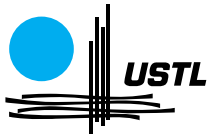


UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

U.F.R. D'I.E.E.A.

Numéro d'Ordre : 3467 Année : 2004

THÈSE

pour obtenir le grade de

DOCTEUR DE L'U.S.T.L.

DISCIPLINE : INFORMATIQUE

présentée et soutenue publiquement,

le 16 septembre 2004, par

BENJAMIN WEINBERG

Titre :

ANALYSE ET RÉSOLUTION APPROCHÉE DE PROBLÈMES D'OPTIMISATION COMBINATOIRE : APPLICATION AU PROBLÈME DE COLORATION DE GRAPHE

Jury :

Président :	Sophie Tison	Professeur, Université de Lille I
Rapporteurs :	Marc Schoenauer	Directeur de Recherche INRIA
	Alexandre Caminada	Professeur, Université de Belfort-Monbéliard
Examinateurs :	Denis Robilliard	Maître de Conférence, Université du Littoral
Directeur :	El-Ghazali Talbi	Professeur, Université de Lille I

Table des matières

Table des figures

Remerciements

Je tiens tout d'abord à remercier le Madame Sophie Tison, professeur des Université de Lille I, qui m'a fait l'honneur de présider le jury.

J'adresse mes remerciements les plus sincères à messieurs Marc Schoenauer, directeur de Recherche INRIA, et Alexandre Caminada, Professeur des Universités de Belfort-Montbéliard qui ont pris le temps de rapporter cette thèse.

Je remercie également Monsieur Denis Robilliard, maître de conférences à l'Université du Littoral-Côte d'Opale, pour sa participation au jury.

Je tiens également à remercier tous ceux qui mon aider à l'aboutissement de ce travail. Je pense bien évidemment à ma familles, mes parents et mes frères, qui m'ont offert sans compter leur soutient moral et encouragements nécessaires ; à mes amis, Titia, Cec, Thieu et Stéphane, qui m'ont permis de trouver l'énergie nécessaire à l'accomplissement de ce projet.

Je ne serait pas honnête si je ne signalais pas le soutien des membres de l'équipe : Clarisse Dhanaens, dont chaque entretien fut un enrichissement ; Nouridine Melab, pour sa franchise et ses conseils avisés ; El-Ghazali Talbi, dont sa direction m'a permis de développer mon autonomie et mon esprit critique ; les thésards de l'équipe (présent et passé), Laetitia, Nicolas, Matthieu, Julien pour leurs relectures et corrections du manuscrit.

Je profite de ce quelques lignes pour signaler l'importance de certains professeurs croisés durant ma scolarité et mes études : Mlle François, Mlle Pigeon, Mr Piette, Mr Sacré.

Chapitre 1

Introduction

Cette thèse s'inscrit dans le domaine de l'optimisation combinatoire. Elle a été réalisée au sein de l'équipe Optimisation PArallèle Coopérative (OPAC) du Laboratoire d'Informatique Fondamentale de Lille (LIFL).

Un problème d'optimisation combinatoire se caractérise généralement par un ensemble fini de configurations $\mathcal{X}$ et une fonction objectif $f : \mathcal{X} \rightarrow \mathcal{Y}$ avec $\mathcal{Y}$ munie d'une relation d'ordre notée $\prec$. Typiquement en optimisation monocritère, $\mathcal{Y}$ est $\mathbb{R}$ ou $\mathbb{N}$ et la relation d'ordre $\prec$ est $\leq$ pour les problèmes de minimisation ou $\geq$ pour les problèmes de maximisation. Résoudre un problème d'optimisation, c'est trouver la configuration $c^\star$ (ou un ensemble de configurations O) tel que $\forall c \in \mathcal{X}, f(c^\star) \prec f(c)$ (ou $\forall c \in \mathcal{X}, \forall c^\star \in O f(c^\star) \prec f(c)$) [GJ79]. L'équipe OPAC s'intéresse aux méthodes de résolution approchées de tels problèmes. Les travaux sont généralement illustrés par l'étude de problèmes difficiles comme le problème de flowshop de permutation (FSP), le problème d'élaboration de tournées de véhicules (VRP) et certains problèmes d'extraction de connaissances. Pour notre part, nous nous sommes particulièrement intéressés au problème de coloration de graphe détaillé dans la suite du manuscrit. Tous ces problèmes appartiennent à la classe des problèmes NP-difficiles. C'est pourquoi, il est souvent préféré une résolution à base d'heuristiques sur les plus problèmes de grandes tailles pour des raisons de temps d'obtention de résultat.

Les différences de comportements des heuristiques "classiques" sur les différents problèmes suggèrent l'existence de problèmes plus ou moins faciles à optimiser. La nécessité d'expliquer ces différences de comportements est renforcée par le fait que de nombreuses approches sont stochastiques et que deux exécutions d'un même algorithme sur le même problème donnent des résultats différents. L'étude des paysages consiste en l'analyse des problèmes permettant de prédire le comportement (éventuellement moyen) d'un algorithme (ou d'une famille d'algorithmes). Pour notre

part, nous nous sommes focalisés sur les études du paysages basées sur des relevés statistiques. Bien qu'il existe des travaux établissant des méthodologies afin de classifier les problèmes à partir de ces relevés, il est parfois nécessaire d'expliciter comment calculer ces relevés, afin de respecter au mieux les spécificités du problème étudié.

L'obtention de résultats de différentes qualités, selon les problèmes et les méthodes utilisées, incite à combiner les algorithmes afin de bénéficier d'un maximum d'avantages. Cette technique s'appelle l'hybridation d'algorithmes et a obtenu un certain succès ces dernières années. Cependant, ces techniques impliquent souvent un surcoût de calculs, qui ne se justifie que si la qualité des solutions trouvée est meilleure. Toutes les hybridations ne sont pas toujours fructueuses.

Dans cette thèse, nous avons apporté des contributions dans les quatres axes suivants :

au niveau de la **théorie de l'optimisation**, nous discutons du théorème du No Free Lunch (**NFL**) [WM97]. Ce théorème établit les limites théoriques à l'application des méthodes heuristiques. Cette étude nous a permis de déboucher sur une notion de structure de classe de problèmes. Nous montrons que les classes de problèmes de k-coloration de graphe respectent cette définition pour tout k supérieur à 3. De ce fait, nous montrons que le théorème **NFL** n'est pas applicable à la classe des problèmes de k-coloration.

D'un point de vue **algébrique**, nous présentons des résultats théoriques sur la symétrie du problème de coloration et nous proposons des moyens techniques permettant d'opérer malgré cette symétrie. Plus précisément, nous proposons une structure de données particulière pour coder un partitionnement. Nous proposons également une formalisation de l'espace de recherche comme un ensemble quotienté par une relation d'équivalence. Cette formalisation nous a permis de développer des outils algorithmiques pour opérer sur cet espace quotient. En particulier, nous avons développé un test d'égalité entre partitionnements en $O(n)$, avec n le nombre de sommets du graphe à colorer alors qu'il existe $k!$ façons différentes de coder le même partitionnement. Nous avons également revisité l'opérateur de voisinage élémentaire. Les résultats obtenus peuvent être appliqués à tous les problèmes de partitionnement (aussi appelés "grouping problems" [Fal98]) : graph coloring, clustering, bin packing, *etc*. Dans ce cadre, le calcul de la distance est non-triviale. Nous avons défini la matrice de raffinement entre deux partitionnements. Cet outil formel nous a permis de définir deux métriques sur l'espace des partitionnements. Les distances sont calculables en temps polynomial. La première compte le nombre exact de mouvements élémentaires pour passer d'un partitionnement à un autre. La seconde est plus économique en temps de calcul et corrélé avec la première. Elle présente par

ailleurs un biais qui est intéressant pour le problème de coloration de graphe. Après cela nous présentons un moyen de calculer l'entropie d'une population de coloration. Ces outils nous ont permis d'effectuer une classification des benchmarks classiques pour le problème de coloration de graphe.

D'un point de vue **génie logiciel**, la mise en œuvre de l'algorithme s'est effectuée à l'aide de la plateforme logicielle **EO** [KMRS01]. Nous en rappelons les principes, puis nous proposons un moyen de modéliser les parties de notre algorithme. Nous avons utilisé cette modélisation pour présenter, d'une part une extension à la plateforme facilitant l'hybridation d'algorithmes et d'autre part des assemblages **EO** que nous avons effectués lors de notre implémentation de composants de notre approche. La force de l'approche utilisé pour la coloration de graphe réside dans sa conception intrinsèquement parallèle. Nous présentons donc une implémentation distribuée de l'algorithme à l'aide de l'environnement PVM [GBD+94].

Au niveau de la **résolution de problèmes difficiles**, nous proposons une façon d'implémenter une métaheuristique évolutionnaire coopérative pour un problème d'optimisation combinatoire. Notre approche a pour objectif d'équilibrer explicitement l'effort de recherche entre les tâches d'intensification et de diversification. Nous précisons les moyens à mettre en œuvre pour sa réalisation. En particulier, nous précisons le fonctionnement de la mémoire adaptative dont le rôle est l'agrégation des informations, afin de mettre en place explicitement l'équilibre entre l'intensification et la diversification. Nous présentons une application de cette méthode au problème de coloration de graphe. L'évaluation de performances de notre heuristique nous a permis de constater la pertinence de deux nouveaux opérateurs de diversification.

Au cours du challenge ROADéF, nous avons également proposé une heuristique pour le problème d'affectation de fréquences avec polarité. L'énoncé du challenge spécifiait une temps de calcul court pour la résolution. C'est pourquoi, nous avons utilisé une approche basé sur une recherche locale, baptisée SIG (*Split Iterated Greedy*). Nous avons adapté l'algorithme aux spécificités du problème en utilisant des variables complexes, des fonctions pour mieux discriminer les configurations et des mécanismes tabous supplémentaires. Les travaux ménés nous ont permis d'atteindre la phase finale du challenge.

Le mémoire s'articule en quatre chapitres :

Le **chapitre 2** rappelle la pertinence de l'utilisation des heuristiques stochastiques dans le domaine de l'optimisation. Puis nous entamons une discussion sur un théorème d'impossibilité : le théorème No Free Lunch (**NFL**). Ce théorème affirme qu'il n'existe pas d'heuristique universelle capable de résoudre efficacement l'ensemble des problèmes d'optimisation. Cependant, on constate dans la communauté l'utilisation fréquente des métaheuristiques pour résoudre les problèmes les plus durs

(entre autres ceux de la classe NP-difficile). Notre discussion constate cette apparante contradiction et relativise l'utilisation du **NFL** en précisant les hypothèses du théorème selon deux axes : les métaheuristiques sont-elles des heuristiques au sens du **NFL** et cherche-t-on vraiment à résoudre tous les problèmes d'optimisation ? Après avoir pris des précautions suffisantes vis à vis du **NFL**, nous présentons des outils statistiques pour l'analyse des problèmes d'optimisation. Ces outils sont communément regroupés sous le terme générique d'analyse du paysage. En particulier, nous présenterons des indicateurs statistiques dont certains seront utilisés dans les parties suivantes.

Le **chapitre 3** présente le problème de coloration. Il rassemble un certain nombre d'informations sur le problème ainsi que sur les façons dont il a déjà été traité. Dans cette partie, nous identifions et formalisons également la symétrie de l'espace de recherche. Cette symétrie de l'espace augmente considérablement la taille de l'espace de recherche puisqu'il y a un facteur exponentiel entre le nombre de configurations et le nombre de façons de les représenter. Nous présentons en particulier dans ce chapitre un "test d'égalité" qui permet de déterminer efficacement si deux codages différents représentent ou non la même coloration d'un graphe. Ensuite nous donnons un nouvel éclairage sur l'opérateur de voisinage élémentaire en effaçant la symétrie de l'espace de recherche.

Dans le **chapitre 4**, nous présentons nos travaux au sujet de l'analyse du paysage du problème de coloration de graphe. Après un bref aperçu d'autres travaux sur le sujet, nous exploitons le formalisme produit dans le chapitre précédent et nous proposons deux distances sur l'espace des colorations. Ces deux métriques ne souffrent aucunement de la symétrie de l'espace de recherche. Elles sont toutes deux corrélées à l'utilisation de l'opérateur de voisinage élémentaire du problème (re)défini dans le chapitre précédent. Les résultats obtenus dans ces deux chapitres vont au delà du problème de coloration de graphe, et peuvent être appliqués à l'ensemble des problèmes de partitionnement. Cependant, nous discutons de leurs applications au regard de leurs complexités et des biais qu'elles engendrent. Nous terminons ce chapitre en effectuant une classification des benchmarks classiques de la coloration de graphe utilisant entre autre la plus économique en temps de calculs des deux métriques proposées.

Dans le **chapitre 5**, nous proposons une approhe coopérative parallèle pour la coloration de graphe. Elle a comme principe d'équilibrer explicitement l'intensification et la diversification au cours de la résolution. D'un point de vue développement, nous avons utilisé la plateforme logicielle **EO** et la bibliothèque PVM pour la parallélisation. Dans ce chapitre, nous proposons un moyen de présenter les algorithmes développés sous **EO** grâce à des schémas "simples" inspirés par UML. Pour finir, nous évaluons une implémentation de **Cosearch** pour le problème de coloration de

graphe. Pour cela, nous spécifions trois agents d'intensification et quatre agents de diversification. Bien que les meilleurs résultats n'ont pas été améliorés, un opérateur inspiré par les travaux sur la symétrie du chapitre 4 paraît prometteur.

Nous terminons ce mémoire en présentant nos conclusions et quelques perspectives à nos travaux.

En complément de ce manuscrit, nous avons placé deux annexes.

L'annexe A est un glossaire rappellant les notations et definitions techniques utilisées dans le manuscrit.

L'annexe B présente les travaux que nous avons effectués pour le challenge ROADéF 2001. Les travaux présentés ici nous ont permis d'atteindre la phase finale du challenge. Cette annexe propose, en particulier, une heuristique locale économique en temps de calcul. Nous avons baptisé cette approche SIG (*Split iterated Greedy*) et nous l'avons appliquée au problème d'affectation de fréquence avec polarité. Dans ce cadre, nous avons proposé et évalué des mécanismes spécifiques comme l'utilisation de fonctions discriminant les configurations de même coût, l'utilisation de variables complexes (appelées liens) et des adaptations de mécanismes tabous.

Chapitre 2

Optimisation combinatoire et paysage

Dans ce chapitre, nous positionnons nos travaux en optimisation combinatoire. En effet, nous sommes souvent assujettis dans ce domaine à deux contraintes contradictoires :

- *l'impossibilité technique de résoudre exactement les problèmes NP-difficiles dans un temps raisonnable ;*
- *l'impératif de fournir à un décideur une solution de la meilleure qualité possible.*

Pour cela, de nombreuses méthodes de résolution approchées ont vu le jour. Lorsque le problème le permet, des méthodes exactes sont préférées, dans le cas contraire des approches heuristiques sont employées.

*Nous commençons par rappeler des méthodes génériques de résolution (exactes et approchées). Puis dans le cadre des approches heuristiques, nous discutons du théorème du No Free Lunch (***NFL***)[WM97]. Ce théorème établit les limites théoriques à l'application des méthodes heuristiques. Cette étude nous a permis de déboucher sur une notion de structure de classe de problèmes. Nous montrons que les classes de problèmes de k-coloration de graphe respectent cette définition pour tout k supérieur à 3. De ce fait, nous montrons que le théorème* **NFL** *n'est pas applicable à la classe des problèmes de k-coloration. Ensuite, nous présentons un modéle formel pour le paysage d'un problème d'optimisation. Pour finir, nous présentons des méthodes génériques d'analyses des problèmes. Ces analyses statistiques sont souvent regroupées sous le terme d'étude de paysage ("fitness landscape").*

Ce travail a été publié dans les conférences suivantes :

- *B. Weinberg and E-G Talbi,* **NFL theorem is unusable on struc-**

tured classes of problems, *Congress of Evolutionary Computation, Portland, Oregon, Juin 2004.*
- *B. Weinberg and E-G Talbi,* **Fitness landscape for metaheuristic design**, *International Symposium on Combinatorial Optimization, Paris, France, Avril 2002.*

2.1 Méthodes de résolution

Nous traitons ici des méthodes de résolution pour des problèmes d'optimisation combinatoire pour lesquels aucun algorithme polynomial n'a, à ce jour, été découvert. Dans le but d'obtenir une solution de meilleure qualité possible, différentes techniques ont vu le jour. Certains modèles sont suffisamment génériques pour pouvoir être appliqués à de nombreux problèmes, on parle alors de métaheuristiques. Nous présentons, dans cette section, ces méthodes de résolution générales réparties en deux grandes catégories : premièrement, les méthodes exactes, basées sur des algorithmes arborescents ; deuxièmement, des méthodes approchées dont la complexité permet d'appréhender des problèmes plus grands mais dont la solution fournie ne garantit pas l'optimalité de la solution. Pour un problème d'optimisation donné, on définit par *espace de recherche* l'ensemble des solutions potentielles ou *configurations*. Par exemple pour un problème de type voyageur de commerce (TSP), une configuration est une tournée visitant toutes les villes. Dans le problème de coloration, que nous étudions dans la suite de notre mémoire, une configuration est l'attribution d'une couleur à chaque sommet du graphe.

2.1.1 Méthodes exactes

L'espace de recherche des problèmes d'optimisation combinatoire présente une particularité : cet espace est fini. En effet, il peut être vu comme l'ensemble des affectations possibles d'un ensemble fini de valeurs à des variables. Par conséquent, il est théoriquement toujours possible de tester toutes les solutions potentielles (ou *configurations*). Cependant du fait de la combinatoire de cet espace, il est *pratiquement* impossible de le faire. Prenons par exemple le cas du problème de MAXSAT : ce problème consiste à affecter des valeurs booléennes à n variables de manière à maximiser le nombre des clauses satisfaites parmi un ensemble de m clauses données. Pour ce problème, l'espace de recherche est de 2^n. Si le problème est de taille importante, la durée de vie de l'univers ne suffirait pas pour obtenir le résultat (voir tableau 2.1).

Les algorithmes de recherche exacte sont basés sur une recherche arborescente. À chaque noeud de l'arbre, l'affectation d'une valeur à une variable est "discutée" (voir figure 2.1). La profondeur de l'arbre est donc le nombre de variables du problème.

n	$\lvert\mathcal{R}\rvert$	temps
10	> 1000	10^-6 sec
30	$> 10^9$	1 sec
50	$> 10^{15}$	> 9 mois
70	$> 10^{21}$	> 760 siècles
90	$> 10^{27}$	$> 9.10^{17}$ millénaires

Ce tableau suppose qu'on traite un problème de MAXSAT de taille n. Dans la seconde colonne ($|\mathcal{R}| = 2^n$) figure une approximation de la taille de l'espace de recherche. La dernière colonne (temps) fournit une estimation du temps de calcul nécessaire pour parcourir l'espace de recherche en supposant qu'une configuration est générée et comparée à la meilleure solution obtenue en un milliardième de seconde. Ceci reste une approximation (minoration) puisque ce temps de génération dépend de la taille du problème et de la machine (actuellement les machines ont des fréquences de quelques Ghz).

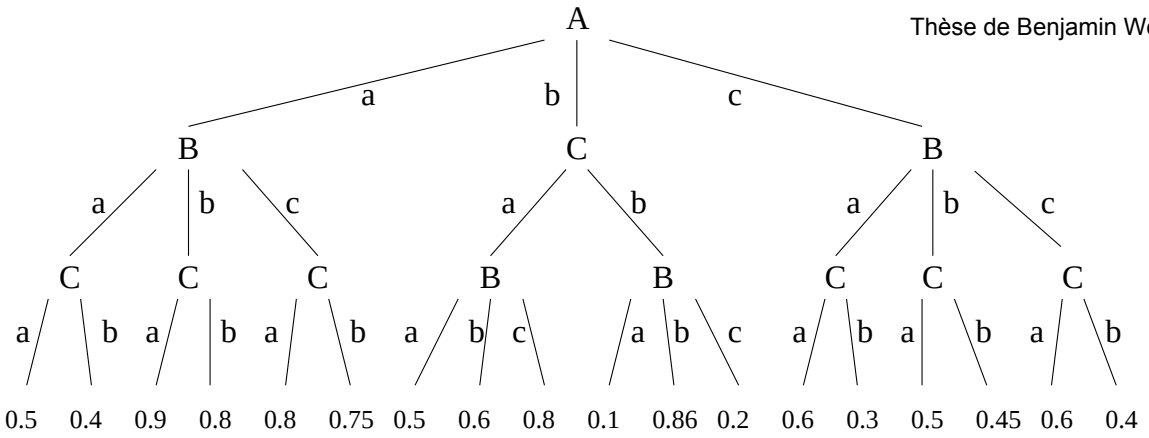
TAB. 2.1 – Illustration du temps d'exploration de l'espace de recherche.

Si l'arbre est entièrement exploré alors l'algorithme a entièrement exploré l'espace de recherche. Dans la communauté "programmation par contraintes", ces techniques sont communément appelées des méthodes de "backtracking" ou de "forward checking". La différence entre ces appellations provient du moment où on effectue le test permettant de détecter si la solution construite vérifie les contraintes du problème. Les contraintes sont propagées par le "forward checking", alors que le "backtracking" vérifie toutes les contraintes nécessaires au moment de l'affectation d'une variable.

Si on peut caractériser des parties de l'arbre à ne pas explorer, alors on peut éviter l'exploration de nombreuses configurations inutiles. On dit alors qu'on élague l'arbre. Nous rappelons succintement deux exemples de techniques d'élagage : le "branch and bound" et le "branch and cut". Remarquons qu'il existe des solveurs complexes combinant différentes techniques comme les méthodes arborescentes présentées ci-après, les méthodes de programmation dynamique, etc. Afin de clarifier le discours, nous exposons les différentes techniques pour des problèmes de minimisation.

2.1.1.1 Branch and Bound

Cette technique nécessite de pouvoir calculer une borne inférieure pour le sous-problème restant. Au moment de chaque affectation, on calcule une estimation de la meilleure valeur de fitness possible. Si cette dernière est moins intéressante que le fitness de la meilleure configuration trouvée jusqu'alors, on peut en déduire que le sous-arbre correspondant ne doit pas être développé. Le choix (le calcul) de la borne inférieure est fortement lié au problème.



Sur chaque branche, une valeur est affectée à la variable du noeud père. Sur cet exemple les variables sont A, B et C ; les valeurs qu'elles peuvent prendre n'appartiennent pas forcément au même ensemble. Ainsi A et B ont trois valeurs possibles (a, b et c) et C n'en a que deux (a et b). On indique au niveau des feuilles le fitness de la configuration. L'ordre d'affectation des variables peut varier selon les branches de l'arbre.

FIG. 2.1 – Arborescence d'une construction de l'algorithme de recherche.

2.1.1.2 Branch and Cut

La technique de "Branch and Cut" s'applique à des problèmes sous la forme de programme linéaire. Bien que soluble en un temps polynomial lorsque les variables sous toutes réelles, le problème devient NP-difficile pour des variables entières, booléennes ou mixtes. Cette méthode consiste à chercher la meilleure solution réelle puis à ajouter des contraintes supplémentaires forçant les variables non-entières à prendre des valeurs entières. Par exemple, si une variable x vaut $2,5$ dans le programme optimal de la relaxation continue, on résoudra deux problèmes. Le premier contiendra la contrainte supplémentaire $x \leq 2$, le second la contrainte $x \geq 3$.

2.1.2 Méthodes approchées

Les méthodes exactes présentées ci-dessus ont une complexité exponentielle dans le cas général. Pour des raisons pratiques, des méthodes dont le temps d'exécution est réduit doivent être utilisées. Ces méthodes approchées ne fournissent pas automatiquement une solution exacte mais des solutions de bonne qualité. Nous présentons ces trois grandes catégories de méthodes :

- les méthodes constructives ;
- les recherches locales ;
- les algorithmes évolutionnaires.

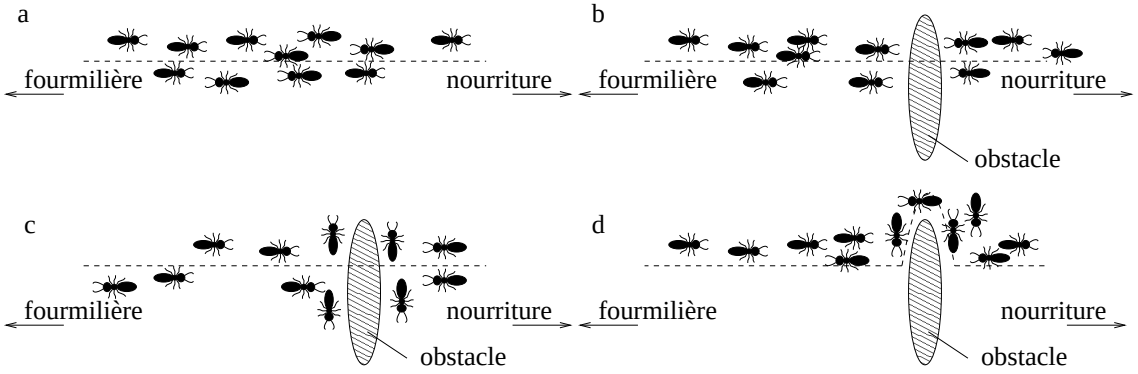
2.1.2.1 Méthodes constructives

Les méthodes de recherche arborescente font partie des méthodes constructives qui garantissent l'optimalité de la solution trouvée. Cependant, même avec l'élagage, cette méthode s'avère trop coûteuse en temps pour certains problèmes. Les méthodes présentées ici, sont basées sur la stratégie gloutonne dont l'idée consiste à ne parcourir qu'une seule branche de l'arbre de recherche. Puisqu'il n'y a pas de remise en cause des choix effectués précédemment, la méthode gloutonne est généralement d'exécution rapide. Comme pour l'algorithme "Branch and Bound", cette méthode est fortement liée au problème. De plus, il peut exister plusieurs méthodes gloutonnes pour un même problème.

La puissance des ordinateurs actuels permet de considérer l'utilisation de méthodes plus coûteuses que les méthodes gloutonnes déterministes mais aussi plus performantes. Par conséquent, d'autres algorithmes dont des méthodes constructives ont donc vu le jour :

- On peut par exemple appliquer une des méthodes exactes vues précédemment, mais en bornant le temps de calcul. Il est intéressant de remarquer que les variables situées en haut de l'arbre de recherche sont moins facilement remises en question. Il est donc important de savoir "bien" affecter les premières variables, ce qui n'est pas toujours évident.
- Recherche aléatoire gloutonne (greedy randomized construction) : cette méthode est en fait un composant de la méthode GRASP qui sera détaillée par la suite. La recherche aléatoire gloutonne est une version stochastique d'un algorithme glouton. Dans cette méthode, la branche explorée dépend du choix d'une graine. Il est donc possible de générer plusieurs solutions différentes. Cette procédure est appelée plusieurs fois avec des graines différentes. La meilleure solution obtenue est mémorisée comme résultat final. Cette méthode peut, contrairement à la méthode précédente, équilibrer l'importance des différentes variables du problème.
- Dans certains problèmes, une configuration peut servir de modèle pour la construction d'une nouvelle solution de meilleure qualité. On peut alors itérer le procédé. Le critère d'arrêt utilisé est souvent un nombre d'itérations sans amélioration. Cette technique, très spécifique, est rencontrée par exemple sur le problème de coloration, elle sera donc détaillée dans le partie traitant de ce problème.
- Les algorithmes de type "colonies de fourmis" (ANTS) ajoutent de la coopération aux algorithmes constructifs. Pour cela, une analogie est faite entre un algorithme constructif et l'insecte cherchant de la nourriture. Schématiquement, on considère que des fourmis partent de la fourmilière et cherchent une source de nourriture. Sur leur passage, elles laissent une substance chimique (appelée phéromone) qui peut être repérée par leurs congénères. Initialement, les fourmis explorent le terrain en se déplaçant (quasiment) au hasard. Puis

quand une ressource est trouvée par une fourmi, l'information est communiquée aux autres par l'intermédiaire d'une phéromone déposée sur le trajet pour une exploitation rapide par toute la colonie. D'un point de vue algorithmique, une fourmi construit une configuration avec une recherche aléatoire gloutonne. Après la construction de la configuration, les choix effectués sont marqués en fonction de la qualité de la configuration obtenue. On itère ainsi le procédé en donnant une plus grande probabilité au choix qui se sont montrés profitables (voir figure 2.2). Au cours de l'algorithme intervient aussi un phénomène "d'évaporation" qui efface progressivement la phéromone, diminuant les probabilités des choix de trajet qui ne se montrent plus suffisamment profitables.



Cette figure illustre le comportement de la colonie en quatre étapes.
a) Les fourmis suivent une piste de phéromone menant à la nourriture (pointillé).
b) Un obstacle coupe la piste de phéromone.
c) Les fourmis choisissent aléatoirement le chemin du haut ou de bas.
d) Un nouvelle piste de phéromone se construit : le meilleur chemin est gardé. La phéromone est effacée sur les éventuels mauvais choix et sous l'obstacle.

FIG. 2.2 – Mécanisme des colonies de fourmis.

2.1.2.2 Méthodes locales

Les méthodes à configuration unique consistent à améliorer successivement une configuration complète. Elles sont dérivées de la recherche locale (Hill Climbing) (voir figure 2.3). Toutes les recherches locales sont basées sur une notion de voisinage. La notion de voisinage consiste à construire pour chaque configuration un sous-ensemble de l'espace de recherche, qu'on appelle voisinage de la configuration. La recherche locale travaille avec une configuration courante, qui est remplacée par l'une de ses voisines. Ce procédé est itéré afin d'obtenir la solution finale. Pour un même problème donné, il est possible d'utiliser plusieurs opérateurs de voisinage. Par

exemple, dans le problème du TSP, les opérateurs city-swap, 2-opt, 3-opt, n-opt, et LK (pour ne citer qu'eux) peuvent être utilisés [FRPT99].

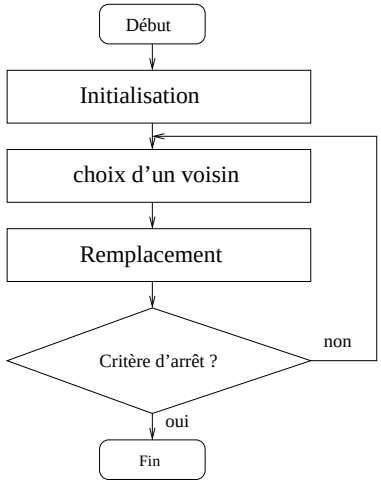


FIG. 2.3 – Schéma d'une recherche locale.

Cependant, la recherche locale présente un inconvénient important. En effet, la solution trouvée peut n'être qu'un optimum local, c'est à dire que la configuration trouvée n'est pas la solution optimale du problème, mais ne présente aucune configuration de meilleure qualité dans son voisinage.

Afin de pallier à ce défaut, différents mécanismes ont été envisagés dans la littérature. Premièrement, le recuit simulé utilise une métaphore empruntée au recuit de la sidérurgie. En sidérurgie, les métaux sont fondus puis refroidis lentement afin d'obtenir une cristallisation régulière, et ainsi des matériaux de bonne qualité. En pratique, le recuit simulé (**SA**) est une technique stochastique basée sur un paramètre communément appelé température, évoluant au cours de l'exécution. Le recuit simulé utilise les critères de Metropolis [KGV83] pour l'acceptation de la configuration voisine en fonction de la perturbation de la qualité de la configuration et de la température. Initialement, la température est élevée autorisant une forte dégradation de qualité puis décroît pour diminuer la probabilité des dégradations importantes. À la fin de l'algorithme, la configuration courante ne peut pratiquement évoluer que vers une configuration de meilleure qualité.

Bien que des résultats théoriques prouvent une convergence asymptotique du recuit simulé sous certaines conditions, il n'est pas garanti d'atteindre l'optimum global du problème en un temps restreint. De plus, dans la pratique, le paramètre de la température peut être difficile à régler.

La recherche taboue (**TS**) est une autre amélioration de la recherche locale permettant de sortir d'un optimum local. Cet algorithme, introduit par Glover [GL93],

utilise une mémoire appelée liste taboue. En effet, cet algorithme mémorise les configurations déjà parcourues pour ne les visiter qu'une seule fois. Cette métaheuristique gère un ensemble de configurations interdites : on dit alors que ces configurations sont taboues. À chaque itération, l'algorithme choisit la meilleure solution voisine qui n'est pas taboue. Ainsi, dans un premier temps, l'algorithme se comporte comme un algorithme de descente classique, puis une fois un optimum local obtenu, il continue à chercher parmi les meilleures configurations proches de cet optimum, puis en s'éloignant progressivement de l'optimum local trouvé. Cependant, la liste taboue est techniquement limitée par la mémoire de la machine, il est donc nécessaire de trouver une parade à cette contrainte. Généralement, seuls les attributs des mouvements sont stockés en mémoire. Cette technique permet à l'algorithme de ne pas retourner sur ses pas. Or généralement, l'interdiction des "mouvements retours" interdit également des configurations non visitées. Il est donc nécessaire de limiter le caractère tabou d'un mouvement. La première technique consiste à limiter l'interdiction dans le temps, c'est à dire de ne faire porter l'interdiction de certains mouvements que sur un nombre limité d'itérations. On parle alors de la taille de la liste taboue. La seconde consiste à lever le caractère tabou si la configuration obtenue par le mouvement vérifie un critère d'aspiration (généralement être meilleur que la meilleure configuration trouvée jusqu'alors). Par ailleurs, des versions élaborées de **TS** utilisent d'autres mécanismes de mémorisation, comme une mémoire à long terme. Cette mémoire permet de mémoriser des informations sur les "régions" inexplorées de l'espace de recherche à l'aide de matrice de fréquences, par exemple. Ainsi une nouvelle exploration de cette partie pourra être initiée. Ce mécanisme a pour but de garantir une exploration équitable de l'espace de recherche en limitant le biais induit par la configuration initiale.

2.1.2.3 Méthodes à base d'ensemble de configurations

Le caractère stochastique de certaines méthodes à configuration unique peut provoquer des variations de la qualité des résultats selon les exécutions. On dit alors que la méthode n'est pas robuste. Par ailleurs, les recherches locales sont rapides et peuvent donc être exécutées plusieurs fois afin d'augmenter leur robustesse. Les méthodes multi-départs (**MLS**) consistent à exécuter plusieurs recherches locales de manière indépendante à partir de configurations différentes. Le meilleur résultat des différentes exécutions sera le résultat final de la méthode. La méthode GRASP applique ce principe en construisant les points de départ avec une recherche aléatoire gloutonne.

Les autres méthodes à configurations multiples réduisent l'indépendance des évolutions des configurations et suggèrent une coopération (entre elles) pour trouver plus vite ou plus sûrement une meilleure solution. L'une des méthodes à configurations multiples les plus connues, est l'algorithme génétique (**AG**) [Gol94]. Cette méthode est basée sur une métaphore darwinienne de la sélection naturelle : "Les plus

adaptées survivent et se reproduisent, tandis que les autres espèces disparaissent." La coopération des configurations s'effectue par le biais d'opérateurs agissant sur un multi-ensemble de configurations, appelé population. Les configurations y sont identifiées à des individus. La population est initialement constituée de configurations générées aléatoirement selon une distribution uniforme sur l'espace de recherche. Ainsi, la population initiale dispose d'une grande couverture de l'espace de recherche. La population évolue de génération en génération en utilisant les opérateurs génétiques :

- La **sélection** permet d'élire un certain nombre d'individus pour la reproduction. Différents mécanismes de sélection existent. Généralement ces opérateurs ont tendance à choisir les meilleurs individus pour participer à la phase de reproduction.
- Les individus sélectionnés se **reproduisent** deux à deux. Chaque paire engendre deux nouveaux individus. Cette opération est effectuée par l'utilisation d'une recombinaison (croisement ou crossover). Les enfants obtenus peuvent subir alors une mutation, c'est-à-dire une perturbation aléatoire et locale. La probabilité d'apparition de ce phénomène est un paramètre de l'algorithme.
- Certains individus de la nouvelle génération **remplacent** des individus de la population courante. Là encore, le remplacement a tendance à privilégier les individus de meilleure qualité, en utilisant des mécanismes soit déterministes soit stochastiques.

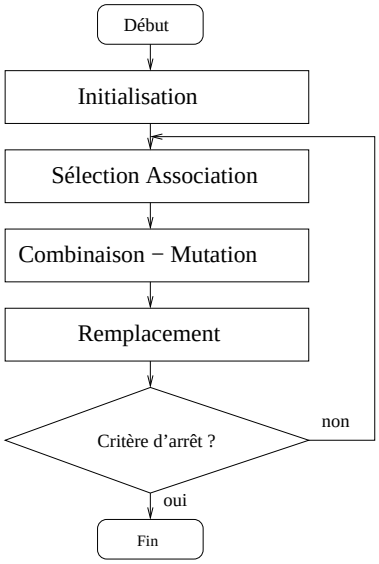


FIG. 2.4 – Schéma d'un Algorithme Évolutionnaire.

Par ce procédé, on espère pouvoir observer une amélioration de la qualité globale de la population. Les algorithmes évolutionnaires (**AE**) sont une généralisation des **AG** (voir figure 2.4). Ce terme générique regroupe tous les algorithmes travaillant sur une population et utilisant des opérateurs génétiques. Dans la pratique, les **AE**

offrent plus de souplesse que les **AG**. Ils sont donc souvent des hybridations bas niveau de composants spécifiques au problème dans un **AG**. Dans cette catégorie, on retrouve entre autres, les algorithmes mémétiques (**MA**) (lamarckien et baldwinien).Les **MA** simulent la longévité d'une solution, dans le sens où les configurations sont modifiées de génération en génération. Les **MA** lamarckiens effectuent des recherches locales sur les configurations de la population, afin de ne garder que des optima locaux. Les **MA** baldwiniens effectuent les recherches locales pour mesurer le "potentiel" d'une configuration. Les optima locaux ne sont cependant pas utilisés lors de la reproduction afin de conserver une diversité de population suffisante.

La recherche par dispersion est une méthode d'optimisation relativement ancienne décrite par Glover [Glo77]. Cette approche évolutionnaire a pour origine les stratégies de création de règles de décision composées et de contraintes de remplacement.

Les études récentes ont montré les avantages pratiques de cette approche pour résoudre divers problèmes d'optimisation. La recherche par dispersion contraste avec d'autres procédures évolutionnaires, telles que les algorithmes génétiques, en utilisant des conceptions stratégiques là où d'autres approches utilisent l'aléatoire.
La recherche par dispersion opère sur un ensemble de solutions appelé l'ensemble de référence, en les combinant pour en créer de nouvelles. À la différence d'une "population" dans les algorithmes génétiques, l'ensemble de référence de solutions dans la recherche par dispersion est relativement petit.

