hybridé ce MCTS avec des algorithmes de recherche locale. Toujours dans le cadre du MCTS, nous avons ensuite cherché à apprendre à sélectionner les bons opérateurs de recherche locale de manière adaptative au cours de la résolution. Une comparaison de différentes techniques d'apprentissage pour le choix de ces opérateurs, allant de stratégies probabilistes simples à des techniques d'apprentissage profond, est étudiée. Enfin, nous avons étendu ce travail sur les hyperheuristiques, au cadre des algorithmes mémétiques, qui alternent l'utilisation d'opérateurs de recherche locale avec des opérateurs de croisement au sein d'une population de solutions candidates.

Title: Learning-Guided Metaheuristics for Graph Coloring

Keywords: Graph Coloring, Monte Carlo Tree Search, Local Search, Memetic Algorithm, Machine Learning, Hyperheuristics

Abstract: In this thesis, we focus on solving graph coloring problems. First, we study theoretical properties of these problems, which allow us to reduce in practice the number of variables and the search space for difficult graph instances. We then present Monte Carlo Tree Search (MCTS) methods for these problems, which involve a search in the legal solution tree guided by learning. To improve performance, we hybridize this MCTS with local search algorithms. Still within the MCTS framework, we then sought to learn how to select the right local search operators adaptively during the search. A comparison of different learning techniques for selecting these operators, ranging from simple probabilistic strategies to more complex deep learning techniques, is studied. Lastly, we have extended this work on hyperheuristics to the framework of memetic algorithms, which alternate the use of local search operators with crossover operators within a population of candidate solutions.