La recherche par dispersion comprend les cinq composants suivants :
- Une méthode de génération de diversification. Ce composant permet de générer (ou de regénérer) un ensemble de configurations couvrant l'ensemble de l'espace de recherche.
- Une méthode d'amélioration pour transformer une configuration en une nouvelle configuration de meilleure qualité.
- Une méthode de mise à jour de l'ensemble de référence pour construire et maintenir un ensemble de références comprenant les meilleures configurations trouvées, organisées pour fournir l'accès efficace à d'autres parties de la méthode. L'incorporation de solutions à l'ensemble de références est effectuée selon leur qualité ou leur diversité.
- Une méthode de génération d'un sous-ensemble pour opérer sur l'ensemble de référence, pour produire un sous-ensemble de ces configurations comme une base pour créer des configurations combinées.
- Une méthode de combinaison de configurations pour transformer un sous-ensemble donné de configurations produites par la méthode de génération d'un sous-ensemble en un ou plusieurs vecteurs combinés de solutions.

2.1.3 Efficacité des heuristiques

Au cours des dernières décennies, ces algorithmes stochastiques ont montré leur efficacité ainsi que leur robustesse sur des problèmes difficiles. Pour expliquer comment le "hasard" peut fournir d'aussi bons résultats, plusieurs justifications ont été formulées.

La théorie des schémas [Gol94] suggère la présence de schémas communs dans les solutions de bonne qualité. Les schémas représentent une configuration partielle. Par exemple, si les configurations sont des chaînes de bits de longueur 8, on peut avoir comme schémas : $10******$, $****1***$, $*0**11*1$, etc. Où l'étoile signifie la non spécification d'un bit. De plus, les **AG** augmentent exponentiellement en théorie la présence des schémas intéressants pour aboutir à une solution optimale sous certaines conditions. Cependant, cette théorie s'est avérée fausse en général à cause de l'existence de problèmes déceptifs et/ou la possibilité de phénomènes d'épistasie. Ce terme désigne un phénomène de marquage ou d'inversion d'un gène en fonction du reste du génome. En d'autres termes, le sens d'un gène n'est pas seulement contenu dans le gène mais dépendant des autres. Dans un **AG**, il signifie généralement une dépendance forte entre les différentes variables du problème. Les problèmes déceptifs forment un ensemble de problèmes pour lesquels le comportement d'un **AG** conduit à une solution non optimale. En effet, ces problèmes sont construits de telle façon que l'algorithme converge vers une configuration appelée attracteur déceptif. Sous certaines hypothèses, Whitley prouve que cet attracteur est en fait le complément binaire de la solution optimale, à un bit près [Whi91]. Depuis, d'autres approches ont été utilisées, comme le paysage de fitness [WS93].

2.2 Théorème NFL

Des théorèmes d'impossibilité théorique existent dans de nombreux domaines. On peut citer comme exemple le théorème de Shannon, qui affirme qu'il ne peut pas exister d'algorithme de compression universel, ou le théorème de Gödel énonçant l'existence de résultats non prouvables. Depuis 1995, il en existe aussi un en optimisation, il s'agit du théorème No Free Lunch (**NFL**). Cependant, il arrive qu'il soit mal compris et qu'il soit utilisé à tort pour critiquer des travaux portant sur les méthodes de résolution. De la même façon, on peut se demander si l'émergence de plateformes logicielles permettant d'utiliser un même algorithme pour plusieurs problèmes est légitime.

Pour répondre à ces interrogations, nous organisons cette partie de la manière suivante. Nous commencons par rappeler le principe du **NFL**, puis à partir de ses éléments nous discutons des limitations de son utilisation.

2.2.1 Les éléments du NFL

Le **NFL** utilise deux ensembles : l'ensemble des configurations $\mathcal{X}$ (aussi appelé espace de recherche) et l'ensemble $\mathcal{Y}$ qui est un ensemble ordonné des coûts possibles pour les configurations. On note $\prec$ la relation d'ordre et on dit que y est meilleur que y' si $y \prec y'$. À partir de ces deux ensembles, on définit un problème d'optimisation comme une application de $\mathcal{X}$ dans $\mathcal{Y}$. En d'autres termes, P est un problème d'optimisation défini sur $\mathcal{X}$ et $\mathcal{Y}$, si pour tout élément x de $\mathcal{X}$, $P(x)$ appartient à $\mathcal{Y}$; $P(x)$ est communément appelé le coût de x ou "fitness" de x. Il y a donc $|\mathcal{Y}|^{|\mathcal{X}|}$ problèmes différents pour un ensemble $\mathcal{X}$ et un ensemble $\mathcal{Y}$ donnés.

Maintenant, nous définissons le comportement d'une heuristique comme la succession des configurations visitées. Pour chaque configuration visitée, l'heuristique demande son évaluation à un "oracle". Puis, elle choisit une nouvelle configuration à visiter en fonction des informations dont elle dispose, c'est-à-dire en fonction des configurations munies de leur fitness. C'est donc la politique mise en œuvre par l'heuristique qui est observée. Sur un problème P ($\in \mathcal{Y}^{\mathcal{X}}$) donné, on note $(a_0, a_1, a_2, ...a_m...)$ la séquence des configurations visitées par une heuristique A.

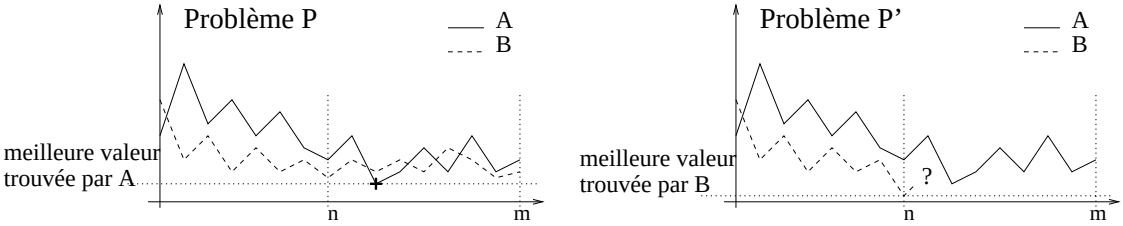
Il est maintenant question de comparer deux heuristiques A et B sur un problème P donné. On dit que A est plus efficace que B à l'itération m s'il n'existe pas de configuration présente dans la séquence de B avant le rang m, meilleure que la meilleure configuration présente dans la séquence de A avant le rang m :

$$\forall i \in [1, m], \exists j \in [1, m], P(a_j) \prec P(b_i)$$

NFL montre que l'efficacité de A sur P présente une contrepartie puisque il est moins efficace sur d'autres problèmes. Il est alors question de construire à partir de P, un problème P' de sorte que l'heuristique B soit plus efficace que l'algorithme A. Dans la pratique, seuls les points visités par A et par B nous intéressent pour construire P'. Afin de pouvoir prédire les points visités par A, on ne change pas leur fitness. Maintenant, on observe le premier point de la séquence de B qui n'appartient pas à la séquence de A et notons le b_n. Si une telle configuration n'existe pas, alors les deux séquences contiennent les mêmes éléments. Par conséquent, B n'est pas strictement moins performant que A. Donc P est déjà un problème où B est plus performant que A au rang m.

Dans le cas où b_n existe, il est alors possible d'attribuer une nouvelle valeur à cette configuration de sorte qu'elle soit meilleure que les configurations visitées par A. Pour forcer B à visiter b_n, on ne change pas le coût d'aucune configuration de rang inférieur à n. Pour finir, les autres configurations peuvent prendre n'importe quelle valeur ; on laisse donc par exemple le coût inchangé. On obtient donc les comportements souhaités : sur ce nouveau problème, A aura le même comportement de la première configuration visitée jusqu'à la $m^{ième}$ configuration, tandis que B aura le même comportement jusqu'à la visite de b_n. La suite de la séquence ne nous

apporte pas d'informations supplémentaires. La figure 2.5 illustre les comportement de A et B sur un telle construction. On a donc la meilleure configuration visitée par B qui est meilleure que toutes les configurations visitées par A. B est donc plus efficace que A sur P'.



Sur cette figure, on représente les exécutions des algorithmes A et B, en abscisse le temps (le numéro de la configuration dans la séquence), en ordonnée le coût (à minimiser) de la configuration. A est meilleur que B car aucun point de la séquence de B n'est inférieur au meilleur point de la séquence de A (marqué d'une croix). La construction du problème P' est réalisée en s'arrangeant par exemple pour que le diagramme de droite ressemble au diagramme de gauche sauf en un point. Ce point b_n a une valeur inférieure à tous les fitness des configurations explorées par A. La suite de la séquence explorée par B n'a pas d'importance.

FIG. 2.5 – Illustration de la construction du nouveau problème pour le **NFL**.

Il est important de signaler que nous n'avons présenté ici que le principe de la preuve. Le lecteur trouvera les détails dans [WM97]. De plus, Wolpert *et al.* montrent grâce au même type de raisonnement, que sur l'ensemble des problèmes d'optimisation de $\mathcal{X}$ sur $\mathcal{Y}$, toutes les heuristiques ont exactement la même efficacité. C'est-à-dire que le fait que A soit meilleur que B se produit "aussi souvent" que le contraire. Ils écartent ainsi la possibilité d'obtenir une heuristique plus efficace qu'une autre, en observant un éventuel "comportement moyen".

2.2.2 Analyse critique de NFL

Bien que le théorème du **NFL** soit vrai sur l'ensemble des problèmes d'optimisation, on constate que pour fonctionner il doit pouvoir changer le fitness d'une configuration sans modifier le fitness des autres. Ceci suppose que les fitness des configurations soient indépendantes les unes des autres. En d'autres termes, on peut voir l'oracle comme une gigantesque base de données de la taille de l'espace de recherche. Cette base est constituée d'une seule table qui associe à chaque configuration (utilisée comme une clef) un unique fitness. Cette table nécessite donc autant d'entrées qu'il y a de configurations dans l'espace de recherche. Cependant, en optimisation combinatoire cet espace de recherche est gigantesque. Une telle base

n'est donc jamais implémentée : le fitness d'une configuration est calculé à partir de données plus restreintes.

Culberson constate cette indépendance nécessaire entre les fitness des configurations pour arriver au résultat du **NFL** [Cul96]. Il met en évidence qu'on se situe rarement dans une situation aussi peu restrictive que celle du **NFL**. Il définit alors cinq scenarii explicitant le niveau de connaissance d'un algorithme :

- **Connaissance complète** - l'algorithme est developpé pour résoudre un problème spécifique. L'algorithme peut prendre en compte toutes les spécificités du problème. C'est ce qui est appelé "one shot problem" (ou problème dédié). Dans ce cas précis, on ne suggère pas la réutilisation de la méthode utilisée (dans son ensemble) pour d'autres problèmes.
- **Connaissance partielle** - l'algorithme sait comment est décrit le problème. Par exemple, il sait que c'est un problème de coloration de graphe dont le nombre de sommets est connu.
- **Connaissance faible** - l'algorithme sait que le problème appartient à une large classe de problèmes comme les problèmes NP-difficiles.
- **Presque sans connaissance** - l'algorithme sait seulement que la fonction objectif n'est pas trop lourde à calculer. Par la suite, Droste *et al.* [DJW99] identifient ce scénario à la complexité de Kolmogorov faible.
- **Sans connaissance** - l'algorithme ne connaît rien à l'exception du fait qu'il minimise une fonction.

Nous utilisons cette classification pour positionner certains résultats sur le théorème **NFL**.

En 1999, Droste *et al.* [DJW99] supposent que la complexité de Kolmogorov puisse être un bon argument pour bloquer **NFL**. En effet, la complexité de Kolmogorov simulerait le fait que la fonction fitness (bien qu'inconnue) soit calculée à partir d'éléments plus simples. Or trois ans plus tard, ils doivent se retracter puisqu'ils démontrent que cette notion n'est pas suffissante pour bloquer le résultat du **NFL** [DJW02]. En effet, la notion de complexité de Kolmogorov tolère les exceptions. En d'autres termes, il est possible de changer la valeur d'un certain nombre de points à partir d'une fonction fitness donnée, en ayant une croissance controlée de la complexité de Kolmogorov. En conséquence de quoi, Droste met en évidence avec le **ANFL** (Almost No Free Lunch) que le scenario "Presque sans Connaissance" ne se distingue pas du scenario "Sans Connaissance".

D'autres résultats présentent les limitations de l'utilisation du théorème. Whitley [Whi00] observe que si l'ensemble des problèmes étudiés est clos par permutation alors **NFL** peut être appliqué. On entend par clos par permutation la propriété suivante :
Soit F un ensemble de problèmes défini sur $\mathcal{X}$ et $\mathcal{Y}$, on dit que F est clos par permutation si :

$$\forall\ f\ \in F,\ \forall\ \sigma \text{ permutation de } \mathcal{X},\ f \circ \sigma \in F \tag{2.1}$$

Schumacher *et al.* [SVW01] montrent l'équivalence entre la fermeture de l'ensemble des problèmes étudiés et le fait que **NFL** soit applicable. On remarque que dans la preuve, les auteurs utilisent de manière forte l'hypothèse que les algorithmes d'optimisation considérés ne visitent qu'une seul fois une configuration. En d'autres termes, une heuristique est une politique pour parcourir la totalité de l'espace de recherche. Cependant on constate ici un moyen de sortir du cadre du **NFL**, en cherchant des classes de problèmes non-clos par permutation.

À ce sujet, Igel *et al.* [IT01] modèrent la portée de **NFL** par deux constats. Premièrement, les auteurs constatent que pour un espace de recherche $\mathcal{X}$ et un ensemble de coût $\mathcal{Y}$ donnés, le nombre d'ensembles de problèmes clos par permutation est extrêmement restreint par rapport au nombre d'ensembles de problèmes possibles : il existe un facteur "doublement exponentiel" entre ces deux nombres. Deuxièmement, les auteurs rapportent le fait que certaines classes de problèmes possèdent des opérateurs de voisinage spécifiques. Les interconnexions entre les opérateurs de voisinage et la fonction objectif sont succeptibles d'empêcher l'application du **NFL**. Nous reprendrons partiellement cette idée dans la suite de notre exposé.

De plus, Woodward *et al.* [WN03] pointent quatre limitations au **NFL** :

- **l'exploration multiple des points** - Whitley et Schumacher utilisent le fait qu'une heuristique ne visite qu'une seule fois les points de l'espace de recherche. Cette hypothèse n'est jamais mise en œuvre lors de la résolution de problème de grande taille.
- **toutes les fonctions** - Woodward *et al.* estiment aussi que l'ensemble des problèmes envisagés par **NFL** est trop vaste. En particulier, il n'est donc pas possible d'utiliser **NFL** sur des ensembles de problèmes plus restreints. Ajoutons que l'ensemble des problèmes ne doit pas être fermé par les permutations (cf formule 2.1).
- **le sur-coût de calcul** - **NFL** ne prend pas en compte d'éventuelles variations du temps entre les différentes soumissions à l'oracle ; en particulier si on tente d'éviter la redondance des configurations visitées.
- **heuristique contre métaheuristique** - les auteurs mettent en avant que **NFL** traite des algorithmes et non pas des stratégies. Les premiers sont Turing-complet, c'est à dire que la totalité du code est spécifiée ; les seconds sont des schémas génériques dont certains composants sont dépendants du problème. Par exemple pour le Hill Climbing, on peut utiliser l'opérateur 2-OP pour résoudre le problème du voyageur de commerce (TSP), ou l'opérateur swap pour résoudre le problème d'affectation quadratique (QAP). On est donc en présence de deux algorithmes pour une même stratégie.

En dépit de leurs mises en garde, les auteurs montrent aussi que **NFL** reste vrai si les algorithmes sont aussi munis de politiques pour s'arrêter et n'explore donc pas tout l'espace.

	ANFL	SCP
notion de structure utilisée	un problème $\mathcal{P}$ est structuré si $\exists$ un algorithme $\mathcal{A}$ compressant $\mathcal{P}$	une ensemble de problèmes $\mathcal{F}$ est structuré si $\exists$ un algorithme $\mathcal{A}$ tel que $\forall\ \mathcal{P} \in \mathcal{F}$, $\mathcal{A}$ compresse $\mathcal{P}$
remarque	l'algorithme peut être différent pour chaque problème	l'algorithme doit être le même pour les problèmes de la classe $\mathcal{F}$
résultat du **NFL**	le résultat du **NFL** est effectif	le résultat du **NFL** n'est pas effectif sur la classe des problèmes considérée

TAB. 2.2 – Notion de structure utilisé pour le ANFL contre notre notion de structure.

2.3 Structure de problèmes d'optimisation

Nous avons vu des restrictions à l'application du théorème **NFL**. Dans cette partie, nous commençons par proposer une définition souple de structure d'une classe de problèmes. Puis nous démontrons que les problèmes de k-coloration de graphe sont structurés. Nous constatons ensuite la difficulté d'extendre notre résultat à d'autres classes de problèmes par une simple réduction polynomiale. Pour finir, nous synthétisons nos apports par rapport au théorème **NFL**.

2.3.1 Notion de structure

Nous connaissons déjà les restrictions du théorème **NFL** utilisant la notion de "fermeture par permutation". Cependant, cette propriété n'est pas forcement facile à utiliser. Nous proposons donc une définition de structure d'un problème impliquant la non fermeture de la classe de problèmes étudiée. Bien que la définition de structure ne le suggère pas, nous utiliserons une notion d'opérateur de voisinage (suggéré par Igel *et al.* [IT01]) pour prouver qu'une classe verifie la notion de structure.

Définition 2.1 *Structure d'une classe de problèmes :*
On dit qu'une classe de problèmes est structurée, s'il existe un algorithme construisant un sous-ensemble strict de l'espace des configurations et tel que la connaissance du fitness de ces configurations permette de déduire le fitness de toutes les autres configurations.

En utilisant la classification de Culberson [Cul96], nous pouvons conclure : si une classe de problèmes donnée vérifie la définition de structure alors les heuristiques les résolvant peuvent passer d'une connaissance partielle à une connaissance complète du problème. Le tableau 2.2 résume les apports du théorème ANFL et de la notion de structure telle que nous l'avons définie.

2.3.2 Structure des problèmes de k-coloration de graphe

La notion de structure que nous avons formalisée n'est pas intéressante si aucune classe connue de problèmes ne vérifie cette définition. Or nous avons trouvé une classe de problèmes classiques d'optimisation qui la vérifie. Il s'agit du problème de 3-coloration de graphe (G3CP), dont nous rappelons ci-après la définition. Soit $G(V, E)$ un graphe non orienté, on appelle 3-coloration une application de V dans l'ensemble $\{1, 2, 3\}$. La fonction fitness est définie par :

$$f(c) = \sum_{i<j} \delta_{c(i)c(j)} \mathbb{I}_E(i, j) \tag{2.2}$$

où δ est le symbole de Kronecker :

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

et $\mathbb{I}_E$ est la fonction indicatrice de l'ensemble des arcs de G :

$$\mathbb{I}_E(i, j) = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{sinon} \end{cases}$$

Cette définition ne prend pas en compte d'éventuels arcs dont les deux extrémités seraient un même sommet.

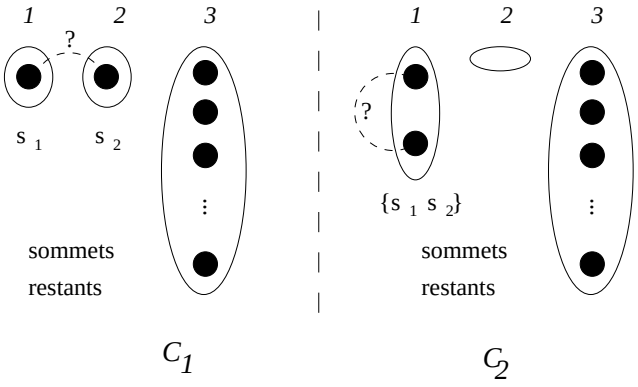
Théorème 1 SCP *: pour tout ensemble V, il existe un sous-ensemble $\mathcal{F}$ de l'espace de recherche $\mathcal{X}$, tel que $|\mathcal{F}| = |V| \times (|V| - 1)$ et pour tout graphe $G(V, E)$ et $G'(V, E')$ et leurs fonctions fitness associées f et f' respectent la relation suivante :*

$$\forall c \in \mathcal{F},\ f(c) = f'(c) \Longrightarrow \forall c \in \mathcal{X},\ f = f' \tag{2.3}$$

Afin de prouver ce théorème nous allons utiliser le lemme suivant :

Lemme 1 *Pour tout graphe $G(V, E)$, pour tout sommet s_1 et s_2, $(s_1 \neq s_2)$ l'observation du fitness de deux configurations bien choisies peut nous garantir la présence ou l'absence d'un arc a entre s_1 et s_2.*

Preuve 2.1 *du lemme 1* Soient s_1 et s_2 deux sommets de G. Notons C_1 et C_2 les deux colorations définies de la manière suivante : pour la coloration C_1, le sommet s_1 est coloré par 1, le sommet s_2 est coloré par la couleur 2, les sommets restants sont colorés par 3. Dans la coloration C_2, les sommets s_1 et s_2 sont cette fois-ci tous les deux colorés par la première couleur, les sommets restants étant toujours colorés par 3. La couleur 2 reste inutilisée dans la seconde coloration. On obtient évidemment : $f(C_2) - f(C_1) = 1$ si et seulement si les sommets s_1 et s_2 sont reliés par un arc (voir figure 2.6).



Les points noirs symbolisent les sommets du graphe, les ellipses rassemblent des sommets par couleurs. Dans la seconde coloration l'ellipse vide symbolise le fait que la deuxième couleur est inutilisée. La présence/absence d'un arc entre s_1 et s_2 est testée par ces deux colorations.
$f(C_1)$ = nombre d'arcs du sous-graphe de G engendré par $V\backslash\{s_1, s_2\}$ + nombre d'arcs du sous-graphe de G engendré par s_1 (=0) + nombre d'arcs du sous-graphe de G engendré par s_2 (=0).
$f(C_2)$ = nombre d'arcs du sous-graphe de G engendré par $V\backslash\{s_1, s_2\}$ + nombre d'arcs du sous-graphe de G engendré par $\{s_1, s_2\}$ (= 1 si $(s_1, s_2) \in E$; = 0 sinon).

FIG. 2.6 – Illustration du lemme 1.

Preuve 2.2 *du théorème SCP.*
Par l'application du lemme 1 sur les $\frac{N\times(N-1)}{2}$ paires de sommets différentes, on obtient C comme l'union des configurations ainsi fournies. D'où $|C| = 2\times\frac{N\times(N-1)}{2} = N \times (N-1) \ll 3^N$.

Nous avons donc prouvé que notre notion de structure peut être utilisée comme limitation au **NFL** et que le problème de 3-coloration vérifie la notion de structure telle que nous l'avons définie. De plus, le même type de résultat peut être obtenu sur les autres problèmes de k-coloration avec $k > 2$. En effet, les 3-colorations utilisées dans la preuve du lemme 1 peuvent être vues comme des k-colorations dont les couleurs supérieures à trois ne sont pas utilisées. Par conséquent, nous avons la preuve que tous ces problèmes sont structurés.

Nous savons malgré tout que les algorithmes de résolution du problème de k-coloration visitent rarement les configurations utilisées pour la propriété de structure. Cependant, d'autres ensembles de configurations peuvent tout aussi bien fixer la fonction fitness. Vis à vis de **NFL**, une fois que l'algorithme de résolution a visité un ensemble de configurations fixant la structure d'un problème, il n'est plus nécessaire de questionner l'oracle : l'algorithme peut connaitre le fitness d'une configuration par un calcul interne. On autorise alors une enumération complète de l'espace de recherche, sans aucune soumission à l'oracle. Une fois l'optimum trouvé, on ne

questionne l'oracle une dernière fois pour signifier l'obtention de l'optimum. Cette stratégie coûteuse est pourtant efficace selon les critères de **NFL**.

2.3.3 Essai d'extension du résultat

Dans la partie précédente, nous avons obtenu un résultat sur un problème d'optimisation NP-difficile. Pouvons-nous en déduire des informations supplémentaires sur d'autres problèmes NP-difficiles en utilisant par exemple des réductions polynomiales ? Dans cette partie, nous nous intéressons à la réduction du problème de 3-SAT en un problème de 3-coloration.

Nous reprenons la définition du problème de décision 3-SAT utilisé dans [Gir01]. Les données du problème sont les suivantes :

- un ensemble de N variables booléennes $p_1, p_2, ...p_N$.
- un ensemble de littéraux. Un littéral est une variable ($L = p_i$) ou la négation d'une variable ($L = \neg p_i$).
- un ensemble de M clauses distinctes $C_1, C_2, ...C_M$. Une clause est constituée de trois littéraux combinés par des *ou logique* ($\vee$).

Le but de ce problème est de déterminer s'il existe une affectation des N variables qui fait que la formule CNF (Conjunctive Normal Form) suivante soit satisfaite :

$$C_1 \wedge C_2 \wedge ... \wedge C_M$$

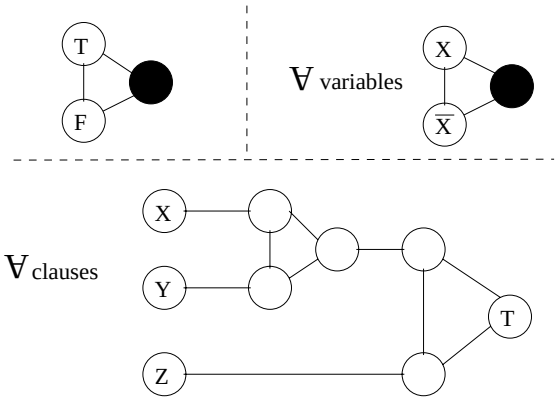
où $\wedge$ est l'opérateur *et logique*.

Le problème de 3-SAT peut être réduit en un problème de 3-coloration en construisant un graphe de $2 \times N + 5 \times M + 3$ sommets [Rob]. Dans un premier temps, on construit trois sommets particuliers (voir figure 2.7) :

- T - Ce sommet simulera la valeur "vrai". Tous les sommets de la même couleur que T seront considérés comme ayant reçu la valeur "vrai".
- F - Les sommets de la même couleur que F seront interprétés comme des sommets de valeur "faux".
- B - Les sommets dont la valeur est ni vrai ni faux ne représenteront pas les variables du problème original.

Dans un second temps, on construit pour chaque variable X deux sommets, qu'on note X et $\overline{X}$. Ces deux sommets sont connectés au sommet B et connectés l'un à l'autre. Ainsi dans une coloration valide les variables sont soit de la même couleur que T (vrai) soit de la même couleur que F (faux) et les sommets X et $\overline{X}$ ont des valeurs complémentaires. Puis pour chaque clause, on construit un sous graphe imposant qu'au moins un littéral soit vrai. La figure 2.7 illustre la construction des principales caractéristiques du graphe obtenu.

Nous observons que résoudre le G3CP fournit une solution pour le problème de 3-SAT. Il est possible de fournir pour toute configuration de 3-SAT une configura-



Cette figure représente la construction du graphe de G obtenu par la réduction polynomiale.
Dans une coloration valide, les sommets de G ont les mêmes valeurs que trois sommets spéciaux : T (vrai), F (faux), B (ni vrai ni faux). Les sommets X et $\overline{X}$ ne peuvent prendre seulement la valeur vrai ou faux, et leurs valeurs respectives doivent être "opposées". Une clause quelconque est représentée par un sous-graphe présenté dans la partie inférieure de la figure, où X, Y et Z représentent des littéraux.

FIG. 2.7 – Réduction polynomiale du 3-SAT en G3CP.

tion du problème de 3-coloration associé. Cependant, de nombreuses solutions sont possibles. Il faut donc lever l'ambiguité, par exemple en imposant un ordre sur les sommets.

Considérons maintenant le 3-SAT comme un problème d'optimisation et non plus comme un problème de décision. Dans la formulation classique, le problème d'optimisation associé consiste à maximiser le nombre de clauses satisfaites. Ce qui revient à minimiser le nombre de clauses insatisfaites. Nous utilisons cette formulation par souci de lisibilité, puisque le problème de coloration qu'on utilise pour la réduction est lui aussi un problème de minimisation. Une clause est dite non-statisfaite si et seulement si tous les littéraux sont "faux". Une telle affectation implique une unique violation de contrainte sur le sous-graphe de la clause correspondante. De ce fait, la meilleure affectation correspond à une coloration de même fitness : le nombre de clauses non-statisfaites dans la configuration du 3-SAT est égal au nombre de violations de contraintes dans la coloration correspondante. Néanmoins, certaines colorations n'ont pas d'antécédants par la réduction polynomiale. Par exemple, ceci se produit quand un littéral possède la même valeur que le noeud B. Par conséquent, il est possible que la coloration minimale trouvée ne soit pas l'image d'une configuration du 3-SAT. De plus, aucune des colorations utilisées pour le lemme 1 ne correspond à une affectation des variables du problème du 3-SAT. Ceci vient du fait que le **G3CP** obtenu est plus "flexible" que le 3-SAT originel. En effet, dans

3-SAT, X et $\overline{X}$ ne peuvent que prendre des valeurs booléennes complémentaires ; contrairement à ce qui se produit dans le G3CP où les valeurs T, F, B peuvent être théoriquement prises. En d'autres termes, réduire 3-SAT en G3CP enlève un certain nombre de contraintes.

2.3.4 Synthèse

En somme, la réduction polynomiale ne permet pas de transférer la propriété de structure d'un problème à un autre. Nous comprenons mieux pourquoi des études spécifiques sont généralement menées sur les classes de problèmes, et qu'il est inefficace d'utiliser la réduction pour résoudre un problème d'optimisation puisque la réduction polynomiale risque de diluer la structure du problème originel. Par ailleurs, il est nécessaire, pour une étude approfondie de 3-SAT, de prouver la présence de structure indépendamment des autres problèmes.

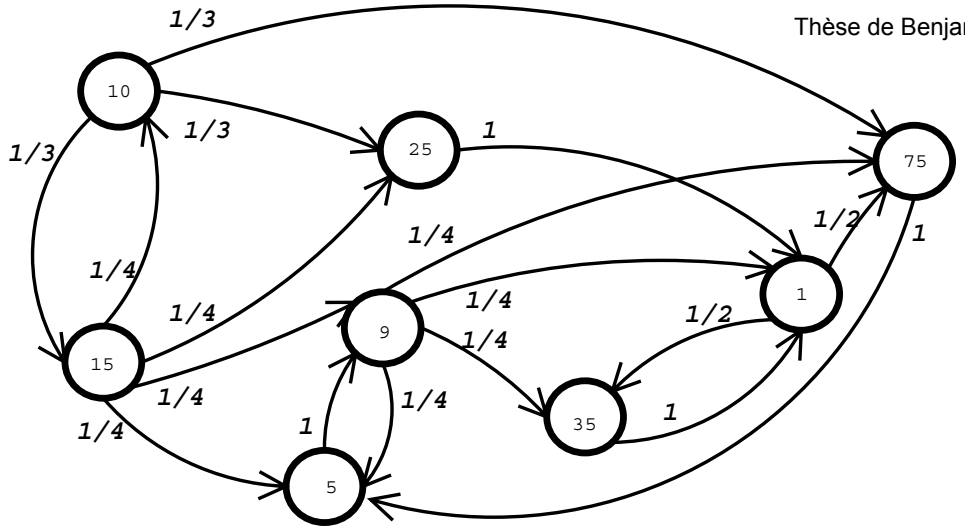
Nous constatons la connexion entre notre approche utilisant une notion de structure et celle de Igel *et al.* employant la notion du voisinage. C'est pourquoi, nous croyons qu'il peut être intéressant d'étudier les travaux sur la PLS-complétude pour la propagation de notre résultat à d'autres problèmes d'optimisation.

Remarquons que les plateformes logicielles comme **EO** (voir 5.2.1.1) ou EasyLocal++ [GS03] fournissent des modèles pour la conception d'algorithmes. Ces modèles sont des versions génériques de métaheuristiques classiques. Ceci conforte l'argument de Woodward [WN03] puisqu'on a affaire ici à du code incomplet. L'utilisateur ne doit spécifier qu'un certain nombre d'éléments tel que le codage d'une solution, la fonction d'évaluation ou les opérateurs qu'il souhaite utiliser. Une métaheuristique peut être alors efficace sur de nombreux problèmes à la redéfinition de ces composant près.

Finalement, nous pouvons conclure qu'il n'y pas de dominance des heuristiques (à cause du **NFL**) mais nous pouvons encore croire à l'efficacité de certaines stratégies (métaheuristiques) sur certaines classes de problèmes.

2.4 Modèle formel des paysages

De la même façon que les algorithmes évolutionnaires (algorithmes génétiques, colonies de fourmis ...) se sont inspirés de la biologie, la terminologie pour les décrire et les expliquer fût reprise de la biologie et on retrouve l'expression "fitness landscape" depuis 1932 [Wri32]. En optimisation, le terme de paysage (fitness lanscape) vient des travaux autour des algorithmes évolutionnaires. Le paysage explique avec des mots simples (vallée, plateau ...) les phénomènes complexes qui se produisent lors du déroulement de ces algorithmes. Maintenant cette terminologie s'est étendue aux autres heuristiques du moins aux heuristiques itératives (comme la recherche



Les arcs de valuation nulle sont supprimés du paysage pour allèger la représentation.

FIG. 2.8 – Exemple général de paysage.

taboue, le recuit simulé ...), basées sur l'amélioration successive de configurations complètes.

L'objectif de cette partie est de présenter des travaux effectués autour du paysage de fitness. Actuellement, il existe plusieurs modèles différents en rapport avec la notion de paysage.

Dans la suite de cette partie, nous utilisons les notations suivantes :
- $\mathcal{O}$: ensemble d'objets aussi appelés configurations.
- $\mathcal{R}$: ensemble des représentations des objets. Il n'y a pas forcément de bijection entre $\mathcal{O}$ et $\mathcal{R}$ (un objet peut avoir plusieurs représentations). En particulier, nous utiliserons dans la partie coloration une implémentation non bijective pour les configurations.
- f : la fonction objectif du problème d'optimisation, que l'on confondra pour l'instant avec le fitness d'une configuration.

Jones propose un modèle pour la notion de paysage [Jon95]. Un paysage est un graphe orienté $G(V, E, \widetilde{f}, \phi)$ avec $\widetilde{f}$ une application de V dans $\mathbb{R}$ et ϕ une application de E dans $[0, 1]$. La somme des valeurs des arcs sortant d'un même sommet est égale à un. La définition du paysage est indépendante de la notion de problème d'optimisation. la figure 2.8 présente un exemple de paysage construit sans rapport avec un problème d'optimisation quelconque.

Jones montre comment construire formellement un paysage à partir d'un opérateur. Par exemple, pour un opérateur OP d'arité quelconque (engendrant l nouvelles configurations à partir de k configurations) on définit une distribution de probabilités, à partir d'un graphe $G(V, E)$ de la façon suivante :

- $V = \mathcal{R}^k \bigcup \mathcal{R}^l$
- $E = \{(v_1, v_2) |$ la probabilité que OP engendre v_2 à partir de v_1 soit non nulle$\}$
- $\phi(v_1, v_2)$, la probabilité que OP engendre v_2 à partir de v_1.
- $\widetilde{f}$, pouvant être le maximum ou la moyenne (ou autre) des fitness du n-uplet.

Il en résulte qu'un paysage est lié à un opérateur. Ainsi les algorithmes manipulant plusieurs opérateurs, comme les algorithmes génétiques qui utilisent au moins une mutation et un crossover, explorent simultanément plusieurs paysages.

Jones met en avant le fait que la notion de paysage est indépendante de la notion de distance entre les configurations. Cependant, les informations apportées par ce modèle restent faiblement exploitable en particulier lorsque les heuristiques se compliquent, par exemple, grâce aux procédés d'hybridation.

La vision de Jones n'est pas la seule façon d'expliquer le comportement des heuristiques. Rudolph rassemble plusieurs travaux expliquant le comportement des heuristiques (et en particulier des algorithmes évolutionnaires) par des chaînes de Markov finies dans [Rud98]. Dans cet article, Rudolph rappelle des comportements asymptotiques des algorithmes évolutionnaires sous certaines conditions, tels que :

- la convergence presque sûre des algorithmes évolutionnaires en un temps fini ;
- un algorithme évolutionnaire visite infiniment souvent[1] l'optimum global.

Cependant, les modèles précédents sont soit lourds et difficiles à manipuler, soit difficiles à étendre en particulier au regard des procédés d'hybridation. Voyons maintenant l'approche, dont nous nous servirons. Cette version peut être vue comme une version simplifiée de la vision de Jones. Commençons par fixer les restrictions. Nous considérons seulement des problèmes combinatoires, dont l'espace de recherche $\mathcal{O}$ est fini. Ensuite, on définit sur $\mathcal{O}$ un opérateur de voisinage symétrique qu'on notera :

$$\begin{array}{rccc} Vois_{\mathcal{O}} : & \mathcal{O} & \longrightarrow & \mathcal{P}(\mathcal{O}) \\ & o & \longmapsto & Vois_{\mathcal{O}}(o) \end{array}$$

$$Vois \text{ est symétrique} \Longleftrightarrow (\forall o' \in \mathcal{O}, o' Vois(o) \Rightarrow o \in Vois(o'))$$

Dans la pratique, l'opérateur de voisinage sur les objets provient d'un opérateur de voisinage sur les représentants. Il se construit naturellemnt de la manière suivante :

$$Vois_{\mathcal{O}}(o) = \{o' \in \mathcal{O} | \exists r \in \mathcal{R} \text{ représentant } o \text{ et } \exists r' \text{ représentant } o', r' \in Vois_{\mathcal{R}}(r)\}$$

On met en place sur l'espace de recherche une structure de graphe valué $G(\mathcal{O}, E_{Vois_{\mathcal{O}}}, f)$ où $E_{Vois_{\mathcal{O}}} = \{(C_1, C_2) \in \mathcal{O}^2 | C_2 \in Vois_{\mathcal{O}}(C_1)\}$ et $f : \mathcal{O} \to \mathbb{R}$

[1] à condition de laisser tourner l'alogrithme un temps infini

Théorème 2 *G est connexe $\Longrightarrow$ il existe une métrique sur $\mathcal{O}$*

Preuve 2.3 *Soit G un paysage construit à partir de l'opérateur $Vois_{\mathcal{O}}$, soit d l'application définie par :*

$$\begin{array}{rcl} d : \mathcal{O} \times \mathcal{O} & \longrightarrow & \mathbb{N} \\ C_1, C_2 & \longmapsto & d(C_1, C_2) = |chemin_{minial}(C_1 \mapsto C_2)| \end{array}$$

Vérifions que d est une distance.

- *d bien définie pour tout x et y de $\mathcal{O}$ (connexe). L'ensemble des arêtes du graphe est fini (car il y a un nombre fini de sommets)* ✓
- $\forall x, y \in \mathcal{O}\ d(x,y) = 0 \Longleftrightarrow x = y$, *par définition* ✓
- $\forall x, y \in \mathcal{O}\ d(x,y) = d(y,x)$, *par définition, on sait que*

$$\begin{array}{rcll} d(x,y) & = & |chemin_{minimal}(x \mapsto y)| & \\ & = & |chemin_{minimal}(y \mapsto x)| & (car\ G\ est\ non\ orienté) \\ & = & d(y,x) & \end{array}$$

✓

- $\forall x, y, z \in \mathcal{O}\ d(x,z) \leq d(x,y) + d(y,z)$, *par définition :*

$$\begin{array}{rcl} d(x,y) + d(y,z) & = & |chemin_{minimal}(x \mapsto y)| + |chemin_{minimal}(y \mapsto z)| \\ & = & |chemin(x \mapsto y \mapsto z)| \\ & \geq & |chemin_{minimal}(x \mapsto z)| \\ & \geq & d(x,z) \end{array}$$

✓

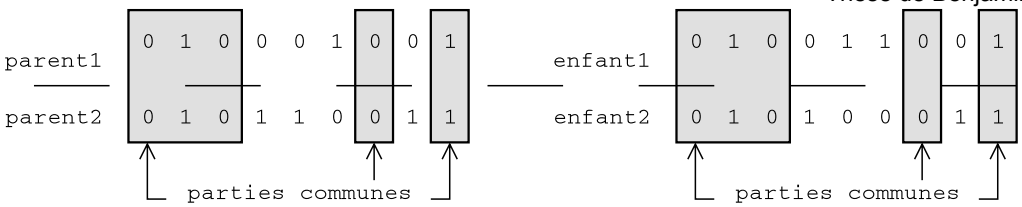
Remarquons que si G n'est pas connexe, cela signifie qu'on ne peut pas toujours passer d'une configuration à une autre par des mouvements élémentaires. On définit alors une mesure de $\tilde{d}$:

$$\tilde{d}(x,y) = \begin{cases} d(x,y) & \text{si } x \text{ et } y \text{ appartiennent à la même composante connexe} \\ +\infty & \text{sinon} \end{cases}$$

En utilisant les règles de calcul habituelles sur l'infini en théorie de la mesure sur $\overline{\mathbb{R}^+}$, cette notion vérifie les propriétés de réflexivité, symétrie et d'inégalité triangulaire. Rappelons ces règles :

$$\begin{array}{rl} \forall a, & a + \infty = \infty \\ \forall a > 0, & a \times \infty = \infty \\ & 0 \times \infty = 0 \end{array}$$

Par ailleurs, le calcul (exact) de cette distance n'est pas toujours possible en un temps raisonnable. Dans ce cas, on peut malgré tout se contenter d'une estimation, pour aboutir à une notion intuitive de proche ou éloigné. Revenons maintenant aux algorithmes évolutionnaires. Dans le travail de Jones, tous les opérateurs utilisés dans



Dans l'exemple de la figure, on dispose de la distance de Hamming sur les individus. La distance maximale entre deux individus est de 9 ; ici $d(parent_1, parent_2)$ *vaut* 4. *La partie grisée met en évidence les parties communes (de taille 5 = 9-4). Ces parties vont nécessairement apparaître chez les enfants. Pour le crossover uniforme, on a les propriétés suivantes :*

$$
\begin{aligned}
&d(enfant_i, parent_j) \leq d(parent_1, parent_2) (= \ 4),\\
&d(enfant_1, enfant_2) = d(parent_1, parent_2),\\
&d(parent_1, enfant_1) = d(enfant_2, parent_2) \ (= \ 1),\\
&d(parent_1, enfant_2) = d(enfant_1, parent_2) \ (= \ 3) \ et\\
&d(parent_1, enfant_1) + d(enfant_1, parent_2) = d(parent_1, parent_2)
\end{aligned}
$$

Notons que les enfants ont plus de chances d'être proches du "milieu" que d'être proches des parents.

FIG. 2.9 – Définition de voisinage par un opérateur de crossover.

une heuristique possèdent leur propre paysage. Dans notre approche, les opérateurs définissent des déplacements différents dans un même paysage. En particulier, les opérateurs de recombinaison permettent, dans une certaine mesure, de conserver cette notion intuitive de proche : la recombinaison de deux parents proches engendre des enfants proches. La figure 2.9 représente le cas d'un crossover uniforme pour des individus codés par chaînes de bits.

Cependant, il est important de remarquer que l'un des points les plus importants dans la conception d'algorithmes évolutionnaires est la définition d'un crossover efficace. Par exemple, de nombreux opérateurs ont été imaginés pour les configurations sous forme de permutations, dont les plus connues sont :

- **partially mapped crossover** (PMX) [GL85]. Ce crossover est similaire à un crossover deux points. Une tranche est copiée des parents vers les enfants, puis les enfants héritent des éléments manquants provenant de l'autre parent. Si l'élément qu'on souhaite affecter est déjà présent dans la tranche, alors on affecte l'élément de même indice dans la tranche de l'autre enfant.
- **cycle crossover** (CX), [OSH87]. Ce crossover utilise la construction de cycles comme principe de base. Pour construire le premier enfant, on copie des éléments du parent 1. À chaque copie d'un élément, on regarde l'élément de même indice chez le parent 2, cet élément nous donne l'indice du prochain élément à

copier. Ce processus est initialisé par la copie du premier élément du parent 1 et se termine quand un cycle est effectué. La complétion de l'enfant est faite par les éléments du parent 2. L'autre enfant est construit par la même méthode en inversant les rôles des deux parents.

- **order crossover** (OX) [Dav85]. Ce crossover commence comme PMX par une copie de tranche, puis on complète les enfants par un parcours des éléments dans l'ordre de l'autre parent.
- **edge recombinaison** (ER) [WSF89]. Cet opérateur fût conçu à partir de la constatation que dans le problème du TSP, les caractères des parents résident dans les arcs. Les enfants doivent le plus possible conserver les arcs intéressants. Cette méthode consiste à construire le (ou les) enfant(s) en ajoutant les villes, une à une : la ville suivante est l'une des villes qui suit ou précède la ville courante dans les tournées parents. L'algorithme est initialisé par la "première" ville d'un parent. Les choix privilégient les villes avec un faible nombre de voisins. Bien que cette méthode puisse construire des tournées incomplètes, le cas se présente peu souvent.

Le lecteur trouvera plus d'informations sur la conception de ces croisements et d'autres et en particulier sur des croisements opérant en utilisant des matrices dans [Mic96].

Un objectif de cette thèse est de mettre en évidence la présence de structures dans certains types de problèmes, ainsi que de présenter des moyens de les exploiter. Pour cela, il serait intéressant d'utiliser cette notion de ressemblance (ou différence) entre deux points de l'espace de recherche.

2.5 Application des études de paysage

La notion de paysage proposée nous permet d'utiliser des outils mathématiques (statistiques) pour analyser le paysage d'un problème spécifique. Un certain nombre de travaux furent effectués sur l'étude de paysages et en particulier sur le paysage du problème d'affectation quadratique. Dans une première partie, nous présentons de nombreux indicateurs, dont la quasi totalité sont des indicateurs statistiques.

2.5.1 Présentation des mesures

Le diamètre de l'espace de recherche est la distance maximale entre deux points de l'espace de recherche. Cette notion est directement déduite par la notion de distance. Elle dépend donc de l'opérateur de voisinage et du paysage (graphe) qu'il induit sur l'espace de recherche. Notons que cette première mesure est correlée à un opérateur donc à une éventuelle heuristique : Un autre opérateur [2] engendre un

[2] donc une autre heuristique.

autre paysage donc un autre diamètre pour l'espace de recherche. Cette remarque sera vraie pour tous les indicateurs suivants.

La seconde mesure est le **diamètre moyen**. Cet indicateur mesure la distance moyenne entre deux configurations. Les configurations peuvent représenter soit l'espace en entier, soit un sous-ensemble particulier, comme par exemple l'ensemble des optima locaux. Rappellons que pour certains problèmes l'estimation de la distance entre deux configurations peut être difficile ou coûteuse.

Une source d'informations produit n événements aléatoires de probabilités respectives p_1, p_2 $...p_n$, avec $\sum_i p_i = 1$. On appelle **entropie** la fonction H définie par :

$$H(p_1, p_2, ...p_n) = -\sum_i p_i \times log_2\ p_i$$

Shannon suppose que plus un système est aléatoire moins il produit d'information. Ainsi, l'entropie H est maximale si tous les événements sont équiprobables, c'est-à-dire si nous avons $p_i = 1/n$ dans ce cas H vaut $log_2(n)$. Grâce à l'entropie on peut mesurer la dispersion d'un ensemble de points. Toutefois, il peut être délicat de mesurer l'entropie d'un ensemble de configurations. En effet, avant de calculer l'entropie il faut choisir d'observer des événements pertinants.

Le second point consiste à définir la rugosité d'un ensemble ou sous-ensemble de solutions. Pour cela, Weinberger [Wei90] introduit la fonction d'autocorrélation $\rho(v)$ définie par :

$$\rho(v) = 1 - \frac{moy_{s,t\in\mathcal{X},d(s,t)=v}\left[(f(s)-f(t))^2\right]}{moy_{s,t\in\mathcal{X}}\left[(f(s)-f(t))^2\right]}$$

où v est une distance pour laquelle on observe la rugosité, moy est la moyenne sur les éléments indiqués en indice, $\mathcal{X}$ l'espace de recherche, d est la distance utilisée sur l'espace de recherche et f la fonction objectif.

2.5.2 Restriction et utilisation

Les indicateurs sont liés aux ensembles de configurations considérées. On remarque qu'il est techniquement impossible d'explorer tous les points de certains de ces ensembles à cause de leur taille. Il peut par exemple s'agir même de la totalité de l'espace de recherche. On utilise à la place des échantillonages statistiques de ces ensembles. Il est important de remarquer que la génération des configurations peut présenter un biais. De plus, l'étude des paysages doit aider à la compréhension des heuristiques. C'est pourquoi nous pensons que notre attention doit effectivement porter sur les ensembles de configurations générés par les heuristiques.

Par ailleurs, les méthodes étudiées sont souvent des méthodes incrémentales. Elle modifie donc les solutions données en paramètre. Par conséquent, on peut observer la variation des indicateurs lors de l'application des heuristiques étudiées. Sur ce

principe, il est possible d'observer des informations supplémentaires. Nous présentons ci-après les notations pour les mesures déjà présentées ainsi que pour plusieurs autres mesures.

- Δ_{Ent} mesure la variation d'entropie. Si l'entropie diminue cela signifie que les solutions ont tendance à se rapprocher les unes des autres. Par conséquent, l'heuristique qui a fait évoluer la population regroupe les solutions dans un nombre restreint de vallées.
- Δ_{Dmm} mesure la variation du diamètre moyen. Cette mesure quantifie également l'éparpillement des solutions. Si le diamètre moyen décroît, cela signifie que l'heuristique converge vers un "massif central[3]", c'est-à-dire une zone de l'espace de recherche regroupant les meilleures solutions.
- Δ_{Amp} mesure la variation d'amplitude de fitness. On utilise les formules suivantes

$$Amp(P) = 100 \times \frac{|P| \times (\max_{s \in P} f(s) - \min_{s \in P} f(s))}{\sum_{s \in P} f(s)}$$

$$\Delta_{Amp} = \frac{Amp(U) - Amp(O)}{Amp(U)}$$

  où U et O sont respectivement les populations initiales et transformées par l'heuristique étudiée.
- *Gap* mesure l'écart moyen entre la valeur de l'optimum et la fitness des individus d'une population donnée. Cet indicateur montre la différence de qualité entre les solutions trouvées par l'algorithme étudié et la meilleure solution connue pour le problème.
- Lmm (U) (la longueur de marche moyenne) mesure le nombre de mouvements élémentaires pour passer d'une configuration U à un optimum en appliquant l'algorithme de Hill Climbing. Si cette mesure est grande[4] alors on en déduit que le choix local n'est pas un bon guide pour l'algorithme. Au contraire, si elle est petite alors l'espace risque d'être très rugueux, et donc difficile à optimiser.

2.6 Synthèse et point de vue

L'utilisation des indicateurs statistiques a pour objectif d'expliquer le succès ou l'échec des heuristiques. Grâce aux informations qu'ils nous fournissent, il est envisageable de pouvoir améliorer les stratégies, afin d'obtenir de meilleurs résultats. Nous croyons qu'il est possible d'automatiser la réactivité d'une heuristique en fonction des résultats statistiques obtenus.

Cependant, il est important de garder à l'esprit les limitations théoriques : il n'est pas possible d'envisager qu'on va générer "la panacée", c'est-à-dire un algorithme qui

[3]Les termes massif central et vallée sont utilisés indifféramment.

[4]grand par rapport au diamètre de l'espace de recherche.

résoudrait tous les problèmes. C'est la mise en garde faite par le théorème **NFL**. Cependant, il n'est pas question non plus de sombrer dans le défaitisme, puisque les problèmes d'optimisation que nous nous posons généralement ont de forte chances d'être structurés.

Bien que la réduction polynomiale ne nous a pas permis d'étendre la structure des problèmes de k-coloration à d'autres problèmes, nous pensons que la réduction PLS peut être un bonne piste pour propager la présence de structure à d'autres problèmes. En effet, il existe des similitudes entre notre notion de structure et la notion de voisinage et par conséquent une notion de réduction préservant la relation de voisinage peut s'avèrer intéressante.

Quoiqu'il en soit pour résoudre les problèmes d'optimisation, il ne faut pas négliger l'étude analytique du problème, ne serait-ce que pour mettre en œuvre les opérateurs adaptés. Dans le même ordre d'idée, les mesures telles que l'entropie ou la distance dépendent fortement du problème étudié.

Implémenter des mécanismes basés sur les mesures de paysage peut s'avérer coûteux en temps de calcul. Un choix des indicateurs devra peut-être être effectué. Il sera donc nécessaire de faire une étude pour déterminer les indicateurs les plus pertinents pour le problème étudié.

Chapitre 3

Problème de coloration de graphe : Tour d'horizon et nouveau cadre formel

Dans ce chapitre, nous faisons une présentation générale du problème de coloration de graphe : historique, application et état de l'art sur les méthodes heuristiques de résolution. De plus, nous présentons des résultats théoriques sur la symétrie du problème de coloration et nous proposons des moyens techniques permettant d'opérer malgré cette symétrie. Plus précisement, nous proposons une structure de données particulière pour coder un partitionnement. Nous proposons également une formalisation de l'espace de recherche comme un ensemble quotienté par une relation d'équivalence. Cette formalisation nous a permis de développer des outils algorithmiques pour opérer sur cet espace quotient. En particulier, nous avons développé un test d'égalité entre partitionnements en $O(n)$, avec n le nombre de sommets du graphe à colorer alors qu'il existe $k!$ façons différentes de coder le même partitionnement. Nous avons également revisité l'opérateur de voisinage élémentaire. Les résultats obtenus peuvent être appliqués à tous les problèmes de partitionnement (aussi appelés "grouping problems") : graph coloring, clustering, bin packing, etc. Une partie de ce travail a été publiée dans **On search space symmetry in partitioning problems**, *EvoCop, Coimbra, Portugal, Avril 2004.*

3.1 Généralités

Cette partie présente la coloration de graphe suivant trois points de vue. Tout d'abord, nous présentons le problème et son origine. Puis, la seconde partie rassemble quelques résultats théoriques intéressants. Enfin, la troisième partie présente des domaines d'application de ce problème.

3.1.1 Historique

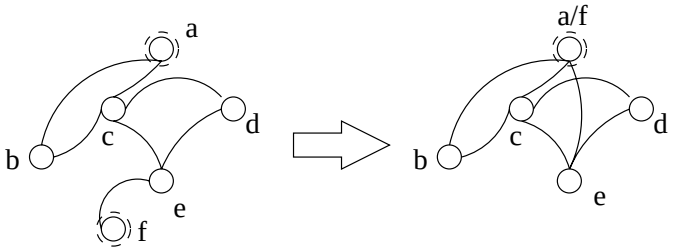
Le problème de coloration est un problème dont les origines sont anciennes : l'une de ses formes existe depuis le XIX^e siècle avec le problème de 4-coloration énoncé par le cartographe anglais Cayley [Err27]. En effet, celui-ci cherchait à savoir s'il était possible de colorer les pays sur une carte à l'aide de quatre couleurs de sorte que deux pays frontaliers soient de couleurs différentes. Ce problème de coloration peut en réalité être formulé comme un cas particulier du problème de coloration de graphe où les pays sont assimilés aux sommets d'un graphe et la relation de mitoyenneté géographique est simulée par un arc. On construit par ce procédé un graphe qui a la particularité d'être planaire, c'est-à-dire qu'il existe une représentation telle qu'aucun arc ne se croise. Or, il a été prouvé que quatre couleurs suffisent pour colorer tout graphe planaire [AH77]. Ceci répond donc à l'interrogation de Cayley après presque un siècle de recherche. Pour ce qui est de la preuve, Kempe l'avait jalonnée mais la quantité de calculs nécessaires[1], importante pour la fin du XIX^e siècle, avait forcé ce délai [Kem79].

Notons qu'il existe plusieurs variations autour du problème de coloration ; Jensen et Toft en présentent plus de deux cents [JT95]. Pour notre part nous en aborderons deux dans la partie théorique de cette thèse pour ne traiter que le premier dans la partie expérimentale :

- Le problème de coloration minimale (Minimal Graph Coloring Problem ou **MGCP**). Ce problème consiste à affecter des couleurs aux sommets d'un graphe de sorte qu'aucun sommet ne soit de la même couleur que l'un de ses voisins et que le nombre de couleurs utilisées soit minimal.
- Le problème de k-coloration (Graph k-Coloring Problem ou **GkCP**). Ce problème consiste à affecter l'une des k couleurs disponibles aux sommets de sorte que le nombre de sommets adjacents de la même couleur soit minimal.

Bien que le **GkCP** ne soit pas l'objectif premier de cette thèse, il est intéressant de le connaître car de nombreuses études [CS02, GH99, HH01b, MD00] cherchent à résoudre le **MGCP** en cherchant des solutions au **GkCP** et en faisant varier k. De plus, certains résultats que nous avons obtenus sur les codages en partitionnements d'ensembles sont aussi adaptés au **GkCP**. Ces brefs énoncés seront formalisés et détaillés dans la partie 3.2.

[1] il faut vérifier plus de 1200 configurations "élémentaires".



Ici, on identifie les sommets a et f du graphe. Le sommet a/f obtenu est connecté avec tous les voisins de a et f (i.e. les sommets b, c et e).

FIG. 3.1 – Exemple de contraction de sommets.

3.1.2 Quelques résultats théoriques

Les problèmes de colorations sont très largement étudiés dans la communauté de l'optimisation combinatoire. Jensen et Toft rassemblent un grand nombre de résultats théoriques sur ces problèmes dans [JT95]. En particulier, le **GkCP** est NP-difficile pour tout $k \geq 3$ [Kar72]. Ceci empêche la résolution exacte du problème dans le cas général.

De plus, le théorème de Lund est très pessimiste [LY93]. En effet, ce théorème prouve l'existence d'un ϵ strictement positif tel que s'il existe un algorithme polynomial pour colorer un graphe avec au plus $\chi(G) \times |V|^{\epsilon}$ couleurs (où $|V|$ est le nombre de sommets, $\chi(G)$ est le nombre minimum de couleurs) alors $P = NP$. En d'autres termes, si on trouve un algorithme polynomial pour colorer un graphe ayant un ratio de garantie indépendant du nombre de sommets alors on aura prouvé que $P = NP$.

Maffray *et al.* rapportent que dans le cas des graphes parfaitement contractiles, il est possible de colorer exactement un graphe [MT03]. Ces graphes se définissent par induction de la manière suivante : G est contractile s'il est complet ou s'il est possible d'identifier deux sommets pour donner un nouveau graphe parfaitement contractile. Cette identification consiste à remplacer les deux noeuds par un nouveau sommet qui a pour voisin l'union des voisins des sommets initiaux. Dans ce cas, l'algorithme de coloration attribue la même couleur aux paires de sommets identifiés (voir figure 3.1). Une telle identification est possible dans un graphe parfait[2] pour toute paire de sommets reliés par un chemin de longueur impaire sans corde.

3.1.3 Applications

Le problème de coloration est un problème important aussi bien d'un point de vue théorique que d'un point de vue pratique [Tri03]. En effet, ce problème apparaît dans de nombreux problèmes d'affectation de ressources :

[2]nombre chromatique de tout sous-graphe est égale à la taille d'une clique maximale.

- affectation de fréquences [Gam86]. Il s'agit d'attribuer des fréquences à des antennes, sachant que certaines antennes risquent d'interférer.
- conception d'emploi du temps (timetabling problem) [MEY95]. Ici, il s'agit de préparer des emplois du temps et des plannings de réservations de salles ainsi que de matériels.
- allocation de registres [CAC+81]. Dans ce problème, il est question d'attribuer les registres d'un ordinateur aux différentes variables d'un programme.
- conception de route aérienne [BB03].
- reconnaissance de formes [Oga86] ...

Dans tous les cas, le problème consiste à ranger des objets dans des classes disjointes, de manière à respecter des contraintes interdisant aux différentes classes de réunir certaines paires. Les sommets du graphe représentent les objets du problème et les arcs représentent les contraintes.

3.2 Modèle mathématique

Dans cette partie, nous donnons l'ensemble des définitions, notations et nos résultats théoriques sur le problème de coloration de graphe. $G(V, E)$ désignera un graphe non orienté qu'on supposera connu.

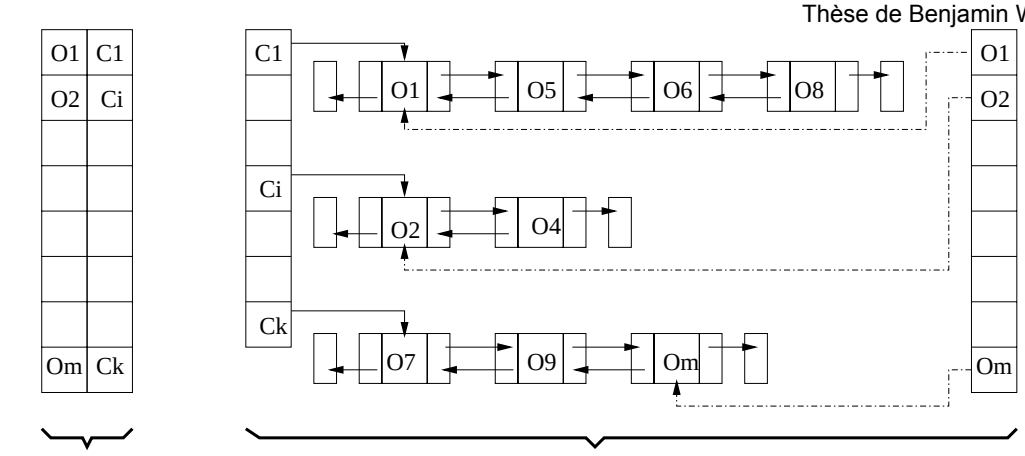
3.2.1 Définitions

Définition 3.1 *Coloration*
On appelle coloration $\mathcal{C}$, une application de V dans l'ensemble des entiers compris entre 1 et $|V|$.

Définition 3.2 *Couleur d'un sommet*
Pour une coloration donnée $\mathcal{C}$ et un sommet s, on appelle couleur de s (relativement à la coloration $\mathcal{C}$) l'image de s par $\mathcal{C}$ (i.e. $\mathcal{C}(s)$).

Définition 3.3 *Classe*
Pour une coloration donnée $\mathcal{C}$, on appelle classe l'ensemble de tous les sommets colorés d'une même couleur. Formellement les classes d'une coloration $\mathcal{C}$ sont données par l'application réciproque ensembliste $\mathcal{C}^{-1}$. On peut remarquer que cette définition autorise les classes vides : si un entier ne possède aucun antécédent, il engendre un ensemble vide par l'application réciproque.

D'un point de vue de l'implémentation, il existe deux méthodes pour coder les colorations. La première méthode revient à spécifier pour chaque élément le numéro du groupe auquel il appartient. Ce codage est appelé "group numbering". Il permet d'effectuer facilement une modification locale d'une variable en ne changeant qu'une



La coloration est codée “deux fois” de gauche à droite :
- un tableau indique le numéro de la couleur de chaque sommet.
- un tableau indique pour chaque couleur une liste chaînée donnant l'ensemble des sommets de cette classe
- un tableau fournit pour chaque sommet un pointeur vers “son” maillon dans l'une des listes.

FIG. 3.2 – Exemple de codage mixte.

seule valeur dans un tableau. La deuxième méthode consiste à décrire le contenu de chaque classe. Cette méthode permet d'exécuter un même traitement sur tous les sommets d'une même classe.

Pour notre part, nous utilisons un codage mixte (voir figure 3.2) permettant de modifier localement (sur une variable) la configuration en O(1) et de récupérer l'ensemble des éléments d'une classe facilement (i.e. pas de parcours de l'ensemble des sommets). Pour réaliser un tel codage il est nécessaire d'ajouter un tableau référençant les différents éléments des classes (à la droite de la figure 3.2).

Définition 3.4 *Violation de contrainte*
Pour une coloration donnée $\mathcal{C}$, on appelle violation de contrainte, l'existence d'un arc dont les deux extrémités sont de la même couleur. Formellement :

$$(v_1, v_2) \in E \text{ et } (\mathcal{C}(v_1) = \mathcal{C}(v_2))$$

On définit alors le nombre de violations de contraintes comme la fonction :

$$\begin{aligned} g : \{colorations\} &\rightarrow \mathbb{N} \\ \mathcal{C} &\mapsto g(\mathcal{C}) = \frac{1}{2} \times \sum_{i,j,\in V} \delta_{\mathcal{C}(i),\mathcal{C}(j)} \times \mathbb{I}_E(i,j) \end{aligned}$$

où $\delta_{i,j}$ est le symbole de Kronecker (i.e. $\delta_{i,j} = 1$ *si* $i = j; 0$ *sinon*) et $\mathbb{I}_E(i,j)$ est la fonction indicatrice de l'ensemble des arcs (i.e. $\mathbb{I}_E(i,j) = 1$ *si* $(i,j) \in E; 0$ *sinon*).

Définition 3.5 *Coloration valide*
On qualifie une coloration $\mathcal{C}$ de valide s'il n'existe aucune violation de contrainte (i.e $g(\mathcal{C}) = 0$).

Définition 3.6 *k-coloration*
Une k-coloration est une application de V dans l'ensemble des entiers de 1 à k (avec $k \leq |V|$). On suppose généralement qu'une telle coloration n'a pas de classe vide.

Définition 3.7 *Nombre chromatique*
Le nombre chromatique d'un graphe est le plus petit nombre k tel que le graphe admette une k-coloration valide. Pour un graphe G donné, on note $\chi(G)$ le nombre chromatique.

Définition 3.8 *Problème de décision*
Soit $G(V, E)$ un graphe non orienté ; soit k un entier. On cherche à savoir si G est *k-colorable*, c'est-à-dire s'il existe une k-coloration $\mathcal{C}$ valide sur G. Ce problème est NP-difficile pour tout k supérieur ou égal à 3 [JT95].
De ce problème de décision il découle deux problèmes d'optimisation présentés ci-après.

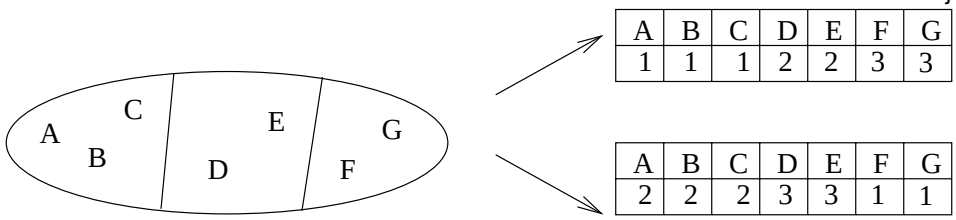
Définition 3.9 *Problème de k-coloration de graphe :* **GkCP**
Soit k un entier, on recherche une k-coloration $\mathcal{C}$ minimale pour le fonction g définie en 3.4.

Définition 3.10 *Problème de coloration minimale :* **MGCP**
On cherche une coloration valide utilisant le moins de couleurs possible. Le nombre de couleurs utilisées par une coloration est mesuré par la fonction suivante :

$$\begin{array}{rcl} f : \{colorations\ valides\} & \rightarrow & \mathbb{N} \\ \mathcal{C} & \mapsto & f(\mathcal{C}) = \max_{v \in V} \mathcal{C}(v) \end{array}$$

C'est la fonction que nous chercherons à minimiser dans notre étude expérimentale.

Nous utilisons pour cela la nomenclature suivante :
- Les colorations seront notées par des lettres calligraphiques $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ ...
- On note $\chi(\mathcal{A})$ le nombre de classes non vides. On construit généralement des colorations sans classes vides inférieures à $\chi(\mathcal{A})$. Ce qui revient à numéroter les couleurs de 1 à $\chi(\mathcal{A})$.
- $\mathcal{A}(u)$ note la couleur du sommet u.
- $\mathcal{A}^{-1}(i)$ désigne l'ensemble des sommets de couleurs i. $\mathcal{A}^{-1}\mathcal{A}(u)$ dénote l'ensemble des sommets de la même couleur que u (i.e. la classe de u).



A	B	C	D	E	F	G
1	1	1	2	2	3	3

A	B	C	D	E	F	G
2	2	2	3	3	1	1

Illustration sur un partitionnement de l'ensemble $\{A, B, C, D, E, F, G\}$. *À droite, deux applications, parmi les 6 (3 !) possibles, représentant le même partitionnement (de gauche).*

FIG. 3.3 – Dualité "partitionnement-application" des colorations.

3.2.2 Espace de recherche : espace quotient

Pour définir les colorations on utilise une application. Cette application attribue un numéro à chaque sommet, or l'information intéressante réside dans le fait que deux sommets ont (ou n'ont pas) la même couleur. En d'autres termes, on cherche plus un partitionnement de V qu'une application. Par ailleurs, un partitionnement peut être codé par des colorations différentes (voir figure 3.3).

Ce phénomène de symétrie engendre des difficultés lors de la résolution du problème. Nous présentons ici une formalisation de l'espace des configurations en un espace quotient. Cette formalisation demandera des outils spécifiques pour le calcul de distance qui seront présentés dans la section 4.2.

Cette partie se décompose en quatre points. Premièrement, nous établissons le cadre formel en définissant une relation d'équivalence. Puis nous présentons l'espace de recherche comme un quotient d'un ensemble par une relation d'équivalence. Les deux derniers points présentent deux opérations élémentaires dans l'espace de recherche, à savoir : le test d'égalité et un opérateur de voisinage élémentaire.

3.2.2.1 Relation d'équivalence

Deux colorations codent la même information si elles représentent le même partitionnement, c'est-à-dire que les couleurs de l'une sont un "renommage" des couleurs de l'autre. Formellement, cela signifie qu'il existe une bijection entre les ensembles d'étiquettes utilisées :

$$\exists\ \sigma : \{1...\chi(\mathcal{A})\} \rightarrow \{1...\chi(\mathcal{B})\}\ , \forall s \in V, \sigma(\mathcal{A}(s)) = \mathcal{B}(s)$$

Définition 3.11 *la relation d'équivalence* $\sim$
On dit qu'une coloration $\mathcal{A}$ est équivalente à une coloration $\mathcal{B}$ ($\mathcal{A} \sim \mathcal{B}$), s'il existe une bijection $\sigma : \{1...\chi(\mathcal{A})\} \rightarrow \{1...\chi(\mathcal{B})\}$, vérifiant pour tous les sommets s de V $\sigma(\mathcal{A}(s)) = \mathcal{B}(s)$.

Vérifions que σ est une relation d'équivalence. Elle doit pour cela être réflexive, symétrique et transitive.

- Réflexive :
  Soit $\mathcal{A}$ une coloration. On a trivialement $id(\mathcal{A}(s)) = \mathcal{A}(s)$ où id est l'identité sur $\{1...\chi(\mathcal{A})\}$. Comme id est une bijection, on en déduit que $\mathcal{A} \sim \mathcal{A}$.
- Symétrique :
  Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Supposons que $\mathcal{A} \sim \mathcal{B}$. Par conséquent, il existe σ tel que pour tout $s \in V$, on ait $\sigma(\mathcal{A}(s)) = \mathcal{B}(s)$. σ est une bijection donc σ^{-1} existe et est une bijection. On applique σ^{-1} à la définition de $\mathcal{A} \sim \mathcal{B}$, on obtient ainsi $\sigma^{-1}(\sigma(\mathcal{A}(s))) = \sigma^{-1}(\mathcal{B}(s))$. Par conséquent $\sigma^{-1}(\mathcal{B}(s)) = \mathcal{A}(s)$ donc $\mathcal{B} \sim \mathcal{A}$.
- Transitive :
  Soient $\mathcal{A}$, $\mathcal{B}$ et $\mathcal{C}$ trois colorations. Supposons que $\mathcal{A} \sim \mathcal{B}$ et $\mathcal{B} \sim \mathcal{C}$. On sait par définition, qu'il existe σ_1 tel que pour tout $s \in V$, on ait $\sigma_1(\mathcal{A}(s)) = \mathcal{B}(s)$ et σ_2 tel que pour tout $s \in V$, on ait $\sigma_2(\mathcal{B}(s)) = \mathcal{C}(s)$. D'où pour tout $s \in V$, on a $\sigma_2 \circ \sigma_1(\mathcal{A}(s)) = \sigma_2(\mathcal{B}(s)) = \mathcal{C}(s)$. σ_1 et σ_2 sont des bijections donc $\sigma_2 \circ \sigma_1$ est aussi une bijection (de réciproque $\sigma_1^{-1} \circ \sigma_2^{-1}$). On obtient donc $\mathcal{A} \sim \mathcal{C}$.

3.2.2.2 Quotient

À partir de la relation d'équivalence $\sim$, on construit des classes d'équivalence en quotientant l'ensemble des colorations par $\sim$. Un élément de la classe est alors appelé un représentant de la classe. La notation des classes se fait à partir d'un élément représentant : $\mathcal{A}_\sim$ signifie l'ensemble des colorations équivalentes à $\mathcal{A}$. On emploiera aussi le terme de partitionnement.

$$\{colorations\}_{/\sim} = \{\mathcal{A}_\sim | \mathcal{A} \in \{colorations\}\}$$

Définition 3.12 *raffinement / déraffinement*
Un partitionnement $\mathcal{A}_\sim$ est qualifié de raffinement d'un autre partitionnement $\mathcal{B}_\sim$ si toutes les parties du premier sont incluses dans les parties du second :

$$\forall i \in 1..\chi(\mathcal{A})\ \exists j \in 1..\chi(\mathcal{B}),\ \mathcal{A}^{-1}(i) \subset \mathcal{B}^{-1}(j)$$

Réciproquement le second partitionnement est qualifié de déraffinement du premier.

L'assimilation théorique des configurations équivalentes n'a d'intérêt que si on peut opérer sur les classes. Dans la pratique, il n'est pas possible de traiter des ensembles de colorations comme une seule configuration. En effet, il y a $\chi(\mathcal{A})!$ colorations équivalentes à la coloration $\mathcal{A}$. Il est dont nécessaire de manipuler les classes de manière plus économique. Il y a deux solutions communément utilisées :

- utiliser des représentants canoniques ;

– définir les opérateurs de façon à ce qu'ils soient indépendants du choix des représentants.

La première idée peut être réalisée de deux façons différentes. Jourdan dans [Jou03] propose un codage lors de la résolution d'un problème de clustering. Il code les configurations en numérotant les clusters de la façon suivante. Le premier cluster contient le premier objet. Le second contient le premier objet qui n'appartient pas au premier cluster et ainsi de suite on numérote tous les clusters. Cette méthode peut être appliquée à toute sorte de problème utilisant des partitionnements. Dans le problème qui nous concerne, les clusters sont les classes.

Marino *et al.* proposent une technique qui ne peut s'appliquer qu'au problème de coloration de graphe [MD00]. La technique utilisée ici consiste à attribuer de façon définitive des couleurs à des sommets références. En d'autres termes, colorer le sommet s dans la couleur i revient à dire que s est de la même couleur que le sommet de référence de la couleur i. Cependant, cette méthode n'est parfaitement applicable que si l'ensemble des sommets référence forme une clique (sous-graphe complet) or ce n'est pas toujours le cas. Dans ce travail, Marino *et al.* suggèrent donc de se contenter d'une clique de taille inférieure au nombre de couleurs recherché[3]. Cependant, le problème de clique maximale est aussi un problème NP-difficile. De plus, la taille de cette clique peut dans certains cas être strictement inférieure au nombre chromatique.

Bien que ces deux techniques soient pratiques pour tester l'égalité de deux partitionnements, elles sont lourdes pour les autres opérateurs. Notre choix s'est alors porté sur une définition d'opérateur invariant par rapport au choix du représentant. Premièrement, nous définissons un "test d'égalité" qui vérifie si deux colorations sont des représentants de la même classe. Puis, nous traiterons l'opérateur de voisinage élémentaire. Nous approfondirons ce principe dans la partie 4.2 traitant d'une métrique sur l'espace des partitionnements.

3.2.2.3 Test d'égalité

Il est important de pouvoir déterminer si deux colorations sont des représentants de la même classe. Pour cela, on peut utiliser une méthode de complexité $O(|V|)$ (voir algorithme 1). Cette méthode exploite le fait que le codage des colorations présente une partie regroupant les sommets par classes. Ainsi, on peut parcourir tous les nœuds une et une seule fois, en les groupant par couleur. Dans l'algorithme 1, si l'on omet le premier test alors le reste de l'algorithme retourne vrai si la coloration $\mathcal{A}_\sim$ est un raffinement de $\mathcal{B}_\sim$. Ce premier test vérifie l'égalité puisque si $\mathcal{A}$ et $\mathcal{B}$ utilisent le même nombre de couleurs ils sont égaux si et seulement si $\mathcal{A}$ (resp. $\mathcal{B}$) est un raffinement de $\mathcal{B}$ (resp. $\mathcal{A}$).

[3] il paraît évident que plus la clique est grande, plus elle est intéressante.

Algorithme 1 Test d'équivalence de deux colorations $\mathcal{A}$ et $\mathcal{B}$.

Require: $sigmaDeI$: ENTIER // candidat pour la $\sigma(i)$:
  // renommage de la classe i de $\mathcal{A}$ dans le système de numérotation de $\mathcal{B}$
  if $\chi(\mathcal{A}) \neq \chi(\mathcal{B})$ **then**
    retourne FAUX
  end if
  for $i \in 1..\chi(\mathcal{A})$ **do**
    soit $s \in \mathcal{A}^{-1}(i)$
    $sigmaDeI = \mathcal{B}(s)$
    for $t \in \mathcal{A}^{-1}(i) \backslash \{s\}$ **do**
      if $\mathcal{B}(t) \neq sigmaDeI$ **then**
        retourne FAUX
      end if
    end for
  end for
  retourne VRAI

3.2.2.4 Opérateur de voisinage élémentaire

L'opérateur de voisinage élémentaire pour un problème de partitionnement consiste tout simplement à changer un élément de groupe. Nous notons cet opérateur $changerClasse$; il nécessite trois paramètres : le partitionnement $\mathcal{A}_\sim$, le sommet s à modifier et la nouvelle classe qui va contenir le sommet s. On remarque que cet opérateur impose une contrainte à ces paramètres. En effet, le troisième paramètre doit être une classe des sommets qui soit une classe de la coloration.

L'objectif, ici, est de trouver un opérateur sur les colorations qui simule l'opérateur $changerClasse$. Nous proposons l'opérateur $changerCouleur$, qui utilise trois paramètres : la coloration $\mathcal{A}$, le sommet s à modifier et le numéro i de la nouvelle couleur, et qui change la couleur du sommet s en la couleur i. En d'autres termes, on souhaite vérifier que le diagramme suivant est commutatif :

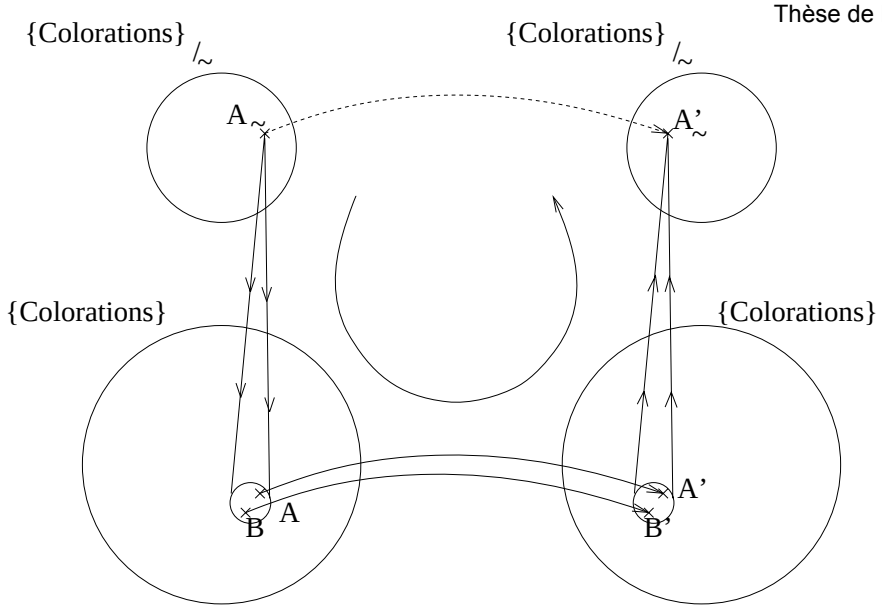
$$\begin{array}{ccc} \{colorations\}_{/\sim}, \quad V, \quad P(V) & \xrightarrow{changerClasse} & \{colorations\}_{/\sim} \\ \uparrow\!\downarrow & & \uparrow\!\downarrow \\ \{colorations\}, \quad V, \quad \mathbb{N} & \xrightarrow{changerCouleur} & \{colorations\} \end{array}$$

où les flèches "verticales" $\uparrow\!\downarrow$ représentent la surjection canonique, c'est-à-dire qu'à un élément $\mathcal{A}$ de $\{colorations\}$ on associe sa classe d'équivalence $\mathcal{A}_\sim$ dans $\{colorations\}_{/\sim}$.

La figure 3.4 explicite l'utilisation du diagramme commutatif.

Preuve 3.1 *de la commutativité du diagramme*

Soit $\mathcal{A}_\sim$ un partitionnement représenté par $\mathcal{A}_1$ et $\mathcal{A}_2$. Il existe une bijection σ telle que $\forall s \in V, \sigma(\mathcal{A}_1(s)) = \mathcal{A}_2(s)$. Soit v un sommet et i une couleur de $\mathcal{A}_1$. No-



Pour opérer sur un partitionnement $\mathcal{A}_\sim$ (en haut à gauche), on commence par trouver la classe d'équivalence des colorations le représentant (en bas à gauche). On choisit un représentant $\mathcal{A}$ quelconque de cette classe. On calcule le voisin $\mathcal{A}'$ de $\mathcal{A}$ en changeant la couleur d'un sommet. La classe d'équivalence de $\mathcal{A}'$ sont les représentants d'un partitionnement $\mathcal{A}'_\sim$. Affirmer que le diagramme est commutatif signifie que $\mathcal{A}'_\sim = changerClasse(\mathcal{A}_\sim)$, et ceci malgré la nécessité de faire un choix du représentant. En d'autre termes pour l'opérateur changerCouleur sur les colorations induit l'opérateur changerClasse sur les partitionnements.

FIG. 3.4 – Illustration de l'opérateur de voisinage dans $\{colorations\}_{/\sim}$.

tons $\mathcal{A}'_1$ (resp. $\mathcal{A}'_2$) la coloration obtenue par $\mathcal{A}'_1 = changerCouleur(\mathcal{A}_1, v, i)$. (resp. $\mathcal{A}'_2 = changerCouleur(\mathcal{A}_2, v, \sigma(i))$). Montrons que $\mathcal{A}'_1 \sim \mathcal{A}'_2$. Pour tout sommet $s \neq v$, $\sigma(\mathcal{A}'_1(s)) = \sigma(\mathcal{A}_1(s)) = \mathcal{A}_2(s) = \mathcal{A}'_2(s)$ et $\sigma(\mathcal{A}'_1)(v)) = \sigma(i) = \mathcal{A}'_2(v)$. σ est effectivement une bijection respectant la définition de l'équivalence des colorations $\mathcal{A}'_1$ et $\mathcal{A}'_2$. Le diagramme est donc commutatif.

Ce qu'on a vérifié sur *changerCouleur* sera vrai sur la plupart des opérateurs de voisinage. En effet, le système de numérotation des classes "reste le même" d'une coloration $\mathcal{A}$ à sa voisine $\mathcal{A}'$. La bijection σ permettant de passer de $\mathcal{A}$ à une coloration équivalente $\mathcal{B}$ sera la même que celle permettant de passer de $\mathcal{A}'$ à $\mathcal{B}'$ (où $\mathcal{B}'$ est la coloration voisine de $\mathcal{B}$).

3.3 Méthodes utilisées

Nous avons vu que le problème **MGCP** est NP-Difficile, il n'existe donc aucune méthode exacte connue pour le résoudre en un temps raisonnable. Cependant, dans de nombreux cas pratiques, on peut se contenter de résultats de bonne qualité. On utilise dans ce cas, comme pour de nombreux autres problèmes, des heuristiques qui fournissent de bonnes solutions approchées en un temps acceptable. Dans cette partie, nous présentons différentes méthodes d'optimisation approchées pour le problème de **MGCP**. Elles sont regroupées en quatre catégories ; les trois premières reprennent des schémas classiques d'algorithmes : méthodes constructives, à configuration unique et à configurations multiples. La dernière catégorie rassemble les méthodes que nous n'avons pas pu classer.

3.3.1 Méthodes constructives ou gloutonnes

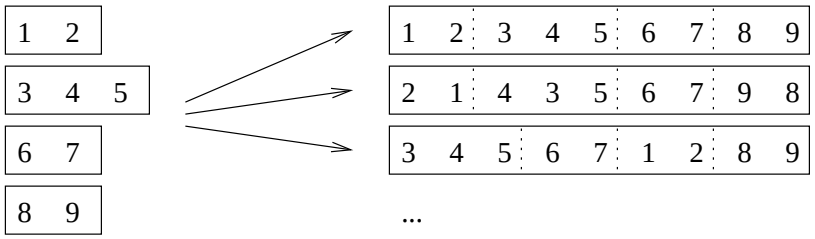
Les méthodes constructives construisent une configuration pas à pas. Pour la coloration de graphe, il existe plusieurs méthodes gloutonnes.

Peut-être la plus simple d'entre elles, la méthode "séquentielle", consiste à colorer les sommets dans un ordre arbitraire donné en leur affectant la plus petite couleur possible, c'est à dire la classe ne contenant aucun sommet adjacent au sommet courant. Si aucune couleur utilisée ne peut colorer le sommet courant alors on crée une nouvelle couleur pour colorer le sommet. Cette méthode s'est avérée expérimentalement peu efficace [JAMS91].

La méthode DSATUR [Bre79] choisit le sommet dynamiquement : le sommet courant est l'un des sommets qui est adjacent au plus grand nombre de sommets colorés différemment. En cas d'égalité, le sommet de degré le plus élevé est choisi prioritairement ; si l'égalité persiste le choix est alors aléatoire. Une fois que le sommet est choisi, il est coloré par la plus petite couleur admissible.

On retrouve également des méthodes dérivées de RLF (Recursive Largest First) [Lei79]. Il s'agit de construire les classes une à une à l'aide des sommets non colorés. Quand une classe est pleine, c'est-à-dire qu'aucun sommet non coloré ne peut prendre cette couleur sans provoquer de violations, alors on commence le remplissage de la classe suivante. Dans [Cul92], Culberson *et al.* commencent par discuter le choix de l'ordre utilisé pour remplir les couleurs (aléatoire, degré croissant, degré décroissant, combinaisons des ordres précédents ...). Ils constatent un phénomène intéressant avec le problème de coloration. En effet, de la même façon qu'il est possible d'induire une coloration avec une permutation, il est possible de construire avec une coloration des permutations de la façon suivante : on place les sommets en les regroupant par couleurs (voir figure 3.5). Formellement, la permutation σ vérifie $\forall i\ \forall j,\ \mathcal{C}(\sigma(i)) = \mathcal{C}(\sigma(j)) \Longrightarrow \forall k \in [i,j], \mathcal{C}(\sigma(k)) = \mathcal{C}(\sigma(j))$. On peut remarquer que l'algorithme glouton décrit ci-dessus fournit, en utilisant une telle permutation, une coloration $\mathcal{C}'$

n'utilisant pas plus de couleurs que la coloration $\mathcal{C}$ ayant engendré la permutation. En effet, quand une nouvelle couleur i est nécessaire pour colorer un sommet, on colore au moins tous les sommets de $\mathcal{C}^{-1}(\mathcal{C}(s))$ non encore colorés avec la couleur i, où s est le premier sommet à être coloré en i. Comme ce phénomène se produit de la première à la dernière couleur lors de la construction de $\mathcal{C}'$, on peut garantir que $\chi(\mathcal{C}') \leq \chi(\mathcal{C})$.



Pour la construction de la permutation, des choix interviennent sur l'ordre des couleurs et sur l'ordre des sommets dans une même couleur.

FIG. 3.5 – D'une coloration à des permutations intéressantes.

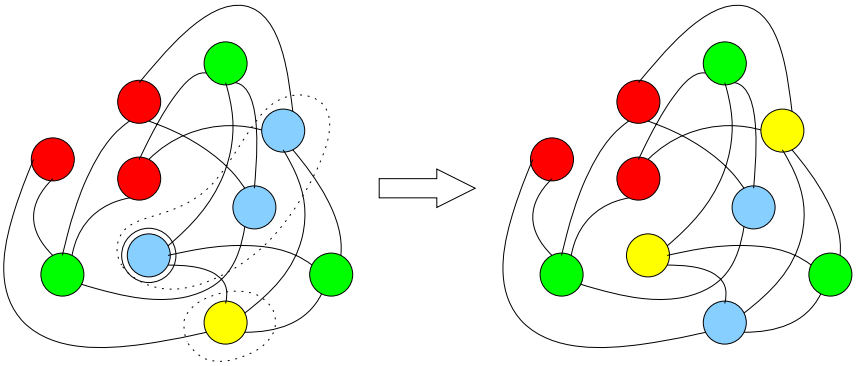
Certaines de ces méthodes se sont avérées relativement efficaces, en particulier les méthodes dérivées de RLF. Certains mécanismes seront réutilisés par d'autres heuristiques.

3.3.2 Méthodes itératives à configuration unique

Le problème de coloration de graphe est un problème très largement étudié. De nombreuses méthodes à solution unique lui furent appliquées. Comme ces techniques sont basées sur l'utilisation d'un opérateur de transformation unaire (opérateur de voisinage), nous traitons d'abord des différents opérateurs possibles :

- L'opérateur le plus simple "changerCouleur" consiste à changer la couleur d'un sommet du graphe. Comme le problème de coloration cherche à minimiser le nombre de couleurs, une version donnant des probabilités différentes aux tirages des différents voisins d'une coloration fut mis au point [JAMS91]. Cette version choisit d'abord la couleur puis un sommet de cette couleur. Ainsi, les sommets des petites classes ont plus de chance de changer de couleurs que ceux des classes de grande taille. Ceci favorise la diminution du nombre de couleurs. Ce voisinage peut être utilisé à la fois pour **GkCP** et **MGCP**. Cependant pour le second problème, la fonction d'évaluation (fitness) risque de ne pas être suffisamment discriminante. C'est pourquoi, on cherche généralement à résoudre plusieurs **GkCP** en faisant varier k lorsque cet opérateur est utilisé.
- Le second voisinage utilise les "chaînes de Kempe". Il a été utilisé exclusivement sur le **MGCP**. Ce voisinage échange des parties des couleurs de façon à garder des colorations valides et de ne pas augmenter le nombre de couleurs.

Pour une coloration $\mathcal{C}$, une chaîne H est définie par un triplet (v, i, j), où i et j sont des couleurs de $\mathcal{C}$ et v est un sommet de $\mathcal{C}^{-1}(i)$. La chaîne est alors l'ensemble des sommets qu'il est possible d'atteindre en partant de v et en prenant alternativement des sommets de $\mathcal{C}^{-1}(j)$ et de $\mathcal{C}^{-1}(i)$. Formellement, H est la composante connexe du sous-graphe induit par $\mathcal{C}^{-1}(i) \bigcup \mathcal{C}^{-1}(j)$ contenant le sommet v. Une fois cette composante connexe construite, il ne reste plus qu'à échanger les couleurs des sommets de la chaîne. On remarque qu'une couleur ne peut disparaître que si H est un singleton (i.e. $H = \mathcal{C}^{-1}(i) = \{v\}$). Le sommet v est alors coloré de la couleur j. Ce voisinage fut utilisé par Johnson *et al.* [JAMS91]. De plus pour faciliter la diminution du nombre des couleurs, ils minimisent la fonction $h(\mathcal{C}) = -\sum_i |\mathcal{C}^{-1}(i)|^2$. Cette fonction privilégie la construction de grandes couleurs (voir figure 3.6).

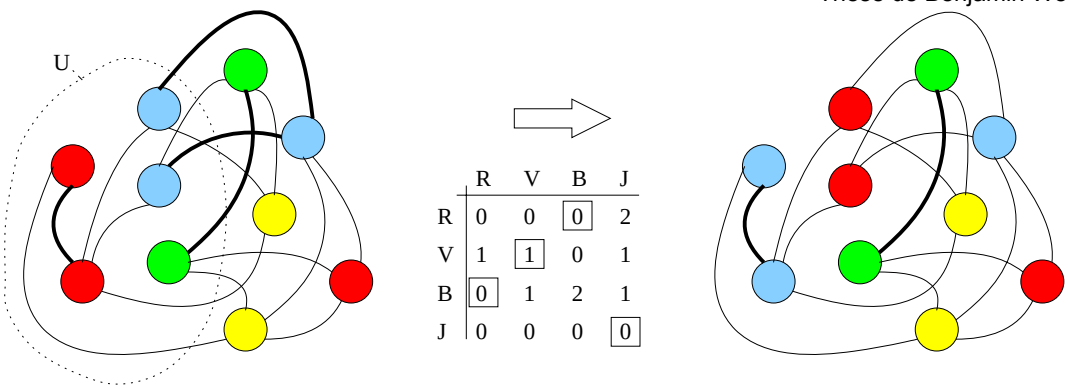


Ici le voisinage est défini par les classes jaune et bleu (entourées par les pointillés) et le sommet de la classe bleu entouré.

FIG. 3.6 – Exemple de voisinage par chaîne de Kempe.

– Le "voisinage de permutation" est de taille exponentielle [GPB98] et ne peut être utilisé que pour le **GkCP**. En effet, pour ce problème, il est possible de trouver un des voisins de meilleure qualité en un temps polynomial. Soit $\mathcal{C}$ une k-coloration. Soit U une partie quelconque de V. U (resp. son complément $\overline{U}$) induit un sous-graphe G_U (resp. $G_{\overline{U}}$), et une k-coloration $\mathcal{C}_U$ (resp. $\mathcal{C}_{\overline{U}}$). L'objectif est d'appareiller les couleurs de $\mathcal{C}_U$ et de $\mathcal{C}_{\overline{U}}$ de manière à induire le moins de violations de contraintes (voir figure 3.7). Ceci revient à résoudre un problème d'affectation linéaire dont les éléments $m_{i,j}$ de la matrice de données $m_{ij} = \sum_{\substack{u \in \mathcal{C}_U.i \\ v \in \mathcal{C}_{\overline{U}}^{-1}(j)}} \mathbb{I}_E(u, v)$. Le problème d'affectation linéaire est un problème classique en recherche opérationnelle. Il est résolu en $O(k^3)$ (la matrice est de taille $k \times k$) par l'algorithme Hongrois [CT80].

Appliquer les opérateurs de voisinage seuls peut conduire les algorithmes de recherche à des optima locaux. Pour la coloration de graphe, comme dans d'autres



Les violations de contraintes sont mises en évidence par des arcs épais. Le tableau au centre présente le nombre de violations de contraintes qu'entrainerait l'affectation d'une couleur de $\mathcal{C}_U$ sur une couleur de $\mathcal{C}_{\overline{U}}$.

FIG. 3.7 – Exemple de voisinage par permutation.

problèmes, des méthodes ont été utilisées pour quitter ces optima locaux. Nous présentons ci-après différents mécanismes de la littérature.

Johnson et al. [JAMS91] ont appliqué le recuit simulé (**SA**) au problème de coloration de graphe. En fait, Johnson *et al.* ont instancié le schéma de cette méthode de plusieurs manières.

1. Ils utilisent des colorations invalides au cours de l'algorithme. Pour guider le **SA**, ils mettent en place une fonction pénalisant les colorations avec de nombreuses violations de contraintes. Pour cet algorithme, le voisinage consiste à changer la couleur d'un sommet[4].
2. Dans la seconde réalisation, le **SA** utilise les chaînes de Kempe comme opérateur de voisinage et les configurations sont des colorations valides.
3. La dernière version décrite ici utilise la résolution successive de plusieurs **GkCP**. Cette méthode donne de très mauvais résultats avec le **SA**.

Nous avons relevé trois travaux particuliers sur la recherche taboue. Tous ces travaux traitent du **MGCP** en résolvant une succession de problèmes de **GkCP** et utilisent comme opérateur de voisinage le changement de couleur d'un sommet. Dans leurs travaux, Hertz *et al.* [HdW87] présentent les colorations comme des partitionnements. Un mouvement est caractérisé par un sommet et un numéro de couleur. En d'autres termes si un sommet s passe de la couleur i à la couleur j, s ne peut pas être recoloré en i avant lt itérations (où lt est la longueur de la liste taboue).

Dorne présente une façon économique en temps pour coder la liste taboue : il utilise une matrice $M_{nombre\ de\ sommets \times nombre\ de\ couleurs}$ [Dor98]. Pour une coloration

[4] un biais supplémentaire est fait pour favoriser la destruction des petites couleurs.

$\mathcal{C}$, un mouvement qui changerait la couleur du sommet s déclare tabou pour lt itérations le mouvement inverse en placant dans la case $m_{s,\mathcal{C}(s)}$ de la matrice la valeur numéro de l'itération courante plus lt. Ainsi tant que le numéro de l'itération courante est inférieur à $m_{s,\mathcal{C}(s)}$ le mouvement affectant à s son ancienne valeur est déclaré tabou.

Galinier *et al.* modifient légèrement l'opérateur de voisinage : ils utilisent une heuristique pour ne pas visiter des voisins inintéressants [GH99]. Pour cela, ils définissent la notion de sommets critiques. Pour une coloration donnée $\mathcal{C}$, les sommets critiques sont les sommets qui provoquent au moins une violation de contrainte. L'heuristique autorise seulement les changements de couleur des sommets critiques. En effet, le changement de couleur des autres sommets ne peut pas faire diminuer le valeur de $g(\mathcal{C})$.

Paquete *et al.* ont utilisé une recherche locale itérée pour colorer les graphes [PS02]. Cette procédure consiste à itérer une succession de recherches locales et de perturbations contrôlées. Pour cette méthode, les auteurs utilisent la stratégie consistant à résoudre plusieurs **GkCP**. Lors de la recherche locale l'opérateur de voisinage est *changerCouleur*. Dans cette étude, quatre méthodes de perturbations furent testées :

- **random move** qui change aléatoirement la couleur de certains sommets d'une coloration.
- **adding edge** qui ajoute temporairement des arcs au graphe.
- **conflict vertices** qui change aléatoirement la couleur de sommets en conflit.
- **direct diversification** qui est une recherche taboue utilisant une longue liste taboue.

Dans [VZ00], Vesel *et al.* proposent une recherche locale inspirée par Petford *et al.* [PW89]. Cette méthode construit la coloration voisine à partir de la courante en utilisant la technique suivante :

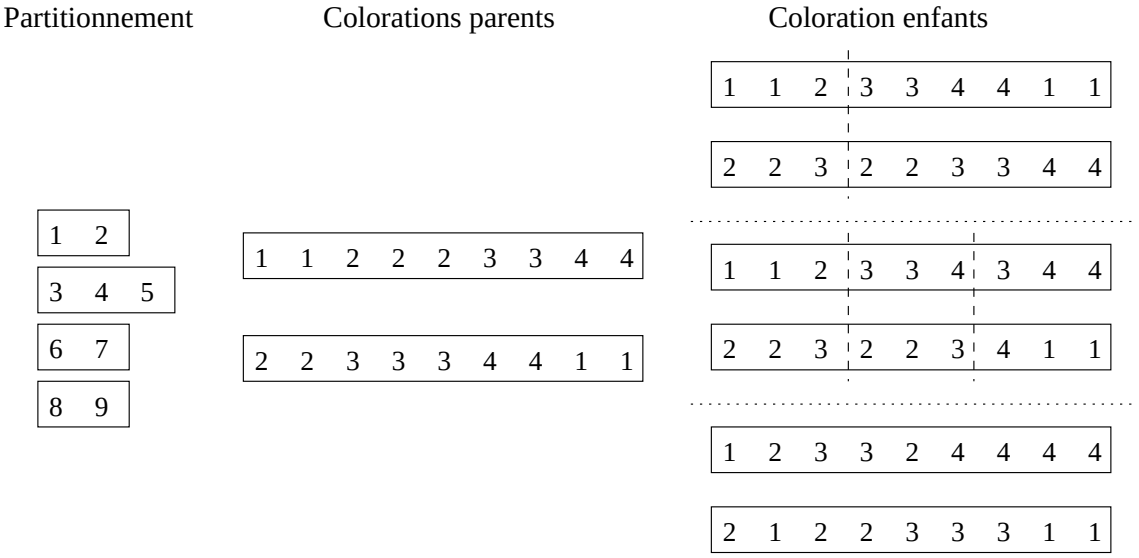
- On choisit un sommet v parmi les sommets en conflit.
- Puis on choisit une couleur de manière stochastique. La probabilité de choisir la couleur i est proportionnelle à $e^{S_i/T}$ où S_i est le nombre de violations de contraintes que la coloration de v par i engendrerait, et T est un paramètre de l'algorithme. Le paramètre T a le même comportement que la température dans un recuit simulé.

En conclusion, on constate que les méthodes de recherche locale ont fortement été utilisées pour résoudre le **MGCP**. On constate aussi qu'à partir des mécanismes présentés ici il est possible de réaliser de nombreuses combinaison ou variantes.

3.3.3 Méthodes itératives à configurations multiples

Des méthodes de résolution à configurations multiples ont aussi été fortement utilisées pour le **MGCP**. Or il survient, lors de l'instanciation de ce type de méthodes,

une difficulté que les méthodes précédemment exposées ne rencontrent pas. En effet, les algorithmes évolutionnaires (**AE**), contrairement aux recherches locales et aux méthodes constructives, gèrent plusieurs configurations. Du fait de la présence de symétrie dans l'espace de recherche (cf section 3.2.2), les opérateurs classiques sont peu efficaces. En particulier, on constate que l'utilisation d'un opérateur de croisement "classique" sur deux représentants différents d'un même partitionnement peut engendrer deux partitionnements différents entre eux mais aussi du partitionnement parent (voir figure 3.8). Dans ces conditions, il est difficile d'imaginer l'émergence de caractères intéressants au cours d'un algorithme utilisant ce genre d'opérateurs.



Cette figure présente trois exemples de croisements classiques sur un codage utilisant des tableaux. De haut en bas nous voyons :
- *le croisement un point effectuant un point de coupure ; ici entre la troisième et la quatrième position.*
- *le croisement deux points effectuant deux points de coupure ; ici le premier entre la troisième et la quatrième position, le second entre la sixième et septième position.*
- *le croisement uniforme choisissant aléatoirement entre la valeur du premier parent et du second.*

FIG. 3.8 – Illustration de croisements de deux colorations équivalentes.

Galinier *et al.* présentent un opérateur particulier pour la coloration de graphe [GH99]. Le GPX (Greedy Procedure Xover) traite les classes des colorations parents indépendamment du numéro utilisé. Pour cela, il sélectionne chez le premier parent la plus grande classe, colore de la même couleur tous les sommets de cet ensemble dans la coloration enfant, décolore tous ses sommets des deux parents puis recommence avec l'autre parent. Dans leurs algorithmes, Galinier *et al.* travaillent avec

$\underbrace{2\,1\,2\,2\,3\,1}_{\text{group number}}$: $\underbrace{2\,1\,3}_{\text{permut group}}$

À gauche "group number" fournit pour chaque objet le numéro de son groupe. À droite "permut group" réarrange l'ordre des parties.

FIG. 3.9 – Exemple de configuration pour un partitionnement.

une valeur de k fixée. C'est pourquoi ils bornent leur procédure pour créer au plus k couleurs chez l'enfant. Il est possible qu'à ce niveau certains sommets ne soient pas colorés dans la coloration enfant. Ces sommets restants sont alors colorés de manière aléatoire. Leurs expérimentations a montré que l'ajout de cette diversité était bénéfique à leur algorithme.

Eiben *et al.* proposent dans [EvdHvH98] une comparaison de différentes approches. En particulier, l'**AE** présenté ici est basé sur l'**AG** proposé par Falkenauer pour les problèmes de "grouping" : GGA [Fal98]. Eiben *et al.* rappellent les opérateurs du GGA à savoir : un opérateur de croisement, un opérateur de mutation, un opérateur d'inversion. Ces opérateurs travaillent sur un codage spécifique des partitionnements. Une partition est composée de deux parties : le "group number" et une permutation des noms des parties ("permut group" en abrégé) (voir figure 3.9).

- L'opérateur de croisement génère un unique enfant e à partir de deux parents notés p_1 et p_2. Pour commencer, e est une copie de p_2 puis on génère deux points de coupure dans *permut group* de p_1. Les sommets colorés dans p_1 à l'aide des couleurs comprises entre les points de coupures sont colorés de la même façon dans e. Dans la coloration e, on décolore tous les sommets d'une classe dont au moins un sommet a été recoloré, ils sont alors placés dans une liste de sommets non colorés. Pour finir, les sommets non colorés sont colorés grâce à une heuristique gloutonne.
- L'opérateur de mutation consiste à décolorer tous les sommets d'un certain nombre de classes. Ils sont ensuite recolorés par une heuristique gloutonne.
- L'opérateur d'inversion inverse une tranche dans le "permut group". Il revient à effectuer un 2-opt sur cette permutation.

Comme nous l'avons vu dans la section 3.2.2, Marino *et al.* propose une façon pour casser la symétrie de l'espace de recherche [MD00]. Par ailleurs les opérateurs ont été modifiés : La mutation a été remplacée par un opérateur qui change la couleur d'un sommet pour lui attribuer une couleur entraînant le moins de violations de contraintes possibles. Similairement, l'opérateur de croisement construit un enfant à partir de deux parents : si la couleur d'un sommet est identique chez les deux parents alors ce sommet sera coloré par cette couleur chez l'enfant ; sinon on choisit la couleur du parent qui fait le moins augmenter le nombre de violations de contraintes.

Une approche originale fut menée par Hurley *et al.* [HSV98]. En effet, ils utilisent un **AE** pour le problème d'affectation de fréquences. En fait, l'algorithme résout un

problème de coloration de graphe en codant les colorations par des permutations. À partir de cette permutation, ils utilisent un algorithme de décodage pour obtenir une coloration et ainsi obtenir la qualité de la permutation comme étant le nombre de couleurs de la coloration associée. L'algorithme utilisé est l'algorithme glouton séquentiel décrit dans la section 3.3.1. Nous savons par ailleurs qu'il existe effectivement au moins une permutation engendrant une coloration optimale par cet algorithme. Dans leur **AE** ils utilisent des opérateurs connus pour les permutations. Les auteurs ont testé les opérateurs de crossover PMX [GL85], OX [Dav85] et CX [OSH87] et c'est ce dernier qui s'est avéré le plus efficace.

Hamiez *et al.* proposent un algorithme de recherche dispersée ("scatter search") pour résoudre le problème de coloration [HH01b]. La recherche par dispersion autorise l'emploi d'opérateurs d'arités diverses en particulier supérieures à deux. Dans cette étude, Hamiez *et al.* développent une généralisation du GPX à une arité supérieure. Cet opérateur procède en k étapes (où k est le nombre de couleurs). A chaque étape i, il construit une nouvelle classe de la configuration enfant $\mathcal{E}$. Une étape se déroule de la manière suivante :

1. On choisit une configuration parent $\mathcal{P}$ (aléatoirement).
2. Dans $\mathcal{P}$, on décolore des sommets[5] pour obtenir une coloration partielle valide.
3. On choisit la plus grande classe possible de $\mathcal{P}$, que nous notons $\mathcal{P}^{-1}(j)$.
4. Pour tout sommet s de $\mathcal{P}^{-1}(j)$, si s n'est pas déjà coloré dans $\mathcal{E}$, on le colore avec la couleur i.
5. Pour tout sommet s de $\mathcal{P}^{-1}(j)$, on décolore s dans toutes les configurations parents.
6. Dans $\mathcal{P}$, on restitue la couleur des sommets décolorés à la ligne 2.

Les sommets non encore colorés à la fin des ces k étapes sont colorés de manière gloutonne afin de minimiser le nombre de violations de contraintes.

Costa *et al.* ont proposé un algorithme basé sur les colonies de fourmis permettant entre autre de colorer les graphes : ANTCOL [CH97]. La plus importante innovation de cette méthode réside dans l'utilisation de deux trainées de phéromones τ_1 et τ_2. Ils combinent ces valeurs avec des indications fournies par deux heuristiques η_1 et η_2. Dans leur étude, ils ont utilisé les algorithmes de DSATUR et RLF.

Fotakis *et al.* proposent dans [FLS01] une autre approche évolutionnaire. Cette méthode présente deux originalités. Premièrement, la métaheuristique utilisée est une approche à population fortement basée sur le **SA**. En effet, on applique un **SA** après toute modification d'une solution, afin de l'améliorer. La seconde originalité provient de son croisement Maximal Independant Set Crossover (MISC). Cet opérateurs ressemble à GPX à une nuance près : quand une classe est construite dans la coloration enfant à partir d'une classe d'un parent, elle est systématiquement com-

[5] on décolore le moins de sommets possible.

plétée par les sommets admissibles non encore colorés, en reprenant le principe de RLF.

Dans son étude, Clerc présente une résolution du problème de coloration de graphe par essaim particulaire OEP [Cle01]. L'originalité de ce travail réside dans le fait qu'il utilise à la fois des colorations valides (avec trop de couleurs) et des k-colorations (autorisant les violations de contraintes). Il a développé plusieurs opérateurs de confinement pour forcer le passage d'une coloration valide à une k-coloration. Dans son algorithme, le nombre de particules varie en cours d'exécution : une particule peut soit se dédoubler soit se supprimer le cas échéant. Pour finir, il conclue sur la nécessité de combiner les OEP avec des algorithmes de recherche locale efficaces.

3.3.4 Autres méthodes de résolution

Les métaheuristiques ne sont pas les seuls méthodes approchées pour s'attaquer au **MGCP**. Nous regroupons ci-après des méthodes dédiées au problème.

Walshaw *et al.* ont mené une étude sur la résolution du **MGCP** par une approche multiniveau [Wal01b, Wal01a, WE02]. Cette approche consiste à générer des problèmes de taille réduite puis de les résoudre, puis d'en déduire une solution pour le problème originel. Le critère d'arrêt pour la réduction de la taille du problème est la reconnaissance d'un graphe dont on connaît une coloration maximale, à savoir : un graphe complet, un graphe en étoile, une chaîne ou un anneau. Pour effectuer la réduction, les auteurs utilisent une procédure qui associe les sommets deux à deux. En d'autres termes, ils passent d'un graphe $G(V, E)$ à un graphe $G'(V', E')$ en construisant les éléments v de V' avec deux éléments s et t de V. Pour deux éléments v_1 et v_2 de V', (v_1, v_2) appartient à E' si (s_1, t_1) ou (s_2, t_2) appartient à V. Walshaw *et al.* tentent tant que c'est possible d'appareiller un sommet qui est voisin de l'un de ses voisins. Si aucun sommet ne peut être appareillé alors le sommet reste seul. Une fois qu'on obtient un graphe facile à colorer, on déduit des colorations sur le problème de taille supérieur. À chaque fois qu'une telle coloration est déduite, une heuristique est utilisée pour éventuellement l'améliorer. Les auteurs ont utilisé une recherche taboue de Hertz *et al.* [HdW87] et l'algorithme "iterated greedy" de Culberson [Cul92].

Barnier *et al.* proposent une méthode de résolution utilisant la programmation par contraintes [BB03]. Ils utilisent pour cela un précalcul d'un ensemble de cliques maximales (au sens de l'inclusion) comme des contraintes globales "tous distincts". Ce procédé permet de détecter les échecs plus rapidement. Par ailleurs, il calcule une borne inférieure grâce à un "branch and bound". Bien que cette borne soit NP-difficile à calculer, puisqu'il s'agit du calcul d'une clique maximale (cette fois-ci au sens de la taille), les auteurs obtiennent de bons résultats sur des problèmes structurés.

3.4 Conclusion

Le problème de coloration de graphe (**MGCP**) est un problème ancien. Bien qu'il soit prouvé comme étant NP-difficile, ses implications pratiques et théoriques en on fait un problème très étudié.

Nous constatons que de nombreuses méthodes dans des directions variées ont vu le jours pour résoudre le **MGCP**. De plus une même idée peut engendrer une multitude de variantes.

On constate aussi à la lecture des différentes études que certains problèmes sont plus ou moins durs pour les heuristiques présentées. Nous proposons donc l'étude de paysages comme un moyen d'analyse des instances du problème. De nombreux algorithmes approchés ont été implémentés pour le résoudre. Généralement ces méthodes reposent sur l'utilisation de plusieurs résolutions du **GkCP**. Entre chaque **GkCP**, des mécanismes existent pour garder le maximum d'information. Nous avons aussi constaté que les algorithmes à configurations multiples ont donné des résultats de bonne qualité. Pour cela, ils utilisent des opérateurs dédiés qui sont efficaces malgré la symétrie dans l'espace de recherche.

Par ailleurs, le **MGCP** présente une symétrie importante. Nous l'avons formalisé afin de mieux comprendre les difficultés qu'elle engendre. Cette formalisation respose sur la définition d'une relation d'équivalence entre les représentants d'un même partitionnement. Grâce à ce formalisme, nous avons revisité l'opérateur de voisinage élémentaire et nous avons développé un algorithme permettant de tester l'égalité entre partitionnements. Le test d'égalité présente une complexité linéaire grâce au codage mixte que nous avons présenté.

De plus, nous verrons dans le chapitre suivant que cette formalisation permet de prouver l'existence de métriques sur l'espace de recherche.

Chapitre 4

Paysages du problème de coloration de graphe

Dans ce chapitre, nous commençons par rappeler les travaux d'analyse de paysage dédiés au problème de coloration de graphe. Or, nous avons vu dans le chapitre 2 que de nombreux indicateurs statistiques nécessitent une estimation de la distance entre des configurations. Pour le **MGCP**, *le calcul de la distance est non trivial à cause de la symétrie du problème (voir chapitre 3). Nous avons défini la matrice de raffinement entre deux partitionnements. Cet outil formel nous a permis de définir deux métriques sur l'espace des partitionnements. Les distances sont calculables en temps polynomial. La première compte le nombre exact de mouvements élémentaires pour passer d'un partitionnement à un autre. La seconde est plus économique en temps de calcul et correlée avec la première. Elle présente par ailleurs un biais qui est intéressant pour le* **MGCP**. *Après cela nous présentons un moyen de calculer l'entropie d'une population de colorations. Pour finir, ces outils nous ont permis d'effectuer une classification des benchmarks classiques pour le problème de coloration de graphe.*
Une partie de ce travail a été publiée à dans **On search space symmetry in partitioning problems**, *EvoCop, Coimbra, Portugal, Avril 2004.*

4.1 Méthodes d'analyse de paysages dédiés à la coloration de graphe

L'étude des paysages des problèmes de coloration de graphe n'est pas une nouveauté. À ce sujet nous présentons deux études pré-existantes : la première basée sur la caractérisation de "parties gelées" ; la seconde utilisant un arbre d'analyse. Ces travaux sont fortement liés à ce problème. Bien qu'ils fournissent des informations adaptées au problème, il est malheureusement difficile d'étendre leur protocole d'étude à d'autres problèmes d'optimisation.

4.1.1 Parties gelées

Dans [CG01], Culberson *et al.* présentent une analyse des problèmes de coloration de graphe basée sur l'observation de paires de sommets. Pour un graphe donné, une paire peut être caractérisée de trois façons :

- **paires gelées** qui qualifient les paires dont les deux sommets sont de la même couleur dans toutes les k-colorations valides ;
- **paires libres** qui qualifient les paires dont les deux sommets sont de couleurs différentes dans toutes les k-colorations valides ;
- **paires effectives** qui qualifient les autres paires.

L'auteur explique en quoi ces notions sont corrélées avec la difficulté de trouver une coloration. Dans la pratique, si une paire est gelée, un algorithme de coloration détectera vite que les deux sommets doivent être de la même couleur. Ce qui par conséquent réduit l'espace de recherche. Dans une moindre mesure, les paires libres ont toujours leurs sommets de couleurs différentes donc peuvent entraîner une réduction de l'espace de recherche. Les paires effectives sont donc les paires qui posent le plus de difficultés pour un algorithme d'optimisation.

Pour leur étude, Culberson *et al.* construisent des graphes k-colorables pas à pas en ajoutant des arcs jusqu'à obtenir un graphe sans paires effectives. Ils appellent cette procédure un développement gelé complet, qu'ils utilisent pour analyser le problème de coloration de graphe.

Ils définissent aussi les notions de graphes effondrés et de graphes critiques. Les premiers correspondent à une réécriture d'un graphe en identifiant toutes les paires gelées à un unique sommet ; cette construction s'etend transitivement à tous les ensembles gelés. Quant au graphe critique, ils le définissent pour un nombre k donné comme un graphe qui n'est pas k-colorable mais qui le devient dès qu'on lui supprime un arc *bien choisi*.

Culberson *et al.* montrent empiriquement l'existence d'un saut du nombre de paires gelées et libres entre les graphes k-colorables et les graphes non k-colorables. Ceci met en évidence l'existence d'une "transition de phase" dans le **GkCP**, c'est-

à-dire une rupture brutale entre les problèmes faciles et difficiles. De plus, lors du développement gelé complet la diminution de taille du graphe effondré correspondant suit le même comportement. D'un point de vue de la difficulté d'un problème, les auteurs prouvent ainsi que les problèmes de coloration difficiles présentent de larges (sous-)graphes critiques.

Cette étude met en évidence dans le problème de coloration de graphe l'existence de problème plus ou moins facile à optimiser. Cependant, elle ne permet pas de déterminer si un nouveau problème donné, dont on ignore effectivement le nombre chromatique, est difficile ou facile à optimiser.

4.1.2 Arbre d'analyse

Hamiez *et al.* dans [HH01a] ont analysé les propriétés des solutions de problèmes de coloration. Cette étude a plus pour objectif de dégager les informations que les heuristiques peuvent découvrir lors de la résolution d'un problème donné, que de traiter tous les problèmes de coloration de graphe comme précédemment. En d'autres termes, il n'est pas question ici de savoir s'il existe des problèmes difficiles et où ils seraient situés mais bien de dire si dans un problème donné il n'y a pas d'informations à relever pour améliorer les algorithmes de recherche. De plus, l'étude précédente traite des graphes dont le nombre chromatique est connu, ce qui n'est pas en général le cas dans le **MGCP**.

Pratiquement, les auteurs relèvent les ensembles représentatifs, c'est-à-dire les ensembles de sommets qui se retrouvent fréquemment de la même couleur dans des colorations valides de bonne qualité[1]. Ils construisent pour cela ce qu'ils appellent une surface d'analyse. Cette structure de données complexe est basée sur un arbre binaire. Un noeud de cet arbre contient un sommet du graphe. Lorsqu'on parcourt une branche, un déplacement d'un noeud n vers le sous-arbre gauche signifie "n est avec les ensembles représentatifs du sous-arbre gauche". Un déplacement d'un noeud n vers le sous-arbre droit signifie "n et le sous-arbre droit forment des ensembles représentatifs différents". De plus, certains noeuds possèdent une information supplémentaire. Ils contiennent un entier indiquant le nombre de fois que l'ensemble de sommets représenté par le début de la branche apparaît dans l'ensemble des colorations étudiées (voir figure 4.1).

À partir de ce principe, deux méthodes d'analyse sont proposées : CRS (Complete Representative Set) et PRS (Partial Representative Set). Le premier traitant les classes de couleur sans regarder les éventuels sous-ensembles. La méthode PRS observe la présence de partie de classe commune aux colorations étudiées. Ces outils ont permis aux auteurs d'améliorer les résultats de leur algorithme évolutionnaire, grâce à une initialisation de leur algorithme par un algorithme glouton reposant sur

[1] Ces colorations sont construites de manière gloutonne.

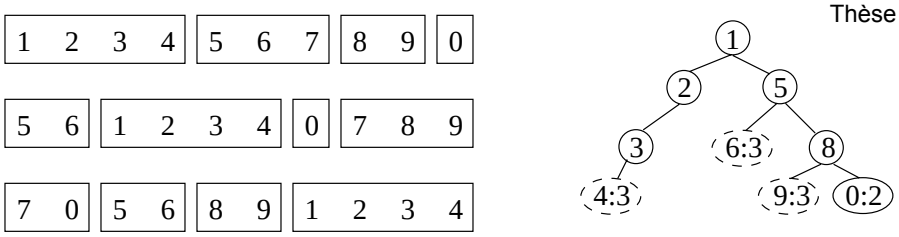


FIG. 4.1 – Exemple de surface d'analyse PRS.

leur analyse.

L'algorithme glouton, utilisé par les auteurs, construit les colorations en deux grandes étapes : Premièrement, k classes sont construites en suivant la méthode RLF (cf 3.3.1). Pour cela, les ensembles représentatifs sont triés par ordre de fréquence. Lorsque les k classes sont construites, les sommets restants sont affectés de manière gloutonne afin de minimiser le nombre de violations de contraintes.

Cette étude est très spécifique au problème de coloration de graphe. Elle met en œuvre une structure de données lourde qui est difficile à transposer pour d'autres problèmes.

4.2 Mesures de distance dans l'espace opérationnel

Nous avons vu que pour l'étude des paysages, il est souvent intéressant de posséder une métrique sur l'espace opérationnel corrélée avec les opérateurs. Or, l'absence de distance pour certains problèmes a conduit à des modélisations complexes de la notion de paysage. Dans cette partie, nous proposons deux distances pour l'espace des colorations aboutissant toutes deux à la reconnaissance de schémas communs et faciles. De plus, nous montrons que les calculs nécessaires pour ces distances peuvent être effectués en temps polynomial.

Nous présentons d'abord des généralités sur les distances pour le problème de coloration. Puis, nous présentons la première distance directement extraite de la construction de l'espace de recherche. Dans une troisième partie, nous présentons une notion de distance corrélée avec la première et plus facile à calculer. De plus, nous verrons que cette dernière distance présente un intérêt supplémentaire pour la coloration de graphe, puisqu'elle favorise la reconnaissance de schémas communs avec peu de couleurs.

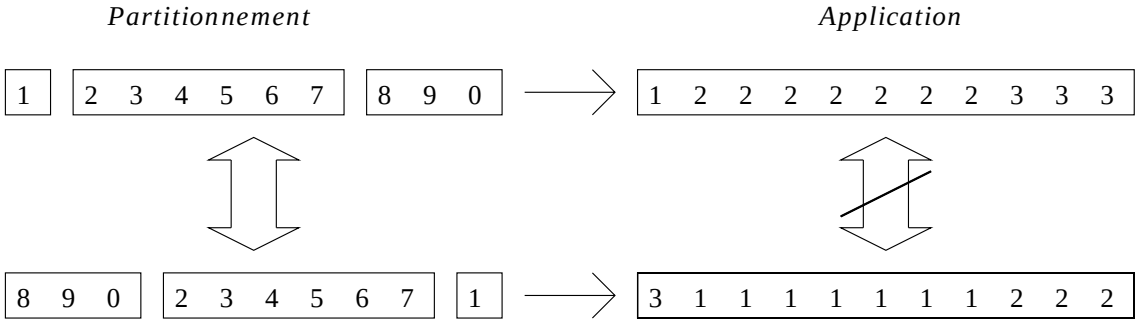
4.2.1 Généralités

Nous avons vu dans la section 2.4 que sous certaines conditions, l'espace de recherche peut être muni d'une notion de distance par l'utilisation d'un opérateur

de voisinage. Cependant, la preuve de l'existence de cette distance ne nous donne pas le moyen de la calculer efficacement. D'ailleurs, dans le problème de coloration de graphe, il est difficile d'établir une distance intéressante. En effet, elle doit prendre en compte la symétrie de l'espace de recherche provenant du codage des colorations (cf 3.2.1). En effet, si une coloration est représentée par une application, il est possible de mesurer une distance, en comptant par exemple le nombre de différences entre ces deux applications. On obtient alors la formule suivante[2] :

$$d_n(\mathcal{A}, \mathcal{B}) = \sum_{s \in V} \delta_{\mathcal{A}(s),\mathcal{B}(s)} \tag{4.1}$$

Or, nous savons maintenant que la représentation de la coloration n'est pas unique et deux applications peuvent coder le même partitionnement. On constate aussi qu'il est même possible en utilisant la distance d_n d'attribuer une distance maximale, en l'occurrence le nombre de sommets du graphe, entre deux colorations équivalentes (voir figure 4.2).



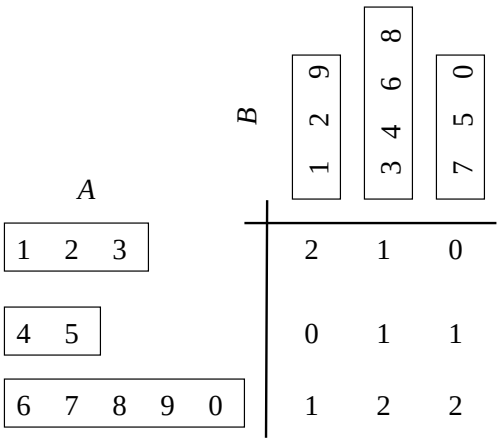
À gauche de la figure, les sommets du graphe sont répartis dans trois ensembles. À droite, les colorations sont codées de la manière suivante : on indique pour chaque sommet le numéro de l'ensemble auquel il appartient. La distance naïve, d_n, nous indique que ces colorations sont très différentes alors qu'elles sont équivalentes.

FIG. 4.2 – Problème de la distance naïve.

Afin de résoudre ce problème, nous proposons et discutons dans cette partie deux distances d_ρ et d_σ reposant toutes deux sur l'identification des "couleurs communes" aux deux colorations. Pour cela, on commence par construire une matrice $\mathcal{M}_{\chi(\mathcal{A})\times\chi(\mathcal{B})}$, dont chaque élément $m_{ij}^{\mathcal{AB}}$ est égal au nombre d'éléments communs aux couleurs $\mathcal{A}^{-1}(i)$ et $\mathcal{B}^{-1}(j)$ (i.e. $m_{ij}^{\mathcal{AB}} = |\mathcal{A}^{-1}(i) \cap \mathcal{B}^{-1}(j)|$) (voir figure 4.3). Une telle matrice est appelée la matrice de raffinement de $\mathcal{A}$ et $\mathcal{B}$.

On note $\mathcal{M}^{\mathcal{AB}} = (m_{ij}^{\mathcal{AB}})_{i,j}$ la matrice de raffinement définie par les colorations $\mathcal{A}$ et $\mathcal{B}$

[2] d_n pour distance naïve puisque triviale mais totalement insatisfaisante.



Il y a deux éléments (les sommets 1 *et* 2*) dans l'intersection de la couleur* $\mathcal{A}^{-1}(1)$ *et* $\mathcal{B}^{-1}(1)$ *donc* $m_{11}^{\mathcal{AB}}$*. Les autres éléments de la matrice sont calculés de la même façon.*

FIG. 4.3 – Illustration d'une matrice de raffinement.

Par la suite, nous aurons besoin des trois propriétés suivantes sur la matrice de raffinement :

1. $\forall i \sum_j m_{ij}^{\mathcal{AB}} = |\mathcal{A}^{-1}(i)|$
2. $\forall j \sum_i m_{ij}^{\mathcal{AB}} = |\mathcal{B}^{-1}(j)|$
3. $\forall i,j \sum_{i,j} m_{ij}^{\mathcal{AB}} = |V|$

Définition 4.1 *ressemblance*
On définit la ressemblance entre deux colorations $\mathcal{A}$ et $\mathcal{B}$ relativement à l'application τ comme la valeur $s(\tau, \mathcal{A}, \mathcal{B})$ calculée par :

$$s(\mathcal{A}, \mathcal{B}, \tau) = \sum_i m_{i\tau(i)}^{\mathcal{AB}}$$

Définition 4.2 *différence*
On définit la différence entre deux colorations $\mathcal{A}$ et $\mathcal{B}$ relativement à l'application τ la valeur $e(\mathcal{A}, \mathcal{B}, \tau)$ calculée comme suit :

$$e(\mathcal{A}, \mathcal{B}, \tau) = |V| - s(\mathcal{A}, \mathcal{B}, \tau) = |V| - \sum_i m_{i\tau(i)}^{\mathcal{AB}}$$

Dans les deux définitions précédentes, l'application τ est une mise en correspondance des couleurs de $\mathcal{A}$ et $\mathcal{B}$ (i.e. $\tau : \{1..\chi(\mathcal{A})\} \longrightarrow \{1..\chi(\mathcal{B})\}$).

Remarque 1 $e(\mathcal{A},\mathcal{B},\tau) = \sum_{i,j\neq\tau(i)} m_{ij}^{\mathcal{AB}}$.

On utilise pour cela le troisième point des remarques sur la matrice de raffinement et la définition de e.

Remarque 2
$(e(\mathcal{A},\mathcal{B},\tau)=0) \Longleftrightarrow (\forall i \in \{1..\chi(\mathcal{A})\}, \forall j \in \{1..\chi(\mathcal{B})\}, (j \neq \tau(i) \Rightarrow m_{i,j}=0))$

Ceci provient directement de la proposition précédente. En effet, $e(\mathcal{A},\mathcal{B},\tau)$ peut être vu comme une somme de termes positifs. Cette somme est nulle si et seulement si chacun des termes est nul.

Remarque 3 *les calculs que nous effectuons peuvent servir à mettre en évidence la partie commune à deux colorations.*

En effet, on peut construire une coloration partielle $\mathcal{C}$ présentant des "éléments communs" à $\mathcal{A}$ et $\mathcal{B}$. $\mathcal{C}$ est définie comme la coloration partielle dont chaque classe est une classe de $\mathcal{B}$ à laquelle on a retiré les sommets n'appartenant à aucune classe de $\mathcal{A}$ mise en correspondance avec la classe de $\mathcal{B}$ considérée (voir figure 4.4 partie de droite). La quantité $e(\mathcal{A},\mathcal{B},\tau)$ mesure alors la qualité de la partie commune trouvée. Plus $e(\mathcal{A},\mathcal{B},\tau)$ est petit plus les parties communes exhibées sont importantes.

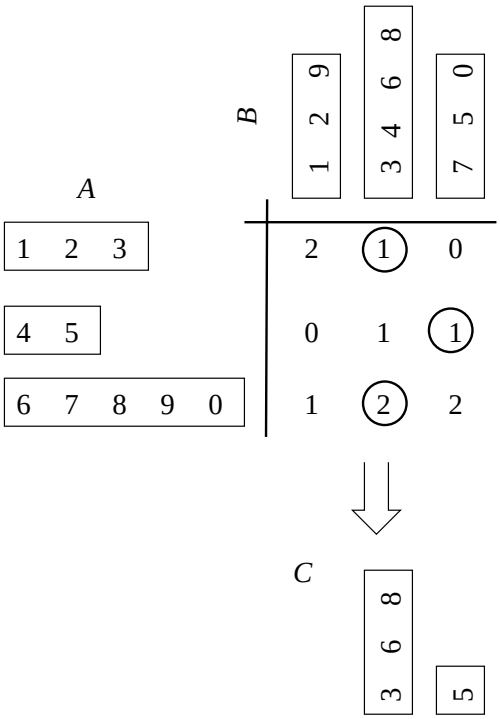
L'algorithme 2 explicite la construction de la partie commune en une complexité linéaire en fonction du nombre de sommets. Dans cet algorithme, on exploite encore une fois le codage par groupe vu dans la partie 3.2.1.

Algorithme 2 Mise en évidence des parties communes.

```
décolorer tous les sommets de C ;
for i de 1..χ(A) do
  for all sommet s de A^{-1}(i) do
    if B(s) = τ(i) then
      colorer s dans C avec la couleur τ(i)
    end if
  end for
end for
```

Propriété 1 *Si $\mathcal{A}$ et $\mathcal{B}$ sont des colorations valides alors la partie commune engendrée par τ est une coloration partielle valide.*

Preuve 4.1 Soit j une classe de $\mathcal{C}$. Soient s_1 et s_2 deux sommets de la classe j de $\mathcal{C}$, cela signifie qu'ils sont réunis dans $\mathcal{B}$ (i.e. $\mathcal{B}(s_1) = \mathcal{C}(s_1) = j = \mathcal{C}(s_2) = \mathcal{B}(s_2)$). Comme $\mathcal{B}$ est une coloration valide, s_1 et s_2 ne peuvent pas être adjacents. $\mathcal{C}$ est donc valide.



Dans le schéma, les cercles représentent les éléments de la matrice de raffinement "sélectionnés" (i.e. $m^{\mathcal{AB}}_{i\tau(i)}$*). À partir d'une application* τ *quelconque, on peut construire une coloration partielle.* $\mathcal{C}^{-1}(i) = \bigcup_{j\in\tau^{-1}(i)} (\mathcal{A}^{-1}(j) \bigcap \mathcal{B}^{-1}(i))$.

FIG. 4.4 – Déduction des parties communes à partir d'une application τ.

Remarque 4 Dans le cadre d'une coloration partielle, le calcul de $e(\mathcal{A}, \mathcal{B}, \tau)$ en passant par l'évaluation de la ressemblance est intéressant. On a alors de manière transparente : $e(\mathcal{A}, \mathcal{B}, \tau) = |V| - \sum_i m^{\mathcal{AB}}_{i\tau(i)} = \sum_{i,j\neq\tau(i)} m^{\mathcal{AB}}_{ij} + |\{s \in V | s \text{ non affecté}\}|$. En effet, les indicateurs mettent en avant le fait que certains sommets sont réunis dans les deux colorations. Or pour les colorations, les sommets non affectés ne doivent être rapprochés d'aucun autre sommet, quand bien même le sommet ne serait pas affecté dans les deux colorations partielles.

On peut alors caractériser la partie commune de plusieurs colorations en itérant le mécanisme.

4.2.2 Construction de la distance d_σ

Nous avons vu précédemment que une fonction τ quelconque était un moyen de mettre en correspondance les couleurs de $\mathcal{A}$ et $\mathcal{B}$. Nous présentons ici la fonction $\sigma^{\mathcal{AB}}$ qui effectue la mise en correspondance la plus proche possible de la définition de l'espace quotient défini en 3.2.2. Grâce à $\sigma^{\mathcal{AB}}$, nous pourrons construire une première métrique sur l'espace des partitionnements.

La relation d'équivalence $\sim$ suggère que la mise en correspondance doit être une bijection des couleurs de $\mathcal{A}$ dans celles de $\mathcal{B}$. En effet, on souhaite que la fonction trouvée approche la bijection de la définition de la relation d'équivalence en identifiant les couleurs des deux colorations. Par conséquent, il est nécessaire que les deux colorations possèdent le même nombre de couleurs. Si les nombres de couleurs des deux colorations sont différents, alors on pourra se ramener au cas d'égalité en ajoutant artificiellement[3] des couleurs vides à la coloration ayant le moins de couleurs.

Par conséquent, on définit $\sigma^{\mathcal{AB}}$ comme une bijection maximisant la quantité $s(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}})$. Puis, on définit $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim)$ comme étant la valeur de $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}})$ pour $\mathcal{A}$ (resp. $\mathcal{B}$) représentant de $\mathcal{A}_\sim$ (resp. $\mathcal{B}_\sim$). Il y a quatre propriétés à vérifier pour affirmer que d_σ soit une distance sur l'espace des partitionnements :

- La définition de d_σ effectue un choix au niveau des représentants utilisés et de la bijection maximale utilisée. Il faut donc vérifier que ces choix n'influence pas le calcul de la distance. En d'autres termes, d_σ est bien définie.
- d_σ doit être une application symétrique, c'est-à-dire que $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = d_\sigma(\mathcal{B}_\sim, \mathcal{A}_\sim)$ pour tout $\mathcal{A}_\sim$ et $\mathcal{B}_\sim$.
- d_σ doit vérifier $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = 0$ et équivalent à $\mathcal{A}_\sim = \mathcal{B}_\sim$. On dit alors que l'espace est séparé par d_σ.
- d_σ doit vérifier l'inégalité triangulaire, c'est-à-dire $d_\sigma(\mathcal{A}_\sim, \mathcal{C}_\sim) \leq d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) + d_\sigma(\mathcal{B}_\sim, \mathcal{C}_\sim)$. Intuitivement parlant, cette propriété garantit qu'il n'y a pas de "raccourci" dans l'espace.

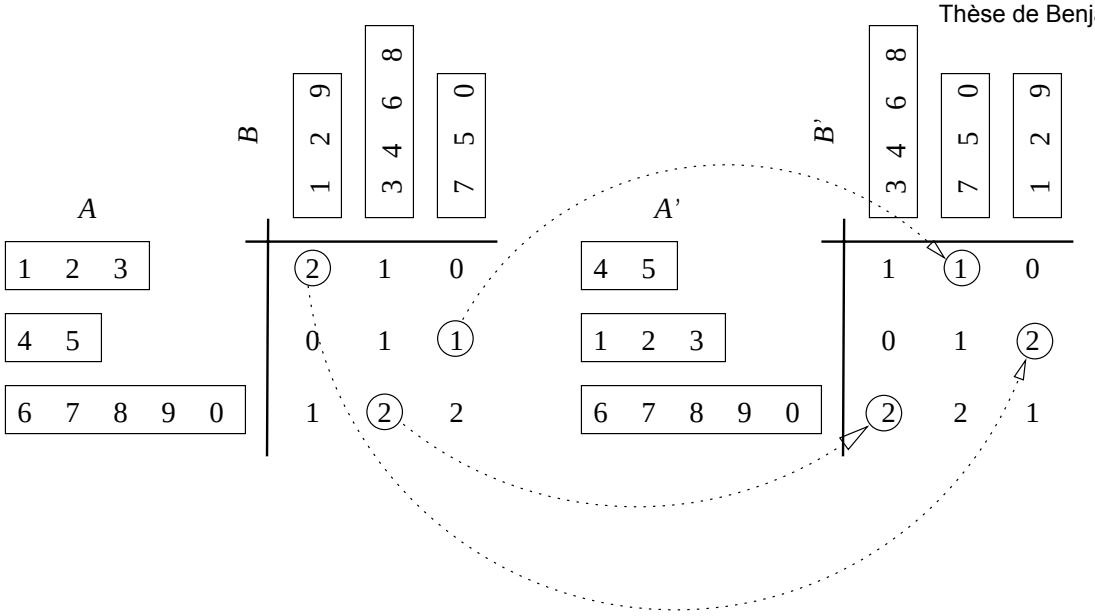
Preuve 4.2 *la valeur de $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}})$ est indépendante du choix de $\sigma^{\mathcal{AB}}$.*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soient σ et σ' deux bijections candidates pour être $\sigma^{\mathcal{AB}}$. $s(\mathcal{A}, \mathcal{B}, \sigma)$ est maximal parmi les bijections or σ' est une bijection. Par conséquent, $s(\mathcal{A}, \mathcal{B}, \sigma) \geq s(\mathcal{A}, \mathcal{B}, \sigma')$. De la même façon $s(\mathcal{A}, \mathcal{B}, \sigma') \geq s(\mathcal{A}, \mathcal{B}, \sigma)$ donc $s_\sigma = s'_\sigma$. D'où le calcul de $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}})$ est indépendant du choix de $\sigma^{\mathcal{AB}}$.

Preuve 4.3 *la valeur de $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim)$ est indépendante du choix des représentants $\mathcal{A}$ et $\mathcal{B}$.*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soient $\mathcal{A}'$ et $\mathcal{B}'$ deux autres représentants de $\mathcal{A}_\sim$ et $\mathcal{B}_\sim$. Il existe donc σ_1 et σ_2 tels que $\forall v \in V,\ \mathcal{A}'(v) = \sigma_1(\mathcal{A}(v))$ et $\mathcal{B}'(v) = \sigma_2(\mathcal{B}(v))$.

[3] pour les calculs.



Les cercles indiquent sur la matrice la mise en correspondance des couleurs de $\mathcal{A}$ (resp. $\mathcal{A}'$) sur les couleurs de $\mathcal{B}$ (resp. $\mathcal{B}'$). Les flèches en pointillés mettent en évidence que les couleurs $\sigma^{\mathcal{A}'\mathcal{B}'}$ peuvent être déduites de $\sigma^{\mathcal{A}\mathcal{B}}$ par la formule $\sigma^{\mathcal{A}'\mathcal{B}'}(i) = \sigma_2\sigma^{\mathcal{A}\mathcal{B}}\sigma_2^{-1}(i)$, où σ_1 (resp. σ_2) est le renommage des couleurs de $\mathcal{A}$ (resp. $\mathcal{B}$) pour obtenir les couleurs de $\mathcal{A}'$ (resp $\mathcal{B}'$).

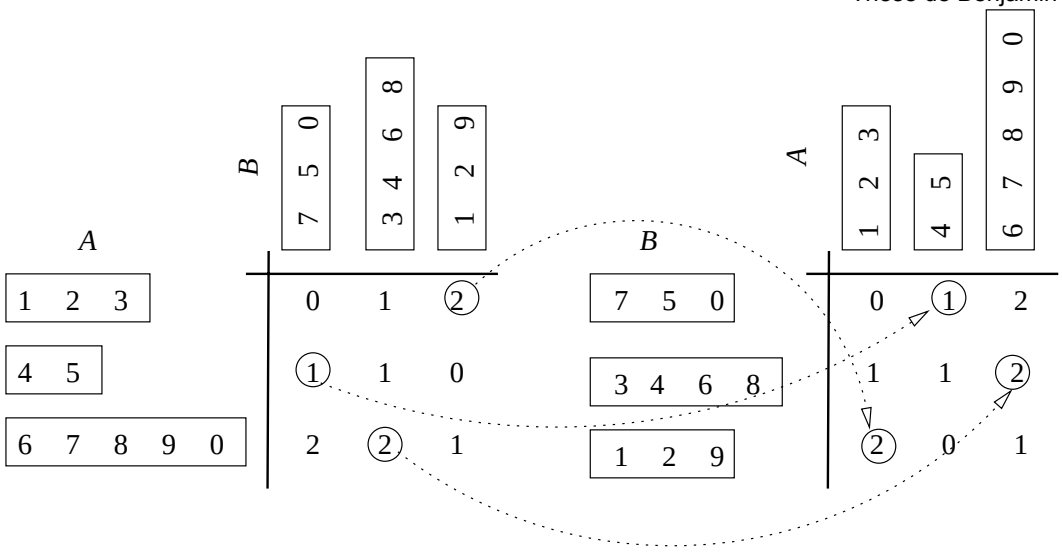
FIG. 4.5 – Illustration de la preuve 4.3 : indépendance du choix des représentants pour le calcul de $d_\sigma(\mathcal{A}, \mathcal{B})$.

On remarque que $m_{ij}^{\mathcal{A}'\mathcal{B}'} = m_{\sigma_1(i)\sigma_2(j)}^{\mathcal{A}\mathcal{B}}$. On a $s(\mathcal{A}', \mathcal{B}', \sigma^{\mathcal{A}'\mathcal{B}'}) = \sum_i m_{\sigma_1(i)\sigma_2(\sigma^{\mathcal{A}'\mathcal{B}'}(\sigma_1(i)))}^{\mathcal{A}\mathcal{B}}$ qui est naturellement inférieur ou égal à $s(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A}\mathcal{B}})$. Puisque $\mathcal{A}$ (resp. $\mathcal{B}$) et $\mathcal{A}'$ (resp. $\mathcal{B}'$) jouent des rôles symétriques on a aussi : $s(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A}\mathcal{B}}) \leq s(\mathcal{A}', \mathcal{B}', \sigma^{\mathcal{A}'\mathcal{B}'})$. D'où $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A}\mathcal{B}}) = e(\mathcal{A}', \mathcal{B}', \sigma^{\mathcal{A}'\mathcal{B}'})$. Par conséquent, d_σ est effectivement définie indépendamment des choix des représentants et de la bijection.

La définition de la distance d_σ est indépendante du choix des représentants et de l'éventuel choix de la bijection entre ces représentants. Par conséquent, la définition de d_σ, que nous proposons n'est pas sujette à ambiguïté. La figure 4.5 illustre cette preuve.

Preuve 4.4 *d_σ est symétrique.*

Soient $\mathcal{A}_\sim$ et $\mathcal{B}_\sim$ deux partitionnements. Notons $\mathcal{A}$ et $\mathcal{B}$ des représentants associés. On sait que $s(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A}\mathcal{B}}) = \sum_i m_{i,\sigma^{\mathcal{A}\mathcal{B}}(i)}^{\mathcal{A}\mathcal{B}} = K$ est maximale. Montrons que $s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{A}\mathcal{B}^{-1}}) = K$ et est maximale. $s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{A}\mathcal{B}^{-1}}) = \sum_i m_{i,\sigma^{\mathcal{A}\mathcal{B}^{-1}}(i)}^{\mathcal{B}\mathcal{A}}$. Or



Les cercles indiquent sur la matrice la mise en correspondance des couleurs de $\mathcal{A}$ (resp. $\mathcal{B}$) sur les couleurs de $\mathcal{B}$ (resp. $\mathcal{A}$). Les flèches en pointillés mettent en évidence que les couleurs $\sigma^{\mathcal{BA}}$ peuvent être déduites de $\sigma^{\mathcal{AB}}$ par la formule $\sigma^{\mathcal{AB}}(i) = \sigma^{\mathcal{AB}-1}(i)$.

FIG. 4.6 – Illustration de la preuve 4.4.

$\mathcal{M}^{\mathcal{BA}} = {}^t\mathcal{M}^{\mathcal{AB}}$. On en déduit que $s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{AB}^{-1}}) = \sum_i m^{\mathcal{AB}}_{\sigma^{\mathcal{AB}^{-1}}(i),i}$. En effectuant le changement d'indice $j = \sigma^{-1}(i)$, on obtient $s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{AB}^{-1}}) = \sum_{\sigma^{\mathcal{AB}}(j)} m^{\mathcal{AB}}_{j,\sigma^{\mathcal{AB}}(j)}$.
Puisque $\sigma^{\mathcal{AB}}(j)$ parcourt tous les indices, on peut simplifier le terme sous la somme. Ceci nous donne :

$$s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{AB}^{-1}}) = \sum_j m^{\mathcal{AB}}_{j,\sigma^{\mathcal{AB}}(j)} = K \tag{4.2}$$

Soit σ' une bijection des couleurs de $\mathcal{B}$ dans celles de $\mathcal{A}$. On utilise la même méthode que précédemment pour en déduire que :
$s(\mathcal{B}, \mathcal{A}, \sigma') = s(\mathcal{A}, \mathcal{B}, \sigma'^{-1}) \leq s^{\mathcal{AB}}_{\sigma}(\mathcal{A}, \mathcal{B}) = s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{AB}^{-1}})$. D'où $s(\mathcal{B}, \mathcal{A}, \sigma^{\mathcal{AB}^{-1}})$ est effectivement maximale est vaut M. Par conséquent, $d_\sigma(\mathcal{A}, \mathcal{B}) = d_\sigma(\mathcal{B}, \mathcal{A})$. La figure 4.4 illustre la preuve.

Preuve 4.5 $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = 0 \Longrightarrow \mathcal{A}_\sim = \mathcal{B}_\sim$

Soient $\mathcal{A}_\sim$ et $\mathcal{B}_\sim$ deux partitionnements. Notons $\mathcal{A}$ (resp. $\mathcal{B}$) un représentant de $\mathcal{A}_\sim$ (resp. $\mathcal{B}_\sim$). Supposons que $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = 0$, c'est-à-dire que $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) = 0$. Raisonnons par l'absurde en supposant qu'il existe un sommet v vérifiant $\sigma(\mathcal{A}(v)) \neq \mathcal{B}(v)$. Par conséquent $m^{\mathcal{AB}}_{\mathcal{A}(v)\mathcal{B}(v)} \geq 1$. Ce qui contredit l'hypothèse de distance nulle. En somme, un tel sommet ne peut donc pas exister. On a donc pour tout sommet

v, $\sigma(\mathcal{A}(v)) = \mathcal{B}(v)$. Ceci prouve que $\mathcal{A}$ et $\mathcal{B}$ sont équivalents (i.e. $\mathcal{A}_\sim = \mathcal{B}_\sim$).

Preuve 4.6 $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = 0 \Longleftarrow \mathcal{A}_\sim = \mathcal{B}_\sim$

Soit $\mathcal{A}_\sim$ un partitionnement. $\mathcal{A}$ peut être représenté par deux colorations $\mathcal{A}$ et $\mathcal{B}$. Par définition $\mathcal{A} \sim \mathcal{B}$ (autrement dit $\mathcal{A}_\sim = \mathcal{B}_\sim$), donc il existe un bijection $\sigma : \{1...\chi(\mathcal{A})\} \to \{1...\chi(\mathcal{B})\}$, telle que tous les sommets s de V vérifient $\sigma(\mathcal{A}(s)) = \mathcal{B}(s)$. Par conséquent, la matrice de raffinement de $\mathcal{A}$ et $\mathcal{B}$ contient exactement un nombre non nul par ligne (et par colonne) au position $m_{i,\sigma(i)}$. En utilisant la remarque 2 (dans le sens réciproque), on obtient que $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) = 0$. Donc $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = 0$.

Théorème 3 *d_σ vérifie l'inégalité triangulaire.*

Afin de montrer cette propriété, on va utiliser les deux lemmes suivants :

Lemme 2 *Pour toutes colorations $\mathcal{A}$, $\mathcal{B}$ et pour tout voisin $\mathcal{A}'$ de $\mathcal{A}$, on a la relation suivante :* $|e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) - e(\mathcal{A}', \mathcal{B}, \sigma^{\mathcal{A'B}})| \leq 1$. Autrement dit, on peut borner la modification de l'indicateur lorsqu'on remplace une coloration par une coloration voisine.

Lemme 3 *Pour toutes colorations $\mathcal{A}$ et $\mathcal{B}$, pour tout sommet s vérifiant $\sigma(\mathcal{A}(s)) \neq \mathcal{B}(s)$. On a l'égalité :* $e(\mathcal{A}, \mathcal{B}', \sigma^{\mathcal{AB}'}) - e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) = -1$;
où $\mathcal{B}' = changerCouleur(\mathcal{B}, s, \sigma(\mathcal{A}(s)))$.

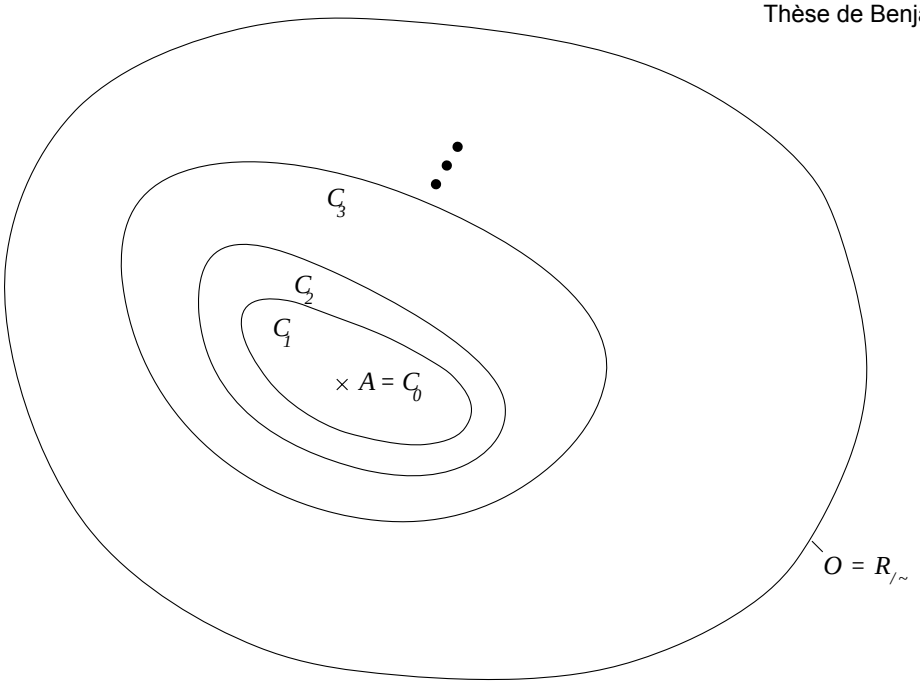
En d'autres termes, si $\mathcal{A}$ et $\mathcal{B}$ sont différents, on connaît un moyen de se rapprocher de $\mathcal{A}$ en partant de $\mathcal{B}$.

Preuve 4.7 *du théorème 3 par récurrence*

Soient $\mathcal{A}$ et $\mathcal{C}$ deux colorations. Montrons par récurrence que pour tout $\mathcal{B}$, $e(\mathcal{A}, \mathcal{C}, \sigma^{\mathcal{AC}}) \leq e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) + e(\mathcal{B}, \mathcal{C}, \sigma^{\mathcal{BC}})$.

1. Soit $\mathcal{P}_n$ la proposition suivante : "$\forall$ $\mathcal{B}$ tel que $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A},\mathcal{B}}) = n$", alors on a l'inégalité suivante : $e(\mathcal{A}, \mathcal{C}, \sigma^{\mathcal{A},\mathcal{C}}) \leq e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{A},\mathcal{B}}) + e(\mathcal{B}, \mathcal{C}, \sigma^{\mathcal{BC}})$
2. Puisque d_σ est séparé on a trivialement $\mathcal{P}_0$ vrai.
3. Supposons $\mathcal{P}_n$. Soit $\mathcal{B}$ une coloration telle que $e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) = n + 1$. Il existe donc un sommet s tel que $s \notin \mathcal{A}^{-1}(\mathcal{A}(s)) \bigcap \mathcal{B}^{-1}(\sigma(\mathcal{A}(s)))$.
   Construisons $\mathcal{B}' = changerCouleur(\mathcal{B}, s, \sigma(\mathcal{A}(s)))$. On sait par le lemme 3 que $e(\mathcal{A}, \mathcal{B}', \sigma^{\mathcal{AB}'}) = n$. On en déduit donc par l'hypothèse de récurrence que $e(\mathcal{A}, \mathcal{C}, \sigma^{\mathcal{A},\mathcal{C}}) \leq e(\mathcal{A}, \mathcal{B}', \sigma\mathcal{AB}') + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}})$.
   Or on sait aussi par le lemme 2 que $e(\mathcal{B}, \mathcal{C}, \sigma^{\mathcal{BC}}) \geq e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}',\mathcal{C}}) - 1$.

$$\begin{array}{rcl} e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) + e(\mathcal{B}, \mathcal{C}, \sigma^{\mathcal{BC}}) & \geq & n + 1 + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) - 1 \\ e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) & \geq & n + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) \\ e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) & \geq & e(\mathcal{A}, \mathcal{B}', \sigma^{\mathcal{AB}'}) + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) \\ e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) + e(\mathcal{B}', \mathcal{C}, \sigma^{\mathcal{B}'\mathcal{C}}) & \geq & e(\mathcal{A}, \mathcal{C}, \sigma^{\mathcal{AC}}) \end{array}$$



On prouve pour une coloration quelconque $\mathcal{A}$, que l'inégalité triangulaire est vérifié pour toutes colorations à une distance donnée n. Grâce à ces colorations, on peut propager l'inégalité triangulaire aux colorations à une distance plus grande $n+1$, jusqu'à recouvrir tout l'espace de recherche.

FIG. 4.7 – Principe de la preuve de l'inégalité triangulaire.

4. donc d_σ vérifie l'inégalité triangulaire.

Cette preuve est illustrée par la figure 4.7.

Preuve 4.8 *du lemme 2*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soit $\mathcal{A}'$ un voisin de $\mathcal{A}$ par le changement de couleur du sommet s. On note $\mathcal{M}^{\mathcal{AB}}$ (resp. $\mathcal{M}^{\mathcal{A'B}}$) la matrice de raffinement induite par le calcul de $e(\mathcal{A},\mathcal{B},\sigma^{\mathcal{AB}})$ (resp. $e(\mathcal{A}',\mathcal{B},\sigma^{\mathcal{A'B}})$). On a les relations suivantes :

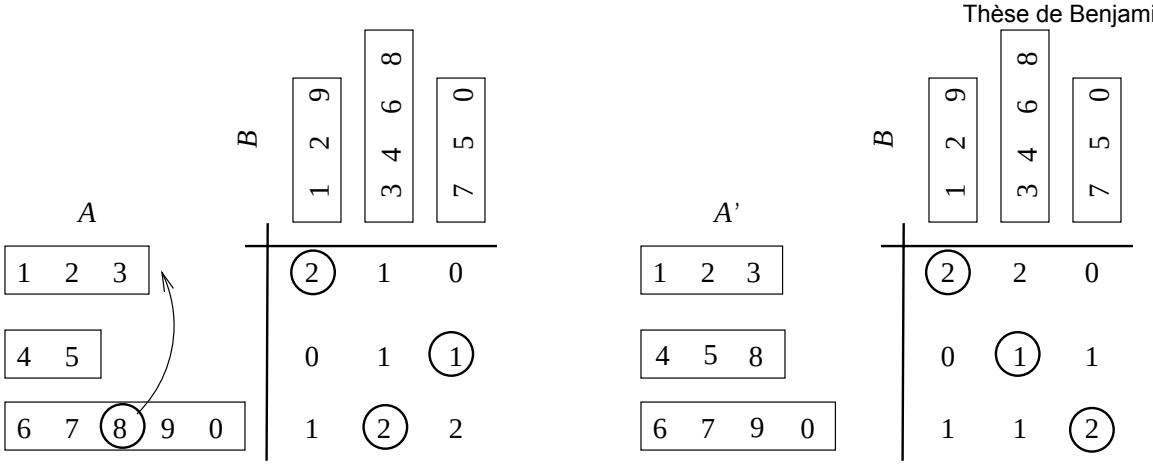
$$|\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{A'B}}_{i\sigma^{\mathcal{AB}}(i)}| \leq 1 \tag{4.3}$$

car le changement de la matrice de raffinement n'est en réalité que la décrémentation d'un élément et l'incrémentation d'un autre élément sur la même ligne.

$$|\sum_i m^{\mathcal{A'B}}_{i\sigma^{\mathcal{A'B}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A'B}}(i)}| \leq 1 \tag{4.4}$$

pour la même raison.

$$\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A'B}}(i)} \geq 0 \tag{4.5}$$



On transforme $\mathcal{A}$ en $\mathcal{A}'$. Cette modification a une incidence restreinte sur la matrice de raffinement. Par conséquent, la valeur de l'indicateur varie peu, malgré les modifications sur la bijection. On remarque dans le cas présent qu'il y a un changement de bijection mais que la distance ne varie pas.

FIG. 4.8 – Illustration du lemme 2.

car $\sigma^{\mathcal{AB}}$ est maximal pour $\mathcal{M}^{\mathcal{AB}}$

$$\sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{AB}}(i)} \geq 0 \tag{4.6}$$

car $\sigma^{\mathcal{A}'\mathcal{B}}$ est maximal pour $\mathcal{M}^{\mathcal{A}'\mathcal{B}}$.

À partir de ces relations on en déduit :

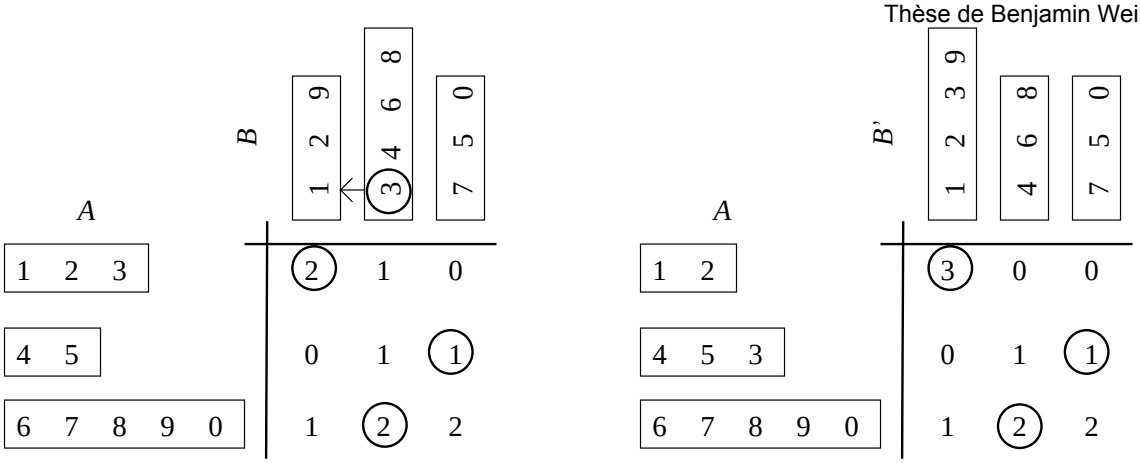
$$\begin{aligned}
\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{AB}}(i)} &\leq 1 \quad \text{(par 4.3)} \\
\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{AB}}(i)} &\leq 1 + \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{AB}}(i)} \quad \text{(en utilisant 4.6)} \\
\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} &\leq 1 \quad \text{(en simplifiant par } \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{AB}}(i)}\text{)}
\end{aligned}$$

et

$$\begin{aligned}
\sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} &\leq 1 \quad \text{(par 4.4)} \\
\sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} &\leq 1 + \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} \quad \text{(en utilisant 4.5)} \\
\sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)} - \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} &\leq 1 \quad \text{(en simplifiant par } \sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)}\text{)}
\end{aligned}$$

Donc $|\sum_i m^{\mathcal{AB}}_{i\sigma^{\mathcal{AB}}(i)} - \sum_i m^{\mathcal{A}'\mathcal{B}}_{i\sigma^{\mathcal{A}'\mathcal{B}}(i)}| \leq 1$. C'est-à-dire $|s(\mathcal{A},\mathcal{B},\sigma^{\mathcal{AB}}) - s(\mathcal{A}',\mathcal{B},\sigma^{\mathcal{A}'\mathcal{B}})| \leq 1$.

En conclusion, $|e(\mathcal{A},\mathcal{B},\sigma^{\mathcal{AB}}) - e(\mathcal{A}',\mathcal{B},\sigma^{\mathcal{A}'\mathcal{B}})| \leq 1$.



Le sommet 3 n'est pas dans l'intersection mise en évidence par la bijection (les cercles). Le fait de rapprocher le sommet 3 de l'intersection en changeant sa couleur dans la coloration $\mathcal{B}$ permet de diminuer la distance de 1.

FIG. 4.9 – Illustration du lemme 3.

La figure 4.8 illustre la preuve 2.

Preuve 4.9 *du lemme 3*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soit s un sommet vérifiant $\sigma(\mathcal{A}(s)) \neq \mathcal{B}(s)$. Posons $\mathcal{B}' = changerCouleur(\mathcal{B}, s, \sigma^{\mathcal{AB}}(\mathcal{A}(s)))$. On peut choisir $\sigma^{\mathcal{AB}'} = \sigma^{\mathcal{AB}}$. En effet, l'élément de la matrice d'indice $\mathcal{A}(s)\sigma(\mathcal{A}(s))$ a augmenté de un alors que les autres éléments de la ligne sont restés les mêmes ou ont diminué[4].
On a donc trivialement $e(\mathcal{A}, \mathcal{B}', \sigma^{\mathcal{AB}'}) - e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}}) = -1$. La figure 4.10 illustre la preuve 3.

Nous avons donc obtenu une métrique d_σ sur l'espace des partitionnements. Afin de calculer la distance d_σ entre deux partitionnements, il est suffisant de trouver la permutation maximale entre les deux colorations les représentant. Ce problème est en réalité un problème de couplage qui peut être résolue par la méthode Hongroise (ou algorithme de Kuhn) [CT80]. L'algorithme 3 donne le principe de cette méthode. Il utilise deux sous-méthodes :

- Réduire : cette méthode retranche à chaque ligne le minimum de la matrice $\mathcal{M}$. Cette opération ainsi que les opérations d'ajout sur une colonne et de retranchement sur une colonne ne changent en aucune façon l'ensemble des solutions optimales.
- Affecter le plus de zéros : ceci se réduit en un problème d'affectation binaire, qui peut se résoudre par un algorithme de flot maximal tel que Ford-Fulkerson par exemple.

[4] seul l'élément d'indice $\mathcal{A}(s)\mathcal{B}(s)$ a diminué.

Algorithme 3 Méthode hongroise pour la résolution du problème de couplage minimum.

Réduire $\mathcal{M}$;
affecter le plus de zéro possible ;
while on ne peut pas affecter n zéros **do**
  marquer toutes les lignes sans zéro affecté ;
  while on peut marquer quelque chose **do**
    marquer toutes les colonnes ayant un zéro non affecté dans une ligne marquée ;
    marquer toutes les lignes ayant un zéro affecté dans une colonne marquée ;
  end while
  Notons r le plus petit élément des lignes et colonnes non marquées ;
  soustraire r aux lignes marquées ;
  ajouter r aux colonnes marquées ;
  Réduire $\mathcal{M}$;
end while

La distance d_σ est adaptée lorsque le nombre de couleurs utilisées est le même pour toutes les colorations considérées. Elle est donc plus intéressante pour le **GkCP** que dans le **MGCP**. De plus, nous verrons que la complexité nécessaire pour calculer la valeur exacte de la distance, bien que polynomiale, est plus importante que pour la métrique à venir.

4.2.3 Construction de la distance d_ρ

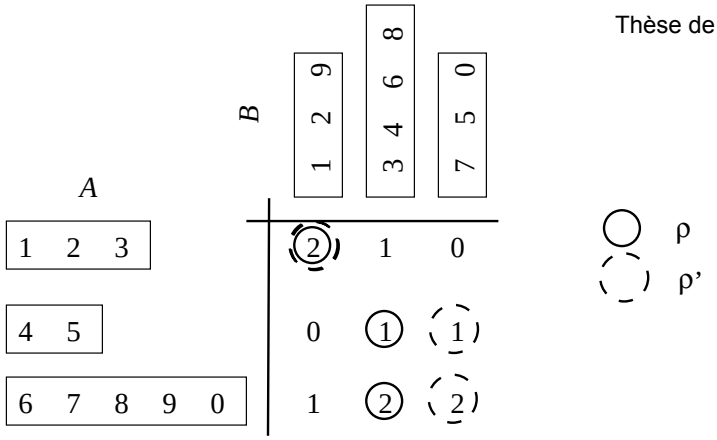
Dans cette partie, nous traitons d'un autre façon la mise en correspondance des couleurs des colorations considérées. Pour deux colorations $\mathcal{A}$ et $\mathcal{B}$, on construit $\rho^{\mathcal{AB}}$ comme suit : $\rho^{\mathcal{AB}}(i) = \arg\max_j (m_{ij}^{\mathcal{AB}})$. Puis, nous construisons la distance d_ρ à l'aide de e et de l'application $\rho^{\mathcal{AB}}$ ainsi définie. Nous verrons que l'indicateur que nous avons construit n'est pas directement une distance. Nous prouvons cependant un certain nombre de propriétés sur $\rho^{\mathcal{AB}}$ permettant de prouver que la forme que nous avons construite à partir de l'indicateur est effectivement une métrique.

Les deux premières propriétés servirons à montrer que la définition de d_ρ que nous utilisons est indépendante de tout choix.

Propriété 2 *La valeur de l'indicateur $e(\mathcal{A}, \mathcal{B}, \rho^{\mathcal{AB}})$ entre deux colorations quelconques $\mathcal{A}$ et $\mathcal{B}$ est indépendante d'éventuels choix de $\rho^{\mathcal{AB}}$.*

Preuve 4.10

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations, soient ρ et ρ' vérifiant toutes deux la propriété de la définition (i.e. on a pour tout i, $m_{i\rho(i)}^{\mathcal{AB}} = max_j m_{i,j}^{\mathcal{AB}} = m_{i\rho'(i)}^{\mathcal{AB}}$). On en déduit que $s(\mathcal{A}, \mathcal{B}, \rho) = s(\mathcal{A}, \mathcal{B}, \rho')$; d'où $e(\mathcal{A}, \mathcal{B}, \rho) = e(\mathcal{A}, \mathcal{B}, \rho')$.



Bien que les applications ρ et ρ' soient différentes, le fait qu'elles soient maximales est suffisant : $e(\mathcal{A},\mathcal{B},\rho) = 5 = e(\mathcal{A},\mathcal{B},\rho')$

FIG. 4.10 – Illustration d'indépendance du choix de l'application ρ maximale.

La figure 4.8 illustre la preuve 2.

Propriété 3 $\forall\mathcal{A}\forall\mathcal{A}'\forall\mathcal{B}\forall\mathcal{B}'(\mathcal{A}\sim\mathcal{A}'$ *et* $\mathcal{B}\sim\mathcal{B}' \Longrightarrow e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = e(\mathcal{A}',\mathcal{B}',\rho^{\mathcal{A}'\mathcal{B}'}))$.

Preuve 4.11

Soient $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{B}$ et $\mathcal{B}'$ des colorations. Supposons que $\mathcal{A}\sim\mathcal{A}'$ et $\mathcal{B}\sim\mathcal{B}'$. Il existe donc par définition σ_a et σ_b tel que pour tout sommet v, $\mathcal{A}'(v) = \sigma_a(\mathcal{A}(v))$ et $\mathcal{B}'(v) = \sigma_b(\mathcal{B}(v))$. Par conséquent, on a pour tout i $\mathcal{A}'^{-1}(i) = \mathcal{A}^{-1}(\sigma_a^{-1}(i))$ et $\mathcal{B}'^{-1}(i) = \mathcal{B}^{-1}(\sigma_b^{-1}(i))$. D'où la matrice de raffinement $\mathcal{M}^{\mathcal{A}'\mathcal{B}'}$ est déduite de $\mathcal{M}^{\mathcal{AB}}$ en réordonnant les lignes et les colonnes :

$$m_{ij}^{\mathcal{A}'\mathcal{B}'} = m_{\sigma_a^{-1}(i)\sigma_b^{-1}(j)}^{\mathcal{AB}} \tag{4.7}$$

On en déduit que :

$$\begin{aligned}\rho^{\mathcal{AB}}(i) = \arg\max_j\ (m_{ij}^{\mathcal{AB}}) &= \arg\max_{\sigma_b(j)}\ (m_{\sigma_a(i),\sigma_b(j)}^{\mathcal{A}'\mathcal{B}'})\\ &= \arg\max_j\ (m_{\sigma_a(i)j}^{\mathcal{A}'\mathcal{B}'})\\ &= \rho^{\mathcal{A}'\mathcal{B}'}(\sigma_a(i))\end{aligned}$$

Puisqu'on somme sur tous les indices i, on obtient :

$$\begin{aligned}e(\mathcal{AB},\rho^{\mathcal{AB}}) = \sum_i m_{i\rho^{\mathcal{AB}}(i)}^{\mathcal{AB}} &= \sum_{\sigma_a(i)} m_{\sigma_a(i)\rho^{\mathcal{A}'\mathcal{B}'}(\sigma_a(i))^{\mathcal{A}'\mathcal{B}'}}\\ &= \sum_i m_{i\rho^{\mathcal{A}'\mathcal{B}'}(i)}^{\mathcal{A}'\mathcal{B}'}\\ &= e(\mathcal{A}',\mathcal{B}',\rho^{\mathcal{A}'\mathcal{B}'})\end{aligned}$$

On constate que la valeur de l'indicateur ne varie pas si on change de représentants.

Propriété 4 *Pour toutes colorations $\mathcal{A}$ et $\mathcal{B}$, si $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$ alors $\rho^{\mathcal{AB}}$ est surjectif.*

Preuve 4.12 *de la propriété 4 par la contraposée.*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Supposons que $\rho^{\mathcal{AB}}$ ne soit pas surjectif alors il existe $j \in \{1..\chi(\mathcal{B})\}$ tel que pour tout $i \in \{1..\chi(\mathcal{A})\}$ on ait $\rho(i) \neq j$. On a donc $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) \geq |\mathcal{B}^{-1}(j)| > 0$ (par le troisième point de la remarque). Par conséquent, $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) \geq |\mathcal{B}^{-1}(j)| \neq 0$. Ce qui prouve la contraposée de la propriété 4.

Propriété 5 *Pour toutes colorations $\mathcal{A}$ et $\mathcal{B}$, si $\chi(\mathcal{A}) < \chi(\mathcal{B})$ alors $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) \neq 0$*

En effet, dans ce cas ρ ne peut être surjectif et en utilisant la propriété 5, la distance entre les deux colorations est nécessairement strictement positif.

Propriété 6 *Pour toutes colorations $\mathcal{A}$ et $\mathcal{B}$ la proposition "$e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$" est équivalente à "pour chaque classe i de $\mathcal{A}$, il existe une classe j de $\mathcal{B}$ tel que $\mathcal{A}^{-1}(i)$ est incluse dans $\mathcal{B}^{-1}(j)$".*

En d'autres termes, $\mathcal{A}_\sim$ est un raffinement de $\mathcal{B}_\sim$. Cette propriété nous servira pour prouver la séparation de d_ρ.

Preuve 4.13 $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0 \Longleftarrow \forall i\ \exists j\ \mathcal{A}^{-1}(i) \subset \mathcal{B}^{-1}(j)$

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations, supposons que pour chaque classe i de $\mathcal{A}$, il existe une couleur j de $\mathcal{B}$ tel que $\mathcal{A}^{-1}(i)$ soit incluse dans $\mathcal{B}^{-1}(j)$. Soient $i \in \{1..\chi(\mathcal{A})\}$ et $j \in \{1..\chi(\mathcal{B})\}$,

$$m_{ij}^{\mathcal{AB}} = \begin{cases} |\mathcal{A}^{-1}(i)| & \text{si } \mathcal{A}^{-1}(i) \subset \mathcal{B}^{-1}(j) \\ 0 & \text{sinon} \end{cases}$$

On en déduit que sur chaque ligne i, il y a au plus un seul élément non nul $m_{ij}^{\mathcal{AB}}$. Par définition de ρ, on en déduit que l'image de i est j. Autrement dit, $\forall i \in \{1..\chi(\mathcal{A})\}, \forall j \in \{1..\chi(\mathcal{B})\}, (j \neq \rho(i)) \Rightarrow m_{ij}^{\mathcal{AB}} = 0)$. D'où d'après la propriété 2, $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$.

Preuve 4.14 $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0 \Longrightarrow \forall i\ \exists j\ \mathcal{A}^{-1}(i) \subset \mathcal{B}^{-1}(j)$

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations, supposons que $\chi(\mathcal{A}) \geq \chi(\mathcal{B})$ et $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$. Soit $i \in \{1..\chi(\mathcal{A})\}$. Comme $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$, on sait d'après la propriété 2, que $\mathcal{M}^{\mathcal{AB}}$ ne possède qu'un seul élément, $m_{ij}^{\mathcal{AB}}$, non nul sur la ligne i. D'où pour tout $s \in \mathcal{A}^{-1}(i)$, $\mathcal{B}(s) = j$, autrement $s \in \mathcal{B}^{-1}(j)$.

Propriété 7 *Pour toutes colorations* $\mathcal{A}$ *et* $\mathcal{B}$, $(\chi(\mathcal{A}) = \chi(\mathcal{B})$ *et* $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0) \Longleftrightarrow \mathcal{A} \sim \mathcal{B}$

Preuve 4.15 $(\chi(\mathcal{A}) = \chi(\mathcal{B})$ *et* $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0) \Longrightarrow \mathcal{A} \sim \mathcal{B}$

On applique deux fois la proposition précédente pour $\mathcal{A}$ et $\mathcal{B}$ puis pour $\mathcal{B}$ et $\mathcal{A}$.

Preuve 4.16 $(\chi(\mathcal{A}) = \chi(\mathcal{B})$ *et* $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0) \Longleftarrow \mathcal{A} \sim \mathcal{B}$

Soit $\mathcal{A}_\sim$ un partitionnement. $\mathcal{A}$ peut être représenté par deux colorations $\mathcal{A}$ et $\mathcal{B}$. Par définition $\mathcal{A} \sim \mathcal{B}$ (autrement dit $\mathcal{A}_\sim = \mathcal{B}_\sim$), il existe un bijection $\sigma : \{1...\chi(\mathcal{A})\} \to \{1...\chi(\mathcal{B})\}$, telle que tous les sommets s de V vérifient $\sigma(\mathcal{A}(s)) = \mathcal{B}(s)$. Par conséquent, la matrice de raffinement de $\mathcal{A}$ et $\mathcal{B}$ contient exactement un nombre non nul par ligne (et par colonne) aux positions $m_{i,\sigma(i)}$. En utilisant la remarque 2 (dans le sens réciproque), on obtient que $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$. Donc $d_\sigma(\mathcal{A}_\sim,\mathcal{B}_\sim) = 0$.

Un lecteur averti, remarquera sans doute que la propriété 7 indique un autre moyen de tester l'égalité des partitionnements.

Théorème 4 *on a la relation suivante :* $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) \geq e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}})$

Cette propriété sera nécessaire pour obtenir l'inégalité triangulaire.

Afin de montrer ce théorème, on utilise la même stratégie que pour d_σ. Cependant, il faut être vigilant car nous n'avons pas les propriétés de symétrie et de séparation : en particulier, pour l'initialisation de la récurrence de la preuve de l'inégalité triangulaire.

Lemme 4 *Pour toutes colorations* $\mathcal{A}$ *et* $\mathcal{B}$, *pour tout* $\mathcal{A}'$ *voisin de* $\mathcal{A}$ *on a* $|e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) - e(\mathcal{A}',\mathcal{B},\rho^{\mathcal{A}'\mathcal{B}})| \leq 1$.

Lemme 5 *Pour toutes colorations* $\mathcal{A}$ *et* $\mathcal{B}$, *pour tout sommet* s *vérifiant* $\rho(\mathcal{A}(s)) \neq \mathcal{B}(s)$, *on a l'égalité* $e(\mathcal{A},\mathcal{B}',\rho^{\mathcal{AB}'}) - e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = -1$, *où* $\mathcal{B}' = changer(\mathcal{B}, s, \rho^{\mathcal{AB}}(\mathcal{A}(s)))$.

Preuve 4.17 *du théorème 4 par récurrence*

Soient $\mathcal{A}$ et $\mathcal{C}$ deux colorations. Montrons par récurrence que pour tout $\mathcal{B}$, $e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}}) \leq e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}})$.

1. Soit $\mathcal{P}_n$ la proposition suivante : "$\forall$ $\mathcal{B}$ tel que $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = n$", alors on a l'inégalité suivante : $e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}}) \leq e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}})$.
2. Vérifions $\mathcal{P}_0$. Soit $\mathcal{B}$ tel que $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = 0$. Par la proposition 6, on sait que toutes les couleurs de $\mathcal{A}$ sont incluses dans une couleur de $\mathcal{B}$.
   D'où $m^{\mathcal{BC}}_{i,j} = \sum_{k,\mathcal{A}^{-1}(k)\subset\mathcal{B}^{-1}(j)} m_{k,j}$.

Puisque $m_{i,j} \geq 0$, on a la relation suivante $\max_i m_{i,j}^{\mathcal{BC}} \leq \max_i \sum_{k,\mathcal{A}^{-1}(k)\subset\mathcal{B}^{-1}(j)} m_{k,j}$.
Par conséquent, $e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) \geq e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}})$. Donc $\mathcal{P}_0$ est vrai.

3. Supposons $\mathcal{P}_n$. Soit $\mathcal{B}$ une coloration telle que $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{A},\mathcal{B}}) = n+1$. Il existe donc un sommet s tel que $s \notin \mathcal{A}^{-1}(\mathcal{A}(s)) \bigcap \mathcal{B}^{-1}(\rho(\mathcal{A}(s)))$.
Construisons $\mathcal{B}' = changerCouleur(\mathcal{B}, s, \rho(\mathcal{A}(s)))$. $d(\mathcal{A},\mathcal{B}') = n$, on en déduit par l'hypothèse de récurrence que $e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}}) \leq e(\mathcal{A},\mathcal{B}',\rho^{\mathcal{A},\mathcal{B}'}) + e(\mathcal{B}',\mathcal{C},\rho^{\mathcal{B}'\mathcal{C}})$.
Or $e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) \geq e(\mathcal{B}',\mathcal{C},\rho^{\mathcal{B}',\mathcal{C}}) - 1$ par le lemme 4.

$$\begin{array}{rcl} e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) & \geq & n+1+e(\mathcal{B}',\mathcal{C},\rho^{\mathcal{B}'\mathcal{C}}) - 1 \\ e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) & \geq & n + e(\mathcal{B}',\mathcal{C},\rho^{\mathcal{B}'\mathcal{C}}) \\ e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) & \geq & e(\mathcal{A},\mathcal{B}',\rho^{\mathcal{AB}'}) + e(\mathcal{B}',\mathcal{C},\rho^{\mathcal{B}'\mathcal{C}}) \\ e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) & \geq & e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}}) \end{array}$$

On a donc prouvé que $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{BC}}) \geq e(\mathcal{A},\mathcal{C},\rho^{\mathcal{AC}})$.

Preuve 4.18 *du lemme 4*

Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soit $\mathcal{A}'$ un voisin de $\mathcal{A}$ par le changement de couleur du sommet s. Les matrices $\mathcal{M}^{\mathcal{AB}}$ et $\mathcal{M}^{\mathcal{A}'\mathcal{B}}$ induites pour le calcul de $e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}})$ et de $e(\mathcal{A}',\mathcal{B},\rho^{\mathcal{A}'\mathcal{B}})$ diffèrent sur deux éléments d'indice $(\mathcal{A}(s),\mathcal{B}(s))$ et $(\mathcal{A}'(s),\mathcal{B})$. De plus, $m^{\mathcal{AB}}_{\mathcal{A}(s),\mathcal{B}(s)} - m^{\mathcal{A}'(s)\mathcal{B}}_{\mathcal{A}'(s),\mathcal{B}(s)} = 1$ et $m^{\mathcal{AB}}_{\mathcal{A}(s),\mathcal{B}'(s)} - m^{\mathcal{A}'\mathcal{B}}_{\mathcal{A}'(s),\mathcal{B}(s)} = -1$. On en déduit que le maximum sur la colonne ne peut varier que d'au plus un.

Preuve 4.19 *du lemme 5*

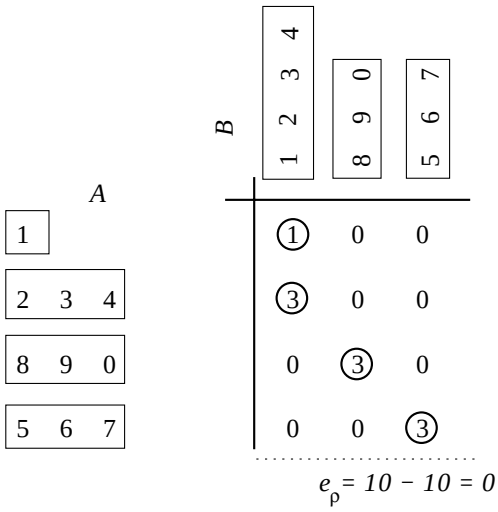
Soient $\mathcal{A}$ et $\mathcal{B}$ deux colorations. Soit s un sommet vérifiant $\rho(\mathcal{A}(s)) \neq \mathcal{B}(s)$. Posons $\mathcal{B}' = changer(\mathcal{B}, s, \rho^{\mathcal{AB}}(\mathcal{A}(s)))$. On sait que $\rho^{\mathcal{AB}'}(\mathcal{A}(s)) = \rho^{\mathcal{AB}}(\mathcal{A}(s))$. En effet l'élément de la matrice d'indice $\mathcal{A}(s)\rho(\mathcal{A}(s))$ a augmenté de un alors que les autres éléments de la ligne sont restés les mêmes ou ont diminué[5]. On a donc trivialement $e(\mathcal{A},\mathcal{B}',\rho^{\mathcal{AB}'}) - e(\mathcal{A},\mathcal{B},\rho^{\mathcal{AB}}) = -1$.

Cependant, l'indicateur que nous venons de construire présente un handicap majeur : ce n'est pas une distance.

Premièrement, nous n'avons pas la séparation. En effet, la proposition 6 nous indique seulement si la première coloration est un raffinement de la seconde (voir figure 4.11).

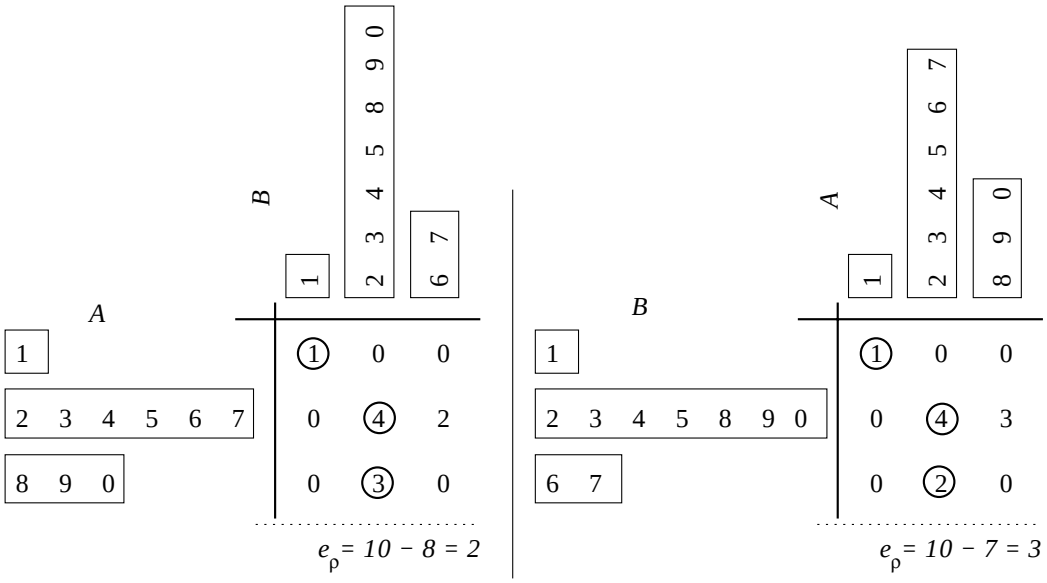
De plus, nous n'avons pas la propriété de symétrie. Par exemple, la figure 4.12 montre deux colorations pour lesquelles le calcul de l'indicateur n'est pas symétrique. Dans la partie gauche, la valeur de l'indicateur obtenue est de deux alors qu'en utilisant la partie droite on trouve une valeur de trois. Ceci fournit deux raisons justifiant que cet indicateur n'est pas vraiment une distance.

[5] seule l'élément d'indice $\mathcal{A}(s)\mathcal{B}(s)$ a diminué.



On constate ici que $\mathcal{A}_\sim$ est un raffinement de $\mathcal{B}_\sim$. L'indicateur fournit une valeur nulle malgré le fait que $\mathcal{A}_\sim$ et $\mathcal{B}_\sim$ soient différents.

FIG. 4.11 – Illustration de la non séparation de l'indicateur reposant sur la fonction $\rho^{\mathcal{AB}}$.



Les deux tableaux représentent les matrices des cardinalités des intersections des couleurs de deux colorations. Dans la première matrice on calcule la distance de $\mathcal{A}$ à $\mathcal{B}$. La seconde matrice est la transposée de la précédente matrice, on y calcule la distance de $\mathcal{B}$ à $\mathcal{A}$. On constate alors que $e(\mathcal{A}, \mathcal{B}, \rho^{\mathcal{AB}}) = 2 \neq 3 = e(\mathcal{B}, \mathcal{A}, \rho^{\mathcal{BA}})$.

FIG. 4.12 – Non symétrie de la fonction de similarité déduite de $\rho^{\mathcal{AB}}$.

Quoiqu'il en soit, la sémantique de l'indicateur est intéressante : la valeur retournée pour deux colorations $\mathcal{A}$ et $\mathcal{B}$ nous indique le nombre de mouvements nécessaires pour passer de $\mathcal{B}$ à $\mathcal{B}'$ de sorte que $e(\mathcal{A}, \mathcal{B}', \rho^{\mathcal{A}\mathcal{B}'})$ soit nulle, c'est-à-dire que chaque couleur de $\mathcal{A}$ soit contenue dans une couleur de $\mathcal{B}$ (cf propriété 6). L'application $\rho^{\mathcal{A}\mathcal{B}}$ nous donne donc une heuristique pour diminuer le nombre de couleurs ; ce qui peut s'avérer intéressant pour le problème de coloration de graphe.

De plus, nous remarquons aussi que les choix des changements correspondent d'une part au choix d'une application ρ maximale, d'autre part au choix de l'ordre des sommets à recolorer : ces choix sont autant de chemins possibles pour passer de $\mathcal{B}$ à $\mathcal{B}'$.

Enfin, il est possible de construire une métrique au sens mathématique sur l'espace des colorations à partir de $\rho^{\mathcal{A}\mathcal{B}}$.

Définition 4.3 *on définit la distance d_ρ comme suit :*

$$d_\rho(\mathcal{A}_\sim, \mathcal{B}_\sim) = max(e(\mathcal{A}, \mathcal{B}, \rho^{\mathcal{A}\mathcal{B}}), e(\mathcal{A}, \mathcal{B}, \rho^{\mathcal{A},\mathcal{B}}))$$

Nous pouvons vérifier que :
- d_ρ est bien définie ; on l'obtient par les propositions 2 et 3. En effet, les valeurs calculées sont indépendantes du choix des représentants et du choix de $\rho^{\mathcal{A}\mathcal{B}}$.
- Par définition d_ρ est symétrique.
- La propriété 5 nous donne la séparation. En effet, en testant la nullité de l'indicateur e_ρ dans les deux sens, on peut en déduire que les deux colorations sont des raffinements l'une de l'autre. Ceci implique que les deux colorations sont équivalentes.
- L'inégalité triangulaire est conservée :
  Soient $\mathcal{A}$, $\mathcal{B}$ et $\mathcal{C}$ trois colorations,

$$\begin{array}{rll}
\text{on a} & e(\mathcal{A},\mathcal{C},\rho^{\mathcal{A}\mathcal{C}}) & \leq e(\mathcal{A},\mathcal{B},\rho^{\mathcal{A}\mathcal{B}}) + e(\mathcal{B},\mathcal{C},\rho^{\mathcal{B}\mathcal{C}}) \\
\text{donc} & e(\mathcal{C},\mathcal{A},\rho^{\mathcal{C}\mathcal{A}}) & \leq e(\mathcal{C},\mathcal{B},\rho^{\mathcal{C}\mathcal{B}}) + e(\mathcal{B},\mathcal{A},\rho^{\mathcal{B}\mathcal{A}}) \\
\text{et} & max(e(\mathcal{A},\mathcal{C},\rho^{\mathcal{A}\mathcal{C}}), e(\mathcal{C},\mathcal{A},\rho^{\mathcal{C}\mathcal{A}}) & \leq max(e(\mathcal{C},\mathcal{B},\rho^{\mathcal{C}\mathcal{B}}), e(\mathcal{B},\mathcal{C},\rho^{\mathcal{B}\mathcal{C}})) \\
 & & \quad + max(e(\mathcal{A},\mathcal{B},\rho^{\mathcal{A}\mathcal{B}}), e(\mathcal{B},\mathcal{A},\rho^{\mathcal{B}\mathcal{A}})) \\
\text{donc} & d_\rho(\mathcal{A},\mathcal{C}) & \leq d_\rho(\mathcal{A},\mathcal{B}) + d_\rho(\mathcal{B},\mathcal{C})
\end{array}$$

Nous avons défini une seconde métrique sur l'espace des partitionnements et un moyen de la mettre en œuvre : l'algorithme 4 explicite la construction de la matrice de raffinement et établit la mise en correspondance ρ.

Propriété 8 *Pour deux colorations données $\mathcal{A}$ et $\mathcal{B}$, trouver $\rho^{\mathcal{A}\mathcal{B}}$ peut s'effectuer en un temps polynomial.*

La complexité d'un algorithme naïf est en $O(N + \chi(\mathcal{A}) \times \chi(\mathcal{B}))$ (voir algorithme 4).

Algorithme 4 Algorithme de complexité $N + \chi(\mathcal{A}) \times \chi(\mathcal{B})$ pour trouver ρ.

```
for i allant de 1 à χ(A) do
  for j allant de 1 à χ(B) do
    m_{i,j} ← 0 ;
  end for
  for all s ∈ A^{-1}(i) do
    m_{i,B} ← m_{i,B(s)} + 1 ;
  end for
end for
for i allant de 1 à χ(A) do
  vbest ← −1 ;
  for j allant de 1 à χ(B) do
    if m_{i,j} > vbest then
      best ← j ;
      vbest ← m_{i,j} ;
    end if
  end for
  ρ(i) ← best
end for
```

La métrique reposant sur la recherche de $\rho^{\mathcal{AB}}$ diminue de manière heuristique le nombre de couleurs. Ceci peut être un guide heuristique lorsqu'on sait que l'algorithme cherche à minimiser le nombre de classes.

4.3 Classification des instances

Bachelet [Bac99] classifie les instances de QAP (Quadratic Assignment Problem) à l'aide de deux indicateurs statistiques : la variation d'entropie et la variation du diamètre moyen. Afin d'appliquer cette méthode, nous avons du effectuer des modifications. En effet, pour le problème du QAP, on génère une population de configurations initiales par un tirage aléatoire uniforme sur l'espace de recherche. Puis, une recherche locale (déterministe) est appliquée à chaque configuration. On mesure ensuite la variation des indicateurs statistique entre ces deux populations.

Pour le problème de coloration, il est difficile de générer des colorations valides de manière uniforme. La seule coloration, qu'on peut construire en garantissant sa validité, est la coloration triviale qui consiste à attribuer une couleur différente par sommet. Cependant, nous avons vu dans le chapitre précédent que des opérateurs spécifiques au problème de coloration de graphe permettent de modifier une coloration valide en une nouvelle coloration valide dont le nombre de couleurs est inférieur ou égal. De tels opérateurs sont (ou peuvent être rendus) stochastiques. Dans le cas

du problème de coloration de graphe, nous avons donc utilisé les indicateurs d'entropie et de diamètre moyen sur des populations dont chaque coloration est générée en itérant l'application d'un tel opérateur à partir de la coloration triviale.

Pour cette étude, nous avons utilisé la métrique d_ρ présentée précédemment. Nous détaillons ci-après le calcul de l'entropie.

Cette section se décompose en trois parties. Premièrement, nous présentons une matrice de fréquences qui nous permettra de calculer l'entropie d'une population. Puis, nous explicitons le calcul de l'entropie pour le problème de coloration. Dans la troisième partie, nous présentons nos expérimentations et conclusion sur un ensemble de instances de coloration.

4.3.1 Matrice de fréquences

La matrice de fréquences est un outil communément utilisée pour repérer des événements récurents au cours des résolutions de problèmes d'optimisation.

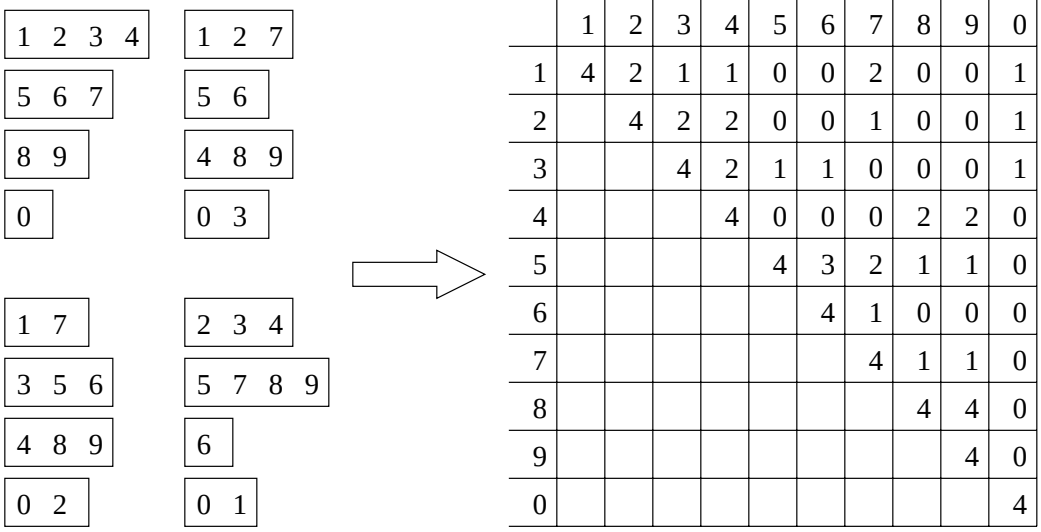
Pour le problème de coloration, il est possible de compter le nombre d'affectations d'une couleur à un sommet. Cependant, nous avons déjà vu de nombreuses fois que la numérotation des couleurs n'est pas intrinsèque au graphe, et encore une fois ceci pose problème. En effet, il est inutile de noter que le sommet s a pris n fois la couleur c, lors de l'exploration des colorations $\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_k$, puisque lors de l'exploration de colorations respectivement équivalentes $\mathcal{B}_1, \mathcal{B}_2, \ldots \mathcal{B}_k$, s peut prendre d'autres couleurs avec des fréquences différentes.

La matrice de fréquences, que nous utilisons, est une matrice $F_{|V|\times|V|}$ qui compte le nombre de fois où deux sommets sont colorés de la même couleur (voir figure 4.13) dans une collection de colorations.

D'un point de vue d'implémentation, une matrice de fréquences n'est pas implémentée comme un tableau bidimensionel. En effet, nous n'utilisons au cours de la résolution uniquement des configurations valides. Il n'est donc pas nécessaire de relever des informations sur les paires de sommets adjacents. En effet, ces sommets sont toujours de couleurs différentes. Il est donc inutile de réserver de la place pour stocker des informations connues d'avance. Nous utilisons donc, pour simuler la matrice de fréquences, un tableau indicé par les paires de sommets non-adjacents. Ce tableau fournit pour chaque paire $p = (i, j)$ la valeur qui aurait figuré dans $f_{i,j}$. On a donc la fréquence d'une paire $p = (i, j)$:

$$F(p) = \frac{f_{i,j}}{N}$$

où N est le nombre de colorations observées.



	1	2	3	4	5	6	7	8	9	0
1	4	2	1	1	0	0	2	0	0	1
2		4	2	2	0	0	1	0	0	1
3			4	2	1	1	0	0	0	1
4				4	0	0	0	2	2	0
5					4	3	2	1	1	0
6						4	1	0	0	0
7							4	1	1	0
8								4	4	0
9									4	0
0										4

Les quatre colorations présentées à gauche de la figure sont réduites dans la matrice de fréquences. La matrice de fréquences étant symétrique, nous avons donc détaillé uniquement la moitié supérieure. La diagonale contient sur chaque élément le nombre de configurations réduites dans la matrice.

FIG. 4.13 – Matrice de fréquences pour la coloration de graphe.

4.3.2 Calcul des indicateurs

Nous pouvons transformer les informations collectées pour obtenir des indicateurs exploitables. Dans la première partie, nous transformons les relevées de fréquences en valeurs d'entropie. Puis, nous présentons une normalisation de la distance pour pouvoir comparer les distances des différents graphes.

4.3.2.1 Entropie

L'objectif est de définir les indicateurs à utiliser, de façon à obtenir une information sur la manière d'appréhender un problème de coloration particulier. Pour cela, nous pouvons calculer l'entropie d'une population à partir de la matrice de fréquence qu'elle engendre. Cette mesure donne directement une information indépendante de la taille du problème, en utilisant la formule suivante :

$$E(p) = -(F(p) \times log(F(p) + (1 - F(p)) \times log(1 - F(p))));$$

Comme précédemment, on impose la validité des colorations observées. Par conséquent, l'entropie des paires de sommets adjacents est nulle. De la même façon que pour la matrice de fréquences, on considère uniquement les paires de sommets non adjacents.

On peut donc calculer la moyenne des entropies comme indicateur de la diversité de l'ensemble des colorations explorées.

4.3.2.2 Distance normalisée

Il nous est aussi nécessaire de modifier légérement les distances définies. Afin de pouvoir calculer des diamètres moyens comparables d'un problème à l'autre. En effet, on constate que plus les problèmes sont de grande taille (i.e. possédant un grand nombre de sommets), plus on a des distances importantes[6]. Nous proposons de normaliser donc la distance en la divisant par le nombre de sommets du graphe. On obtient alors :

$$d'_\star(\mathcal{A}_\sim, \mathcal{B}_\sim) = \frac{1}{N} d_\star(\mathcal{A}_\sim, \mathcal{B}_\sim) \tag{4.8}$$

avec $\star$ pouvant être ρ ou σ et N le nombre de sommets du graphe.

On remarque naturellement que cette normalisation n'affecte en rien les propriétés des distances telles que la symétrie, la séparation et l'inégalité triangulaire.

4.3.3 Expérimentations

Nous avons utilisé des graphes standards disponibles à partir du site de Trick[7]. Ces graphes sont issus des archives du second challenge DIMACS et ont été regroupés en différentes familles selon la manière utilisée pour les générer.

- **DSJ** : graphes générés aléatoirement en spécifiant le nombre de sommets et la densité.
- **CUL** : graphes générés aléatoirement en spécifiant le nombre de sommets et le nombre de couleurs. Il respecte aussi le fait que la variation du degré de chaque sommet est aussi faible que possible.
- **REG** : graphes basés sur des problèmes d'affectation de registres dans des codes réels.
- **LEI** : graphes générés aléatoirement en spécifiant le nombre de sommets et le nombre de couleurs.
- **SCH** : graphes basés sur des problèmes d'emploi du temps.
- **LAT** : graphe basé sur le problème carré latin.
- **BOO** : graphes tiré d'ouvrage de littérature.
- **MIL** : graphes géographique. En fait les sommets représentent des villes américaines. Elle sont reliées par un arc si elle sont suffisamment proche.
- **QUE** : graphes rassemblant toutes les contraintes des problèmes des reines. Ce problème consiste à placer le maximum de reines sur une échiquier $n \times n$ de sorte qu'aucune ne soit en prise.

[6] que ce soit pour d_σ ou d_ρ.
[7] http ://mat.gsia.cmu.edu/COLOR/color.html

– **MYC** : graphes basés sur la transformation de Mycielski.

Afin de classifier les problèmes de coloration, nous avons généré des populations de 1000 colorations en itérant 100 fois l'opérateur "casse" que nous présenterons plus en détail dans la partie 5.1.3.2.2. Les résultats obtenues sont rassemblés dans le tableau 4.1. En fonction des résultats obtenus, on classifie les benchmarks dans trois catégories :

– type (0) avec un diamètre réduit et une entropie élevée.
– type (1) avec un diamètre et une entropie réduits.
– type (2) avec un grand diamètre et une entropie réduite.

Dans le tableau 4.1, nous indiquons également si le problème est "facile" à optimiser pour des algorithmes de recherche utilisant l'opérateur de cette étude. Si c'est le cas le graphe est marqué par un astérisque dans la deuxième colonne.

Nom	facile ?	entropie	E	dist/nb Noeuds	D	Classif	Orig.
myciel3	*	0,628924	g	0,337305455	p	0	MYC
myciel4	*	0,587959	g	0,421524348	p	0	MYC
myciel5	*	0,553345	g	0,483195745	p	0	MYC
myciel6	*	0,524092	g	0,528166316	p	0	MYC
anna	*	0,521305	g	0,457962319	p	0	BOO
myciel7	*	0,500019	g	0,566167539	p	0	MYC
queen5_5	*	0,496417	g	0,401236	p	0	QUE
queen6_6	*	0,473174	g	0,528544444	p	0	QUE
miles250	*	0,464762	g	0,636692188	p	0	MIL
DSJC125.1		0,464112	g	0,6715128	p	0	DSJ
jean	*	0,442858	g	0,709	p	0	BOO
david	*	0,442239	g	0,723143678	p	0	BOO
queen7_7	*	0,440212	g	0,591563265	p	0	QUE
huck	*	0,408825	g	0,556777027	p	0	BOO
queen8_8	*	0,389579	g	0,634732813	p	0	QUE
games120	*	0,384504	g	0,704240833	p	0	GAM
queen9_9		0,348993	g	0,670520988	p	0	QUE
fpsol2.i.3	*	0,347577	g	0,511317647	p	0	REG
DSJR500.1c		0,346624	g	0,265764	p	0	DSJ
queen10_10		0,319681	p	0,696937	p	1	QUE
DSJC125.9		0,31906	p	0,4121712	p	1	DSJ
queen8_12		0,317062	p	0,692884375	p	1	QUE
zeroin.i.3	*	0,311885	p	0,438909223	p	1	REG
le450_5D		0,309982	p	0,59432	p	1	LEI
zeroin.i.2	*	0,309378	p	0,430264455	p	1	REG
le450_5c		0,308174	p	0,596482222	p	1	LEI
mulsol.i.3	*	0,298973	p	0,516967391	p	1	REG
mulsol.i.4	*	0,297612	p	0,518278919	p	1	REG
mulsol.i.2	*	0,295655	p	0,51288617	p	1	REG

Nom	facile ?	entropie	E	dist/nb Noeuds	D	Classif	Orig.
queen11_11		0,293472	p	0,721233884	p	1	QUE
mulsol.i.5	*	0,283336	p	0,507360753	p	1	REG
DSJC125.5		0,278921	p	0,6587744	p	1	DSJ
miles500	*	0,265853	p	0,678928906	p	1	MIL
DSJC250.9		0,24917	p	0,611232	p	1	DSJ
miles750	*	0,185943	p	0,625415625	p	1	MIL
DSJC500.9		0,181289	p	0,6851	p	1	DSJ
school1_nsh		0,165432	p	0,507613636	p	1	SCH
mulsol.i.1	*	0,160454	p	0,321202538	p	1	REG
fpsol2.i.1	*	0,15965	p	0,688546371	p	1	REG
zeroin.i.1	*	0,156852	p	0,278798578	p	1	REG
miles1000	*	0,146828	p	0,574983594	p	1	MIL
school1		0,116513	p	0,259286494	p	1	SCH
miles1500	*	0,104536	p	0,391441406	p	1	MIL
fpsol2.i.2	*	0,337668	p	0,766554324	g	2	REG
DSJC250.1		0,334007	p	0,81886	g	2	DSJ
inithx.i.3	*	0,333522	p	0,808152979	g	2	REG
le450_5a		0,328196	p	0,844844444	g	2	LEI
le450_5b		0,327545	p	0,845135556	g	2	LEI
inithx.i.2	*	0,324045	p	0,801711628	g	2	REG
DSJR500.1		0,315449	p	0,856962	g	2	DSJ
queen12_12		0,271225	p	0,740006944	g	2	QUE
queen13_13		0,255377	p	0,756142012	g	2	QUE
DSJC500.1		0,243953	p	0,866498	g	2	DSJ
queen14_14		0,2407	p	0,771096939	g	2	QUE
le450_15a		0,240562	p	0,865346667	g	2	LEI
le450_15b		0,239899	p	0,864744444	g	2	LEI
queen15_15		0,225598	p	0,783702222	g	2	QUE
inithx.i.1	*	0,220745	p	0,750991898	g	2	REG
le450_25a		0,217486	p	0,858097778	g	2	LEI
queen16_16		0,214643	p	0,794734375	g	2	QUE
le450_25b	*	0,206651	p	0,858708889	g	2	LEI
DSJC250.5		0,1935	p	0,792264	g	2	DSJ
le450_15c		0,184282	p	0,865177778	g	2	LEI
le450_15d		0,18373	p	0,864886667	g	2	LEI
flat300_20_0		0,176619	p	0,80659	g	2	CUL
flat300_28_0		0,173824	p	0,808156667	g	2	CUL
flat300_26_0		0,173146	p	0,808466667	g	2	CUL
le450_25c		0,167029	p	0,862357778	g	2	LEI
DSJC1000.1		0,166168	p	0,903364	g	2	DSJ
le450_25d		0,165723	p	0,863168889	g	2	LEI
latin_square_10		0,12825	p	0,790161111	g	2	LAT

	massif central / 0	multi-massif / 1	uniforme / 2
Δ_{Ent}	Grand	Petit	Petit
Δ_{Dmm}	Petit	Petit	Grand
Schéma			

FIG. 4.14 – Schématisation des trois types de paysages.

Nom	facile ?	entropie	E	dist/nb Noeuds	D	Classif	Orig.
DSJC500.5		0,126469	p	0,834844	g	2	DSJ
DSJC1000.9		0,120684	p	0,738842	g	2	DSJ
DSJC1000.5		0,0799953	p	0,864023	g	2	DSJ
flat1000_50_0		0,0799452	p	0,865261	g	2	CUL
flat1000_60_0		0,0799441	p	0,86501	g	2	CUL
flat1000_76_0		0,0799319	p	0,864808	g	2	CUL
DSJR500.5		0,0547549	p	0,751864	g	2	DSJ

TAB. 4.1: Classification des benchmarks classiques.

Le premier fait que nous pouvons constater est l'existence de problèmes difficiles et de problèmes faciles dans toutes les classes. Cependant, les classes que nous avons définies sont corrélées avec la difficulté du problème. En effet, les problèmes du type (2) semblent plus difficiles à optimiser que ceux du type (0). En cela on peut réutiliser la taxonomie de Bachelet [Bac99] en qualifiant le type (0) de *massif central*, le type (1) de *multi-massif* et le type (2) d'*uniforme*. Par ailleurs, on remarque que les problèmes "flat*" conçus pour être difficiles par Culberson [Cul92] appartiennent effectivement au type (2) (voir figure 4.14).

L'utilisation des critères d'entropie et de diamètre n'ont pas exactement le même sens que dans le travail de Bachelet [Bac99]. Dans son étude, Bachelet mesure les variations d'indicateurs sur les populations. Par conséquent, ces critères (grand/petit) de classification devrait être inversés par rapport à ceux que nous utilisons. Or ce n'est pas le cas de l'entropie. On constate donc que l'entropie possède ici un sens opposé.

Ce phénomène peut en partie s'expliquer par la façon que nous utilisons pour calculer l'entropie. En effet, nous nous intéressons uniquement aux paires de sommets. En particulier, nous regardons s'ils sont réunis dans la même couleur ou non dans les colorations visitées. Or bien qu'il n'y ait qu'une seule façon d'être de la même couleur, il existe plusieurs façons d'être différents. Ainsi, un sommet qui peut prendre de nombreuses couleurs par rapport au reste de la coloration présentera une entropie faible. Réciproquement, un sommet qui ne peut prendre deux couleurs avec

la même probabilité, aura tendance à présenter une entropie forte.

4.4 Conclusion

Dans ce chapitre, nous avons traité la question du paysage des problèmes de coloration. Bien que le problème soit ancien, la symétrie de l'espace de recherche incitait à utiliser des techniques spécifiques. Nous avons développé des outils formels pour supprimer les effets de la symétrie. Ces outils reposent sur la construction de ce que nous appelons une matrice de raffinement de deux colorations. Grâce à cette matrice, nous pouvons calculer deux distances sur l'espace de recherche. Les métriques que nous avons construites peuvent être calculées en temps polynomial. La première d_σ a été contruite de façon à coïncider avec le nombre de mouvements élémentaires pour passer d'une coloration à une autre. Cependant, le temps de calcul nécessaire, nous a poussé a concevoir la seconde métrique d_ρ plus économique. Le sens de d_σ est proche de celui de d_σ mais elle présente un biais qui est intéressant pour le **MGCP** puisqu'elle tend à faire diminuer le nombre de couleurs. Nous avons remarqué au moment de la construction des métriques que nous disposions des pistes pour construire un crossover par la mise en correspondance des parties communes.

Par ailleurs, on peut remarquer encore une fois que ces travaux dépassent le cadre de la coloration de graphe et peuvent plus généralement être utilisés pour tous les problèmes de partitionnement.

Nous avons aussi explicité un moyen de calculer la diversité d'un ensemble de colorations en adaptant le calcul de l'entropie.

Grâce à la combinaison de ces indicateurs, nous avons classifié les problèmes classiques de coloration de graphe en trois catégories : *massif central*, *multi-massif* et *uniforme*. Par ailleurs, nous avons constaté que notre classification est corrélée avec la difficulté intrinsèque des problèmes.

Cependant, nous pensons qu'il est possible d'affiner cette classification en améliorant par exemple le calcul de l'entropie.

Chapitre 5

Approche Parallèle Coopérative pour la coloration de graphe

Ce chapitre présente une façon de concevoir et d'implémenter une métaheuristique évolutionnaire coopérative pour le problème de coloration de graphe. Notre approche a pour objectif d'équilibrer explicitement l'effort de recherche entre les tâches d'intensification et de diversification. Nous précisons les moyens à mettre en œuvre pour sa réalisation. En particulier, nous précisons le fonctionnement de la mémoire adaptative dont le rôle est l'agrégation des informations, afin de mettre en place explicitement l'équilibre entre l'intensification et la diversification.

La mise en œuvre de l'algorithme s'est effectuée à l'aide de la plateforme logicielle **EO***[KMRS01]. Nous en rappelons les principes, puis nous proposons un moyen de modéliser les composants de notre algorithme. Nous avons utilisé cette modélisation pour présenter d'une part une extension à la plateforme facilitant l'hybridation d'algorithmes et d'autre part des assemblages* **EO** *que nous avons effectués lors de notre implémentation des composants de notre approche.*

La force de notre approche réside dans sa conception intrinsèquement parallèle. Nous présentons donc une implémentation distribuée de l'algorithme.

Nous terminons ce chapitre par l'évaluation de performances de notre heuristique et nous constatons la pertinence de deux opérateurs de diversification particuliers.

Nos travaux ont fait l'objet d'une publication dans l'ouvrage Bioinspired Algorithms and Applications : **A Cooperative parallel metaheuristic applied to graph coloring problem***, édité par S. Olariu et A.Y. Zomaya, CRC Press, USA, 2004.*

5.1 Cosearch

Dans cette partie, nous traitons du modèle **Cosearch**. Dans la première partie, nous exposons les objectifs de ce modèle. Puis, dans la seconde partie, nous présentons un ensemble de moyens pour atteindre les objectifs définis. Pour finir, nous verrons dans la troisième partie une application au problème de coloration de graphe.

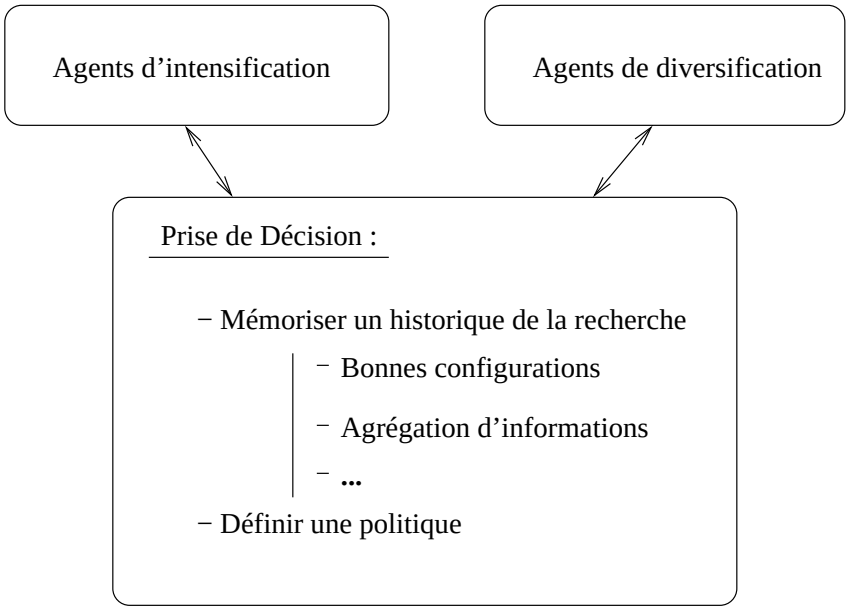
5.1.1 Objectifs de Cosearch

Comme de nombreux problèmes d'optimisation combinatoire, le problème de coloration de graphe (**MGCP**) nécessite pour sa résolution une grande puissance de calcul. Le parallélisme offre un moyen de trouver la puissance nécessaire. Les calculs sont alors distribués sur les processeurs d'une machine parallèle. Cette distribution peut permettre de nouveaux schémas d'exécution en spécialisant les différents traitements sur les différents processeurs. Par exemple, nous pouvons imaginer un algorithme itérant la distribution de configurations initiales à des algorithmes de recherche taboue évoluant en parallèle avec des paramètres différents, puis collectant les résultats avant une nouvelle phase parallèle. Au moment de la synchronisation des résultats une coopération est possible, par exemple en fournissant des points de départ aux différentes recherches taboues en fonction de l'ensemble des résultats collectés.

Dans ce cadre, **Cosearch** est un "méta-modèle" de résolution coopérative parallèle. Ce méta-modèle a permis dans [Bac99] de développer une heuristique robuste pour le problème d'affectation quadratique. **Cosearch** a pour principe de base d'équilibrer l'exploitation et l'exploration de l'espace de recherche. Ces deux tâches sont effectuées respectivement par les agents d'intensification et de diversification (voir figure 5.1). Nous espèrons ainsi avoir une vision plus globale de l'espace de recherche et de ce fait améliorer la qualité des solutions fournies. Une mémoire dite *adaptative* équilibre explicitement les deux tâches, en s'appuyant sur l'ensemble des informations qu'elle collecte au cours du déroulement de l'algorithme. Cette mémoire constitue donc une pièce centrale de la méthode : les communications directes entre agents sont interdites, seule la mémoire adaptative possède la connaissance de leurs existences.

5.1.2 Présentation de Cosearch

À ce niveau d'abstraction, de nombreuses heuristiques implémentent **Cosearch**. En effet, trop peu d'éléments sont précisés (que ce soit l'importance des différents agents, les mécanismes utilisés par ces agents, les informations mémorisées dans l'historique ou la politique suivie). Par exemple, une recherche taboue peut implémenter



Le modèle général **Cosearch** *décrit les grandes lignes de l'heuristique. Des agents d'intensification cherchent à améliorer la qualité globale de la meilleure configuration trouvée jusqu'alors. Des agents de diversification cherchent à garantir une bonne exploration de l'espace de recherche. L'équilibre entre la diversification et l'intensification est effectué au niveau de la boîte centrale. Elle est amenée à prendre des décisions. Pour cela, il est nécessaire de définir ce qui est pertinent d'observer, ainsi que la politique à appliquer au vu des critères observés.*

FIG. 5.1 – Principes de **Cosearch**.

ce modèle : l'intensification étant assurée par la recherche de meilleures configurations dans le voisinage de la configuration courante ; la diversification étant assurée par la gestion de la liste taboue interdisant certains mouvements dans l'espace de recherche. Peut être encore plus caricatural, la méthode de descente peut être vue comme une instance de **Cosearch** en considérant vide la tâche de diversification.

Cosearch est une modèle hybride de haut niveau, c'est-à-dire que nous devons définir des interactions entre des heuristiques qui résolvent des problèmes complets. Ces interactions sont réalisées de manières différentes selon qu'il s'agisse de la tâche d'intensification ou de celle de diversification. Ces informations sont collectées dans ce que nous avons présenté comme la mémoire adaptative. La mémoire adaptative est en quelque sorte un tableau blanc permettant une certaine forme d'échange d'informations entre les agents. Cependant, cette mémoire est la pièce maîtresse de **Cosearch** car elle pilote grâce aux informations collectées les autres composants de l'application. Elle contient :

– Une population de configurations rassemblant un ensemble de configurations

parmi les meilleures que nous ayons découvertes au cours de la recherche. Ces configurations indiquent une zone de l'espace de recherche jugée prometteuse.
- Des matrices de fréquences et/ou d'autres informations réduites ou transformées. Ces informations peuvent avoir un codage spécifique pour chaque problème étudié. Une matrice de fréquences a pour objectif d'agréger les informations d'un nombre élevé de configurations, puisqu'il n'est pas possible de toutes les stocker. Ainsi, il est possible d'avoir une idée globale des zones visitées de l'espace de recherche.

En plus de la mémoire adaptative, **Cosearch** intègre deux composants dont nous suggérons les grandes lignes de leurs comportements :
- **Des agents d'intensification** qui améliorent les meilleures configurations obtenues jusqu'alors. Ces agents peuvent être implémentés par différentes heuristiques. Cependant, il paraît naturel d'utiliser une approche basée sur une recherche locale car ces algorithmes sont peu coûteux en temps de calcul. Nous pouvons espérer trouver que les meilleures configurations résolvant le problème présentent des caractéristiques communes avec les meilleures configurations trouvées.
- **Des agents de diversification** qui génèrent des configurations présentant des caractères différents de ceux observés. Là encore des agents peuvent être implémentés de différentes façons. Malheureusement, il n'existe pas d'idée intuitive de ce qui est une bonne diversification. De plus, la phase de diversification ne doit pas détruire les apports des autres éléments de l'algorithme. Nous proposons donc de perturber les meilleures configurations en ajoutant des caractéristiques provenant de régions peu ou insuffisamment explorées.

La figure 5.2 illustre notre proposition.

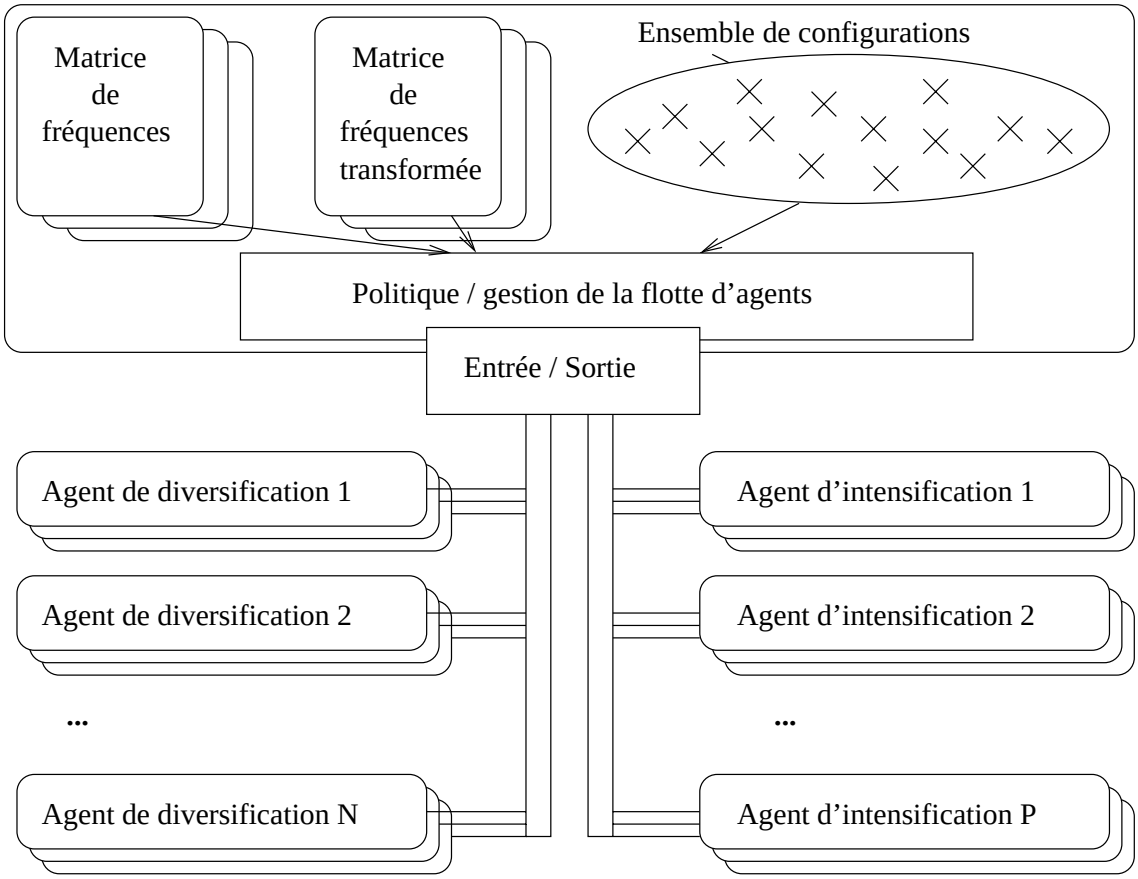
5.1.3 Approche Cosearch pour la coloration de graphe

Cette section se décompose en trois parties. Dans la première, nous présentons la mémoire adaptative pour la coloration. Dans la seconde partie, nous présentons trois agents d'intensification. La troisième partie traite des agents de diversification.

5.1.3.1 La mémoire

En plus de contenir une collection de configurations de bonne qualité, la mémoire adaptative doit enregistrer des informations supplémentaires pour détecter les régions insuffisamment explorées. Premièrement, la collection de configurations que nous utilisons est un ensemble sans doublons. Nous appelons cet ensemble *l'ensemble élite*. Nous utilisons pour cela le test d'égalité vu en 3.2.2.

D'autre part, nous mesurons le temps écoulé depuis la dernière exploration d'une région de l'espace de recherche. Pour cela, nous utilisons deux tableaux A et B indicés



La mémoire adaptative stocke un ensemble de configurations représentant des points de départ potentiels (en haut à droite) et des informations réduites et/ou transformées à l'aide de matrice de fréquences et/ou des transformations par des fonctions d'entropie (en haut à gauche). En fonction de ces informations, des décisions sont prises pour gérer au mieux l'ensemble des agents (boîtes au dessous des E/S) évoluant en parallèle. Une même heuristique peut calculer de nouvelles configurations à partir de points initiaux différents. Différentes heuristiques peuvent être utilisées pour la même tâche (intensification ou diversification).

FIG. 5.2 – Schéma du modèle **Cosearch**.

par les paires de sommets non-adjacents. L'exploration de l'espace de recherche est considérée comme une séquence des colorations visitées. À chaque fois qu'une coloration est reçue par la mémoire adaptative à une date t, nous mettons à jour les tableaux A et B de la manière suivante :

$$A(s_1, s_2) = \begin{cases} A(s_1, s_2) \text{ si } \mathcal{C}(s_1) = \mathcal{C}(s_2) \\ t \text{ sinon} \end{cases} \qquad B(s_1, s_2) = \begin{cases} t \text{ si } \mathcal{C}(s_1) = \mathcal{C}(s_2) \\ B(s_1, s_2) \text{ sinon} \end{cases}$$

Après quoi, nous incrémentons t et nous traitons la coloration suivante. Au debut de l'algorithme, la date courante et toutes les cases de ces tableaux sont initialisées à zéro. Nous pouvons ainsi détecter que des sommets sont réunis (A) ou séparés (B) depuis (trop) longtemps au cours de la recherche.

5.1.3.2 Les composants d'intensification

Dans cette partie, nous présentons les différents opérateurs d'intensification que nous avons développés. Ils présentent la particularité de transformer une coloration en une autre en garantissant que le nombre de couleurs utilisées n'augmente pas. Ces opérateurs peuvent être considérés comme complexes car ils consistent à changer la couleur de plusieurs sommets. Nous avons pris le parti de traiter le **MGCP** directement, c'est-à-dire sans itérer des résolutions du **GkCP** avec k variant mais en considérant uniquement les colorations valides et en autorisant des nombres de couleurs différentes. Pour cela, nous avons implémenté un certain nombre de composants élémentaires que nous avons ensuite combinés dans notre métaheuristique hybride parallèle.

Nous proposons trois agents d'intensification basés chacun sur un des trois opérateurs suivants :

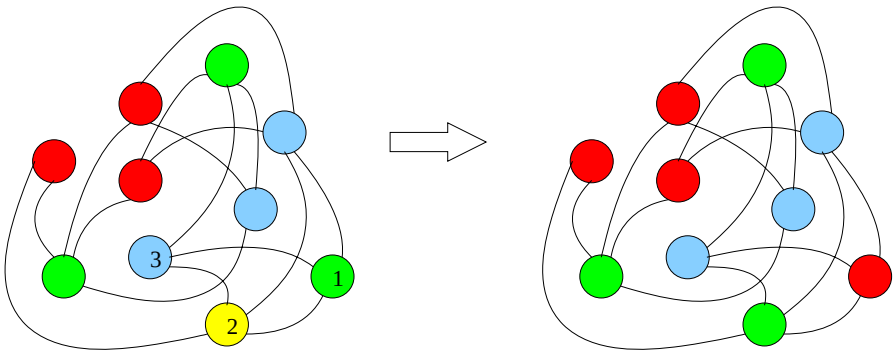
- **sature** basé sur l'heuristique gloutonne RLF et IG (voir 3.3.1) ;
- **casse** un nouvel opérateur dédié à la coloration de graphe agissant à l'inverse du précédent ;
- **casse sature** un opérateur combinant les deux approches précédentes.

5.1.3.2.1 Opérateur sature Nous avons repris le principe de *Iterated Greedy* (voir section 3.3.1). Pour construire une nouvelle coloration $\mathcal{C}'$ à partir d'une coloration $\mathcal{C}$, nous parcourons toutes les classes de $\mathcal{C}$ et essayons d'augmenter (saturer) chacune d'entre elles. en y ajoutant un maximum de sommets (voir algorithme 5 et figure 5.3).

Contrairement à la méthode Iterated Greedy de Culberson [Cul92], cette technique est rendue stochastique en mélangeant l'ensemble des sommets non (re-)colorés lorsque nous complètons une classe. Ceci permet de ne pas privilégier certaines paires et ainsi d'obtenir une meilleure exploration par cet opérateur. Les colorations construites par cet opérateur sont dites "saturées". Opérer uniquement sur les colorations saturées réduit l'espace de recherche.

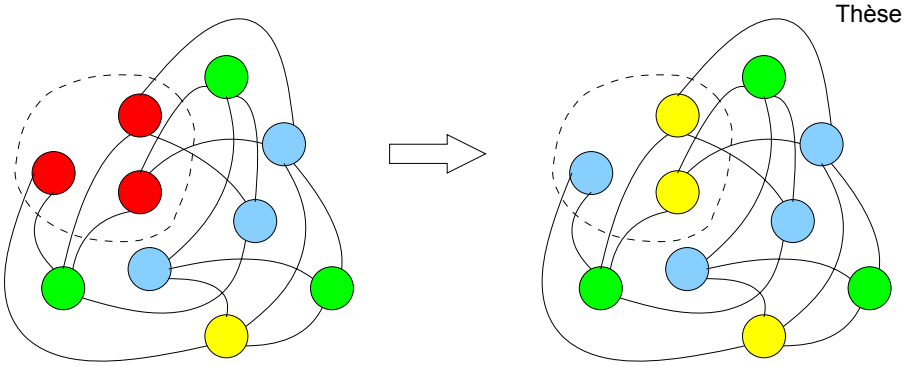
Algorithme 5 Opérateur de Saturation itéré

Require: $\mathcal{C}$ la coloration courante
  $\mathcal{C}'$ a initialement aucun sommet coloré ;
  Définir un ordre de parcours des classes de $\mathcal{C}$;
  for all classe i de $\mathcal{C}$ **do**
    $\mathcal{C}'^{-1}(i) = \mathcal{C}^{-1}(i) \bigcap \{$les sommets non colorés de $\mathcal{C}'\}$.
    Compléter $\mathcal{C}'^{-1}(i)$ avec les sommets non colorés de $\mathcal{C}'$.
  end for



Les classes sont saturées dans l'ordre rouge, vert, bleu, puis jaune. Le sommet 1 *est le premier sommet qu'on puisse colorer en rouge (le sommet* 3 *n'est donc pas coloré en rouge). Puis nous saturons la couleur verte, le sommet* 2 *peut maintenant être coloré en vert. La saturation de la classe bleue n'apporte pas de changement à la configuration.*

FIG. 5.3 – Exemple de saturation de classes.



Les classes sont cassées dans l'ordre rouge, vert, bleu, puis jaune. Les sommets de la première classe sont répartis dans les autres classes. Après cette étape, les tentatives de casse des autres classes se soldent par des échecs.

FIG. 5.4 – Exemple de casse d'une classe.

5.1.3.2.2 L'opérateur Casse Cet opérateur a été conçu pour fonctionner à l'opposé de l'opérateur *sature*. En effet, l'opérateur de saturation cherche à augmenter la taille des classes, ce qui entraîne une diminution du nombre de couleurs. À l'inverse, nous pouvons essayer de gagner directement une couleur en distribuant les sommets d'une classe dans les autres classes.

L'opérateur Casse consiste à parcourir les classes d'une coloration et de tenter de les détruire en plaçant les sommets les composants dans les autres classes (voir algorithme 6 et figure 5.4). Nous essayons donc de changer la couleur d'un maximum de sommets dans une classe. Les sommets restants forment une classe "incassable" (relativement au reste de la coloration).

Dans cet algorithme, nous constatons que différents ensembles sont parcourus tel que l'ensemble des classes à casser, les sommets de la classe courante et les classes susceptibles de recevoir les sommets. L'ordre de parcours de chaque ensemble peut être important. Dans le cas présent, la première boucle (ligne 1) parcourt les classes dans un ordre aléatoire uniforme, afin de permettre une meilleure exploration de l'espace de recherche. L'ordre de parcours des sommets de la seconde boucle n'a pas d'incidence réelle. En effet, les sommets appartenant à une classe que nous souhaitons casser sont non-adjacents deux à deux. Par conséquent, changer la couleur de l'un d'entre eux ne change pas les contraintes sur les autres sommets de la classe. La troisième boucle parcourt les classes dans le même ordre que la première. Ainsi nous placons en priorité les sommets dans les classes qui n'ont pas pu être cassées, puisqu'elles sont considérées comme incassables par rapport à la coloration courante.

5.1.3.2.3 L'opérateur Casse sature Cette heuristique reprend et combine les idées des deux précédents opérateurs. En effet, quand une classe est déclarée incas-

Algorithme 6 Opérateur de casse itéré.

Require: $\mathcal{C}$ la coloration courante

for all classe i de $\mathcal{C}$ **do**
  for all sommet s de $\mathcal{C}^{-1}(i)$ **do**
    for all classe j de $\mathcal{C} \neq i$ **do**
      if s est colorable en j **then**
        colorer s en j ;
        break ;
      end if
    end for
  end for
end for

sable (par rapport au reste de la coloration), il est peut être intéressant de la saturer par des sommets des classes non encore cassées. Cela revient à chercher l'optimum parmi les colorations saturées (comme pour l'opérateur de saturation). L'algorithme 7 explicite le fonctionnement de cet opérateur.

Algorithme 7 Opérateur casse sature itéré.

Require: $\mathcal{C}$ la coloration courante

for all classe i de $\mathcal{C}$ **do**
  for all sommet s de $\mathcal{C}^{-1}(j)$ **do**
    for all classe j de $\mathcal{C} \neq i$ **do**
      if s est colorable en j **then**
        colorer s en j ;
        break ;
      end if
    end for
  end for
  if $\mathcal{C}^{-1}(i)$ est non vide **then**
    Compléter $\mathcal{C}'^{-1}(i)$ avec des sommets des classes suivantes.
  end if
end for

5.1.3.3 Les composants de diversification

Les opérateurs d'intensification peuvent être bloqués dans un optimum local et la façon dont nous les utilisons ne nous permet pas de détecter quand leur application devient inutile. Nous proposons quatre moyens de sortir d'un optimum. Nous utilisons pour cela des opérateurs de diversification. Nous proposons deux opérateurs unaires (fonctionnant comme une mutation) et deux opérateurs quaternaires (fonctionnant à la manière d'un crossover).

5.1.3.3.1 Les opérateurs unaires Les opérateurs unaires que nous avons développés forcent la recherche de solutions dans un certain "schéma". Ainsi, si une région de l'espace de recherche a été délaissée alors ces opérateurs y imposent une recherche. Nous imposons aux colorations de présenter deux types de caractères :

- Forcer deux sommets à être de couleurs différentes (**FD**). Pour une coloration donnée, nous commençons par placer les deux sommets dans des classes différentes en ajoutant, le cas échéant, une couleur supplémentaire. Puis nous appliquons l'opérateur de casse vu en 5.1.3.2.2 en interdisant aux deux sommets d'être colorés de la même couleur. Pour cela, nous ajoutons un arc fictif entre ces deux sommets[1].
- Forcer deux sommets à être de la même couleur (**FM**). Nous commencons par placer les deux sommets dans une même classe initialement vide, puis nous complètons cette coloration partielle par un algorithme de saturation dont l'ordre des sommets est donné par l'une des meilleures colorations trouvées. Ensuite, nous appliquons l'opérateur de casse vu en 5.1.3.2.2 en interdisant aux deux sommets d'être colorés par des couleurs différentes. Pour cela, nous fusionnons les sommets : ils forment en quelque sorte un nouveau sommet dont l'ensemble des voisins est l'union des voisins des deux sommets initiaux[2].

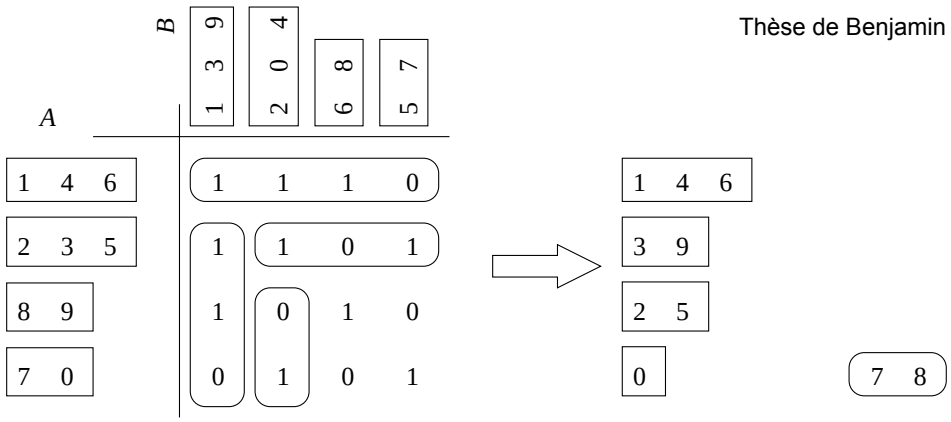
Remarquons que les caractères ainsi utilisés ne souffrent pas de la symétrie du problème et peuvent donc être utilisés sur n'importe quelle représentation d'un partitionnement.

5.1.3.3.2 Les opérateurs quaternaires Les opérateurs de recombinaison ont rarement été utilisés pour la résolution du **MGCP** et la stratégie utilisée consiste le plus souvent à résoudre plusieurs **GkCP**. Comme nous nous sommes intéressés à une approche directe du **MGCP**, nous proposons les deux croisements suivants. Le premier est inspiré de GPX [GH99], que nous revisitons à l'aide des résultats sur la matrice de raffinement. Puis, nous présentons un autre crossover directement inspiré du calcul de la distance.

L'opérateur GPX [GH99]

Initialement, le crossover GPX a été développé pour résoudre le **GkCP**. En effet, il opère sur des colorations possédant un même nombre de couleurs et pouvant présenter des violations de contrainte. Nous présentons une version de l'opérateur traitant des colorations valides avec un nombre variable de couleurs. En fait, nous constatons que le critère d'arrêt sur k est arbitraire quand le nombre de couleurs est autorisé à varier. Il est par conséquent possible de continuer à construire des nouvelles classes tant qu'il y a des sommets non colorés. Nous constatons également que si les deux colorations parentes sont valides alors la coloration résultante le sera également. Si nous observons le comportement de GPX au niveau de la matrice

[1] L'arc est retiré après l'application de cet agent.
[2] De la même façon la fusion cesse à la fin de l'algorithme.



GPX prend alternativement une classe du parent $\mathcal{A}$ puis de $\mathcal{B}$. La classe choisie est celle qui a le plus de sommets non-colorés dans l'enfant. Lorsque k (ici 4) classes sont construites dans l'enfant, nous colorons les sommets restants au hasard (pour plus de diversité).

FIG. 5.5 – Illustration de GPX vu par rapport à la matrice de raffinement.

de raffinement, nous constatons que nous colorons les sommets sur une ligne puis sur une colonne, en choisissant la ligne (resp. la colonne ayant le plus d'éléments non-colorés) (voir figure 5.5).

À partir de GPX, nous proposons une version qui ajoute une heuristique pour la construction des classes : après avoir ajouté les sommets non-colorés dans la coloration fille d'une classe d'un parent, nous saturons la classe par des sommets non-colorés dans la coloration fille. L'ordre d'examen des sommets complétant la classe utilise une permutation générée aléatoirement comme dans la méthode de saturation (cf 5.1.3.2.1).

L'opérateur $\mathbf{X}_\rho$

L'opérateur de croisement, que nous proposons, est basé sur la reconnaissance des parties communes (cf section 4.2). Cet opérateur commence par construire une coloration partielle exhibant les parties communes aux deux colorations parentes. Ensuite, nous complètons cette coloration par une procédure gloutonne. Comme la mise en relation σ (basée sur la bijection) impose que le nombre de couleurs des enfants soient au moins aussi grand que celui des parents, nous l'avons écarté[3]. Nous avons donc utilisé la mise en relation basée sur ρ (voir 4.2.3).

Cet opérateur (pour X_ρ) consiste donc à générer deux enfants e_1 et e_2, à les compléter en saturant les premières classes à la manière de l'opérateur de saturation puis à continuer la construction de la coloration par ajout de classes saturées. L'ordre des sommets utilisés pour la complétion de e_1 (resp. e_2) est donné par p_1 (resp. p_2) :

[3]De plus le temps de calcul est plus important que ρ.

les sommets non colorés dans e_1 appartenant à la même classe de p_1 sont regroupés dans la permutation.

5.2 Implémentation

Cette section se décompose en deux parties. Nous exposons d'abord la plateforme logicielle **EO**. Nous proposons d'ailleurs une schématisation en "boite de verre" pour cette plateforme et une extension facilitant l'hybridation d'algorithmes. La seconde partie traite de la parallélisation de **Cosearch** pour la coloration de graphe.

5.2.1 Plateforme logicielle EO

La conception de métaheuristiques est de plus en plus souvent une affaire d'hybridation de différents schémas "classiques". Il est fréquent de voir des **AG** dont certains opérateurs ont été changés, par exemple en remplaçant la mutation par une recherche locale. Malgré cela, de grandes lignes restent communes et lors de la conception d'une heuristique il est intéressant de pouvoir réutiliser des composants classiques en redéveloppant le moins possible.

Il existe de nombreux projets fournissant du matériel logiciel pour l'implémentation de métaheuristiques. On peut par exemple citer la bibliothèque Localizer++ [MH01] ou la plateforme EasyLocal++ [GS03]. Ces deux produits traitent de méthodes de résolution par recherche locale mais avec des points de vue très différents. La première est une bibliothèque, c'est à dire que le code de l'utilisateur appelle les fonctionnalités du produit par des routines "protégées". La seconde est une plateforme. Dans ce cas, le contrôle est inversé et la plateforme appelle le code de l'utilisateur.

Pour notre part, notre choix s'est porté sur la plateforme **EO**[KMRS01]. Cette plateforme répond à quatre objectifs principaux :

- **Réutilisabilité** : au moment de la conception d'un algorithme, de nombreux mécanismes classiques peuvent être réutilisés. Il est intéressant pour le programmeur de pouvoir les réutiliser.
- **Extensibilité** : outre un gain de temps sur la phase de développement et de debuggage, on observe également un gain d'efficacité au niveau du prototypage des composants. De plus, la plateforme étant "open source", de nouveaux composants facilitant les développements des algorithmes de chacun peuvent être intégrés.
- **Flexibilité** : la philosophie de **EO** (voir partie 5.2.1.1) propose une décomposition quasi-atomique des tâches à effectuer.
- **Adaptabilité** : la possibilité d'intégrer ses propres composants permet de combiner de mécanismes génériques (sélections, remplacements ...) avec des

mécanismes dédiés à l'application (codages, opérateurs ...).
À l'origine **EO** est exclusivement dédiée aux algorithmes évolutionnaires. Elle a été ensuite étendue pour implémenter des heuristiques locales, parallèles et distribuées dans le cadre de la thèse de Cahon [CMTS03, CMT04]. Dans ce cadre, nous proposons l'ajout de composants apportant plus de souplesse pour l'hybridation.

Dans une première partie, nous rappellons la "philosophie **EO**", c'est-à-dire les principes de base à respecter pour une bonne intégration de nouveaux composants à la plateforme. Dans la deuxième partie, nous établissons les notations nécessaires ainsi qu'une nomenclature permettant de schématiser "simplement" les composants et composés **EO**. La troisième partie présente un ensemble de composants que nous avons ajouté et discute de leurs avantages en terme de la flexibilité.

5.2.1.1 Philosophie d'EO

La présentation de la plateforme, que nous faisons ci-après, reprend les grandes lignes de celle de Cahon *et al.*[CMTS03]. Un objectif d'**EO** est de permettre le développement de prototypes d'heuristiques pour de larges classes de problèmes. De ce fait, les éventuelles extensions à cette plateforme doivent permettre un maximum de réutilisabilité des composants en offrant la possibilité d'appliquer les nouveaux composants pour la résolution de nombreux problèmes. C'est pour cela qu'**EO** a été développé en C++. En particulier, **EO** utilise largement le mécanisme de "template" pour implémenter son "framework" principalement grâce aux deux fonctionnalités suivantes :

- En utilisant les templates de manière vide, **EO** spécifie des classes avec un certain savoir-faire. Nous appellerons ces classes génériques des interfaces. Ces interfaces peuvent être implémentées soit par **EO**, soit par un utilisateur. Une nouvelle implémentation (fournie par **EO** ou développée par un utilisateur) d'une interface fournit un nouveau comportement spécifique. Nous pouvons, par exemple, citer le cas de la classe eoContinue qui spécifie toutes les classes dont les objets servent de critère de continuation pour un algorithme. En d'autres termes, une instance d'une classe héritante de eoContinue retourne "Vrai" pour signifier que l'algorithme doit continuer et "Faux" quand il doit s'arrêter.
- Les autres classes templates ont un usage plus conventionnel. Ces classes templates contiennent du code utilisable pour la résolution de différents problèmes. Elles héritent d'interfaces et n'ont pas pour vocation d'avoir des classes filles développées par un utilisateur. Par exemple, la classe eoGenContinue spécifie un opérateur de continuation retournant "Vrai" tant qu'un nombre d'itérations n'est pas atteint.

EO fournit donc une grande hiérarchie de classes (près de 300 classes). Bien qu'il existe une documentation, qui est générée automatiquement par doxygen[4], elle reste relativement hermétique. En effet, cette dernière propose uniquement la hiérarchie de classes. Ces informations sont indispensables à l'extension de la plateforme mais très peu pratiques pour son utilisation. De plus, une grande partie de la hiérarchie de classes n'a pas vraiment besoin d'être connue par l'utilisateur. Nous proposons donc un moyen intéressant de documenter **EO** à l'aide de schémas.

5.2.1.2 Notation et schématisation

Nous présentons ici une façon simple de visualiser les compositions de métaheuristiques possibles avec **EO**, grâce à une schématisation à base de boîtes et de flèches. Certains mécanismes sont décrits de façon similaire à ce que ferait UML [BRJ99]. Cependant, nous n'avons pas besoin de toute la richesse de ce langage. Nous détaillons ici uniquement ce qui nous est nécessaire en y incluant quelques personnalisations. En particulier, nous ajoutons un code de couleurs afin de visualiser d'un simple coup d'œil les tâches incombant à un utilisateur.

Cette section est découpée en deux parties. Premièrement, nous exposons les choix que nous avons retenus ; puis dans un second temps nous les discutons.

5.2.1.2.1 Choix retenus Nous distingons trois sortes[5] de composants. À chaque sorte de composants est attribuée une couleur parmi vert, rouge et orange.

- **les composants de stratégies** : ils contiennent généralement du code. Ils sont regroupés par fonctionnalités grâce à des interfaces de cette catégorie (colorés en vert). Par exemple, nous pouvons citer la classe eoSelect qui spécifie le comportement de sélection ; les classes eoDetSelect, eoDummySelect, eoSelectMany, eoSelectNumber et eoSelectPerc possèdent les codes pour des types de sélection spécifique. Un utilisateur peut très bien ne jamais développer de composant de cette sorte et utiliser ceux fournis par **EO** ; ils sont colorés en vert car aucun travail n'est demandé par ces composants.
- **les composants liés aux applications** : Ces composants sont généralement à définir par l'utilisateur et dépendent du problème à résoudre. Dans ce cas, nous pouvons citer, par exemple, le codage des configurations comme les permutations pour les problèmes d'affectation quadratique et du voyageur de commerce, ou des codages plus complexes dans des problèmes de coloration ou d'élaboration de tournées de véhicules. La conception d'un nouveau codage pour les configurations d'un problème spécifique implique la conception de nouveaux opérateurs opérant sur ces configurations. Nous les colorons en rouge.

[4] http ://www.stack.nl/~dimitri/doxygen/index.html
[5] nous ne parlons pas de type car la notion existe dans un autre sens en C/C++.

– **Les composants mixtes** : ils proposent des assemblages cohérents des composants de chacune des deux catégories précédentes. Cet assemblage est effectué grâce aux constructeurs. Ces composants possèdent de l'algorithmique pour activer ces composants de manière contrôlée. Dans cette catégorie, nous trouvons par exemple la classe eoEasyEA qui est le MÉTA algorithme évolutionnaire de la plate-forme et la classe eoSGATransform permettant d'obtenir les fonctionnalités des transformations classiques dans un algorithme génétique. Là aussi, des interfaces sont utilisées pour spécifier les fonctionnalités des composants. Par exemple, la classe eoTransform appartient à cette catégorie car les classes filles seront forcément un mélange de stratégies et de composants dédiés à une application. Ces composants appartiennent à la catégorie orange car le travail de l'utilisateur consiste à définir le meilleur assemblage possible.

Une grande partie des composants **EO** est "templatisée" par EOT. Ce type doit représenter la configuration du problème. En fait, EOT apparaît dans tous les composants dédiés à l'utilisateur. Nous avons convenu de ne pas surcharger les schémas en ne faisant pas figurer ce paramètre de généricité. Cependant, des classes utilisent des paramètres génériques supplémentaires. Ces classes n'existaient pas dans la première version de **EO**. Puisqu'il s'agit d'un apport, nous mentionnons explicitement l'emploi de ces paramètres par soucis de transparence. Les paramètres génériques d'un composant sont représentés dans une boite bleue en haut à droite du composant.

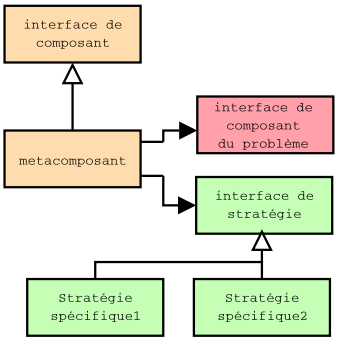
Signalons que l'utilisation des templates C++ ne permet pas de spécifier le type des paramètres de généricité de l'objet. Il appartient donc à l'utilisateur de fournir des paramètres templates cohérents. En particulier, le paramètre EOT doit être une classe héritant de la classe EO[6].

Pour le reste, nous utilisons deux types de relations entre les classes : l'agrégation et l'héritage (voir figure 5.6).

Nous utilisons des flèches pleines pour montrer que les composants à l'origine des flèches agrègent les composants pointés par ces flèches. L'agrégation utilise le mécanisme de référence. Ainsi, le composé possède une vue sur les objets qui le composent. Par conséquent, deux composés peuvent avoir chacun une vue sur un même composant. Dans ce cas, nous disons que le composant est partagé entre les composés.

Certains composants possèdent une collection de composants ; dans ce cas, la cardinalité de la relation est indiquée au dessus de la flèche par une relation $n..m$. Afin de faciliter la lecture, nous nous arrangerons, si cela est possible, pour placer les composants à droite du composé qui les agrège.

[6]cette classe porte le même nom que la plateforme de par la place centrale qu'elle occupe.



Cette figure représente un schéma possible pour des composants **EO**. *Une métaheuristique ou un métacomposant est spécifié par son interface. Il nécessite deux composants, le premier dépendant du problème, le second étant une stratégie générique dont deux formes spécialisées existent déjà dans les bibliothèques* **EO**.

FIG. 5.6 – Exemple de schémas.

Dans nos schémas, il est intéressant de faire aussi figurer la hiérarchie de classes. En effet, les composants d'un composé sont des interfaces spécifiant des savoir-faires. Plusieurs combinaisons sont alors possibles lors de l'instanciation. Nous utiliserons alors une flèche évidée pointant du côté de la classe mère.

Les derniers points à signaler se situent au niveau de la distinction entre les schémas de classes et les schémas d'instances. La distinction se fait au niveau des agrégations des composants. Dans **EO**, l'agrégation s'effectue par référence donc lors d'une instanciation d'un schéma de classe la relation d'appartenance n'indique pas si un composant est partagé par deux composés ou au contraire s'il existe deux composants répartis dans les deux composés. Si le partage de composants est nécessaire à une application, nous suggèrons qu'il soit résolu de la façon suivante : nous créerons un composant "de type orange" qui forcera par son constructeur le partage de composants nécessaire. Par exemple, la température du recuit simulé sert à la fois pour le critère d'arrêt et le critère d'acceptation de la configuration voisine. Par conséquent, le constructeur du recuit simulé consiste à effectuer l'assemblage d'une recherche locale avec le même objet température pour le critère acceptation et le "continuateur".

5.2.1.2.2 Discussion sur la schématisation Dans cette partie, nous discutons un certain nombre de choix faits pour cette schématisation. Ils sont présentés du moins important au plus important.

Le choix de couleurs montre la graduation du travail nécessaire à l'utilisateur :
- **vert**, l'utilisateur utilise le composant. Seul quelqu'un souhaitant augmenter la plateforme développe des composants verts ;

- **orange**, l'utilisateur instancie la classe pour composer sa propre heuristique. Sa tâche est alors un choix des composants de l'instance ;
- **rouge**, l'utilisateur doit d'une part bien connaître son problème et d'autre par développer un composant ayant cette interface s'il souhaite obtenir des résultats de bonne qualité ;
- **bleu**, nous utilisons cette couleur pour mettre en évidence une contrainte que l'utilisateur doit respecter.

Dans un schéma d'instance, le sens de la flèche d'agrégation est plus fort. Si un objet est partagé, il est pointé par deux flèches. Dans le schéma de classes, le sens étant plus faible, nous avons décidé de laisser une certaine souplesse à ce niveau et ceci afin de permettre une meilleure lisibilité.

Cette schématisation présente à la fois la hiérarchie de classes et les schémas de composition. Elle permet de visualiser facilement les schémas d'heuristiques évolutionnaires possibles avec la plateforme. Cependant, cet avantage peut dans le même temps être un défaut. En effet, les schémas risquent de vite devenir de taille importante. Il sera alors nécessaire de faire des coupes dans les schémas "trop globaux". Nous essayerons tant que faire se peut de couper les schémas de classe au niveau des interfaces (la figure 5.7 est déjà une restriction du schéma global d'un AEH et il a été conçu sur une feuille A2).

La plupart des composants **EO** ne possèdent qu'une seule méthode en plus de son (ou ses) constructeur(s). Nous appelons cette méthode "parenthèses" car elle surcharge un opérateur particulier du C++. Contrairement à la syntaxe habituelle en C++ (nomDObject.nomDeMethode(paramètres)), lors d'un appel à cette méthode, nous n'a pas besoin du point et d'un nom de méthode ; il suffit de fournir les paramètres à l'objet. Par exemple, le code qui suit montre la construction d'un **AG**, utilisant les objets maSelection, monQuadOp, monMonOp, monEvaluateur et monContinuateur comme sélection, opérateurs génétiques, évaluateur et test de continuation. L'instruction suivante effectue l'exécution de l'**AG** sur une population d'objets de la classe T_conf.

```
/* ... */
eoEasySGA<T_conf> monAG(maSelection,
                        monQuadOp,
                        monMonOp,
                        monEvaluateur,
                        monContinuateur);
monAG(unePopulation);
/* ... */
```

Remarquons ici que le paramètre générique sert à faire un contrôle de type : un **AG** travaillant sur des "T_conf" ne pourra être construit qu'avec des composants ayant le même paramètre template. De même, l'exécution se fera ensuite uniquement sur une population d'objets de type T_conf.

Cependant, certaines classes ne respectent pas ce principe et possèdent d'autres méthodes. Quoiqu'il en soit la manipulation directe des méthodes d'une classe n'est effectuée que très rarement par un utilisateur. En général, son rôle consiste à spécifier une composition en instanciant (déclarant) les objets issus de la plateforme. L'utilisateur accède à une seule méthode : la méthode parenthèses de la composition finale. Ainsi, il ne fait donc pas appel au code de la plateforme. Il n'est donc pas nécessaire de faire apparaître le nom des méthodes dans les schémas. L'objectif du schéma est de visualiser ce qui est nécessaire de mettre en œuvre pour la réalisation d'une heuristique. Par conséquent, nous ne mentionnons que les noms des composants sans préciser les méthodes qui les composent. Quand l'utilisateur développe un composant, il retrouve ainsi l'interface à laquelle il doit se reporter. À ce moment là, la documentation générée par doxygen complète alors la schématisation en exposant la sémantique des fonctions à implémenter.

Les composants **EO** sont souvent eux-mêmes des composés. Cette composition s'effectue lors de la construction de l'objet. Or, **EO** peut fournir plusieurs constructeurs pour un même objet. Par exemple, la classe eoEasyEA, qui engendre des algorithmes évolutionnaires, possède cinq constructeurs différents. Ces constructeurs sont autant de façons d'initialiser les variables membres de eoEasyEA. Chaque constructeur induit un nouveau schéma. Nous ne pouvons donc construire par cette méthode une vue unique de toute l'architecture de **EO** mais au contraire la vue qui correspond au travail effectué. Pour cela, nous suggérons à l'avenir de limiter la multiplicité des constructeurs. Si toutefois plusieurs constructeurs peuvent aider les utilisateurs, nous suggèrons de créer une classe mère possèdant le constructeur le plus général possible et des classes filles qui possèdent chacune un unique constructeur plus spécialisé.

5.2.1.3 Algorithme évolutionnaire hybride

En fait, la conception de cette métaheuristique sert à pallier un certain manque de souplesse de la classe eoEasyEA. Cette classe devrait être capable d'instancier tous les algorithmes évolutionnaires. Cependant, certaines hybridations étaient difficiles à réaliser. En fait, ceci était partiellement dû à un manque de réflexivité de la notion d'algorithme (la classe eoAlgo). Le deuxième point gênant de la classe évolutionnaire était l'impossibilité (sémantique) de faire un calcul incrémental du fitness. Cette impossibilité produit deux difficultés :

- Afin de réaliser un calcul optimisé du fitness, il était nécessaire de contourner la plateforme soit en calculant le fitness au moment de la modification locale, soit en laissant des informations nécessaires (généralement dans la configuration) pour le calcul par l'évaluateur.
- Le deuxième point empêche des possibilités d'hybridation. En effet, il était impossible de faire opérer des mécanismes génériques basés sur le fitness d'une configuration avant la phase d'évaluation.

Pour remédier à cela, nous proposons de réduire le plus possible le temps où le fitness d'une configuration possède un fitness invalidé. Nous avons donc ajouté un ensemble de classes à la plateforme **EO**. Les figure 5.7 et 5.8 présentent un schéma de classes global pour la partie HEA[7]. Un certain nombre de classes **EO** peut être récupéré pour la réalisation de cette métaheuristique. Les classes marquées d'un check rouge sont des stratégies dont toutes les classes filles étaient déjà existantes dans **EO**. Elles étaient déjà utilisables pour la classe eoEasyEA. Les autres classes sont donc nouvelles.

La partie la plus importante est de permettre un maximum d'hybridation, c'est-à-dire pouvoir intégrer un composant complexe là où il existait un composant simple. Pour cela, il est donc nécessaire que ces composants interchangeables (simples ou complexes) implémentent la même interface.

Nous avons donc pris comme base l'interface eoAlgo. Cette interface de type orange spécifie toutes les classes dont les instances modifient une population passée en paramètre de la méthode parenthèses. Les individus de la population doivent avoir leurs fitness valides à l'entrée de la procédure. La population est alors modifiée en garantissant la validité des fitness des configurations obtenues. En ceci eoAlgo se distingue de eoTransform qui est plus central dans la plateforme **EO** standard. En effet, eoTransform modifie certains individus en indiquant que le membre fitness est invalide pour qu'il soit calculé par la suite. Dans la partie que nous avons ajoutée, nous avons séparé d'une part le eoCycle (sélection, modification, remplacement) de l'itération d'une modification (eoEasyHEA). Par exemple, le code suivant décrit un algorithme génétique ultra-élitiste opérant sur des configurations du type T_conf à l'aide d'opérateurs spécifiques.

```
/* ...
  monXover (resp. maMut, monEval) est un croissement (resp.
  mutation, evaluateur) spécifique au problème implémentant
  l'interface eoQuadOp (resp. eoMonOp, eoEvalFunc)
*/
eoCompleteQuadOp<T_conf> monXoverComplet(monXover,monEval);
eoRate tauxXover(1.0);
eoQuadOp2ProbaAlgo<T_conf> lesXovers(monXoverComplet,tauxXovers);
eoCompleteMonOp<T_conf> maMutComplete(maMut,monEval);
eoRate tauxMut(0.1);
eoMonOp2ProbaAlgo<T_conf> lesMuts(maMutComplete,tauxMut);
eoRelayAlgo<T_conf> maModification(lesXovers);
maModification.add(lesMuts);
eoPlusReplacement<T_conf> monRemplace;
eoSelectBest<T_conf> maSelection;
eoCycle<T_conf> monCycle(maSelection, maModification, monRemplace);
```

[7] pour "Hybrid Evolutionary Algorithm".

```
eoGenContinue<T_conf> monContinuateur(1000);
eoEasyHEA<T_conf> monAG(monContinuateur,
                        monCycle);
/* ... */
```

Nous avons utilisé le même principe pour les algorithmes à configuration unique. Nous avons créé la classe eoCompleteMonOp, qui spécifie un opérateur changeant une configuration dont le fitness est valide en une autre configuration dont le fitness est valide. Cependant parmi les recherches locales, nous pouvons distinguer deux grands groupes :

- les algorithmes, comme le recuit simulé, qui génèrent une seule configuration voisine à partir de la configuration courante, puis une politique décide si la configuration courante devient la configuration générée.
- les algorithmes, comme la recherche taboue et le hill climbing, qui génèrent un ensemble de solutions et appliquent une politique sur cet ensemble pour sélectionner une configuration, puis décide si la configuration courante devient la configuration sélectionnée.

5.2.1.4 Conclusion

EO est une plateforme ouverte et extensible fournissant un large panel de composants. Son développement en C++ a permis d'obtenir une plateforme fortement réifiée de granularité fine, ce qui répond aux objectifs de modularité et de flexibilité.

Or actuellement, **EO** reste une plateforme complexe et difficile à appréhender pour le néophyte et peu inaccessible pour le non-informaticien, bien que des applications comme GUIDE commence à le démocratiser [CS03]. Nous avons proposé une schématisation permettant d'appréhender les compositions **EO** potentielles et effectives. Par ailleurs, nous avons proposé un ensemble de composants (HEA) permettant plus de réflexivité et la conception rapide de prototypes d'algorithmes hybrides.

5.2.2 Parallélisation de Cosearch pour la coloration

La synchronisation des différentes tâches est réalisée par trois types d'automates :

- Des automates contrôlant les agents distribués (voir figure 5.9).
- Des automates contrôlant une flotte d'agents d'un même type[8] (voir figure 5.10). Un automate de ce type sert d'interface entre la mémoire et l'ensemble des agents d'un même type.
- un automate contrôle la mémoire adaptative (voir figure 5.11).

La fin de l'algorithme est décidée de manière centralisée au niveau de la mémoire adaptative. La fin est choisie comme une politique mais survient après un nombre fixé d'itérations.

[8] sature, casse, ...

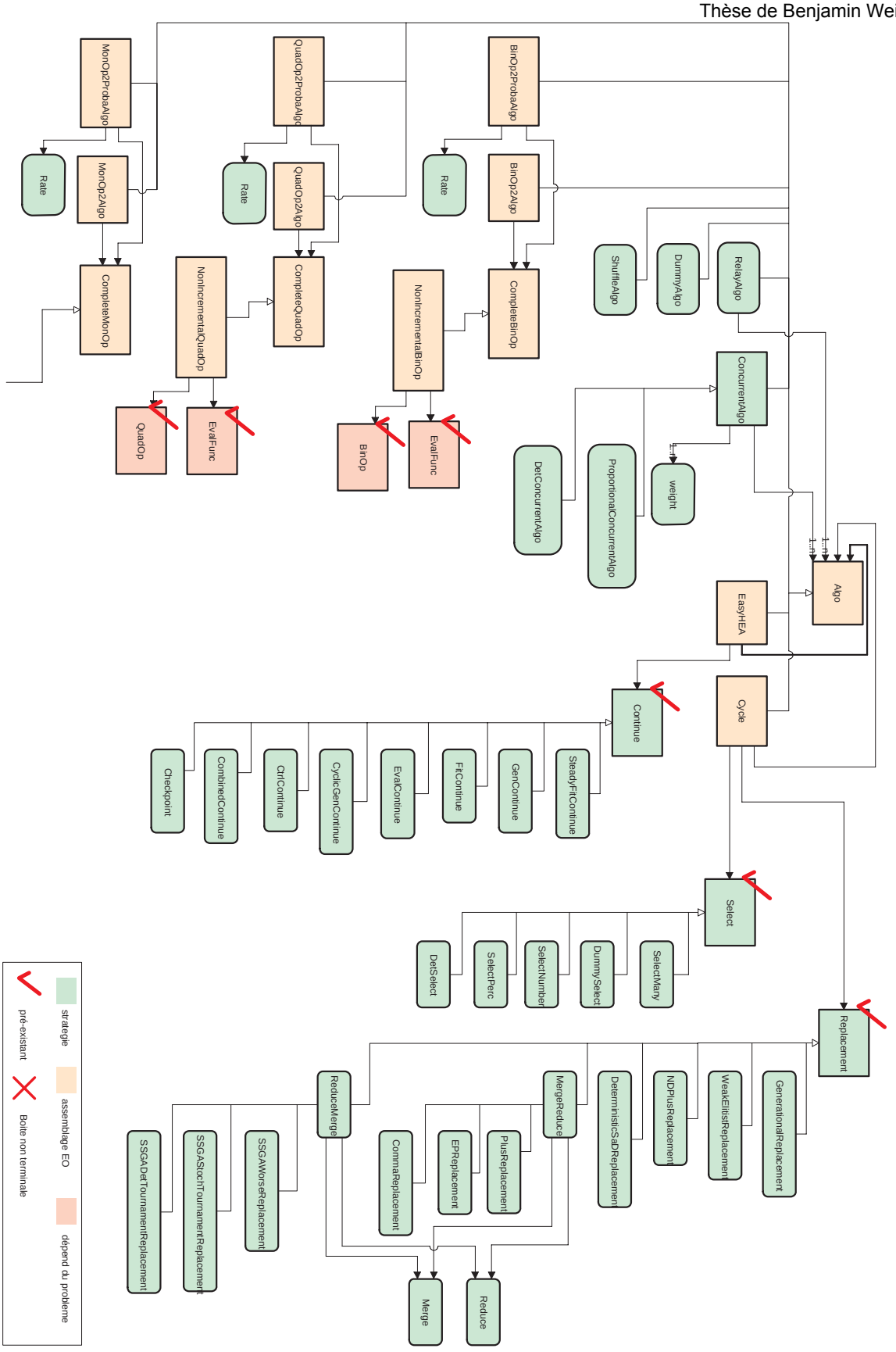


FIG. 5.7 – Schéma de classes pour les composants **EO** (partie I : les algorithmes à population).

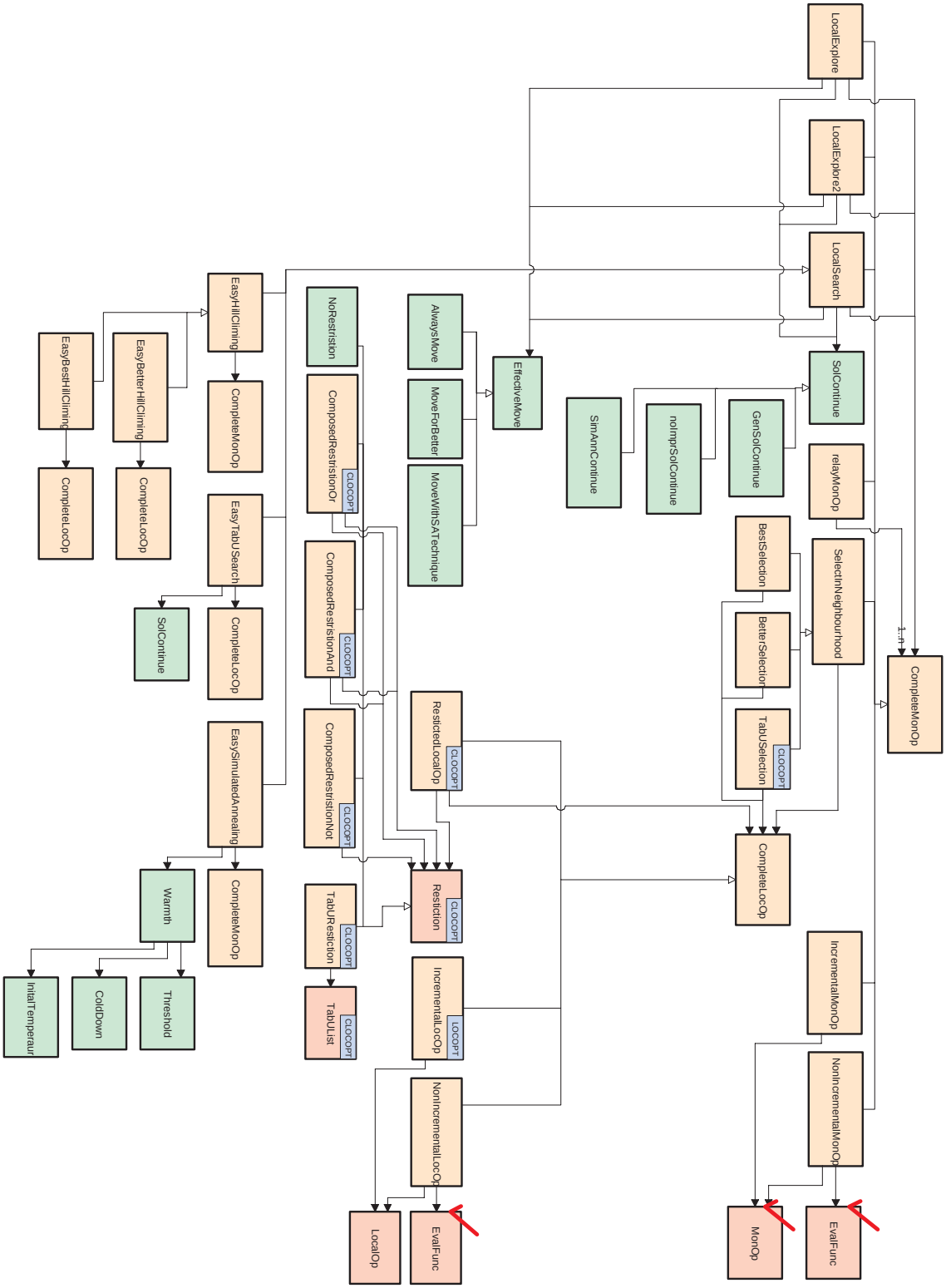


FIG. 5.8 – Schéma de classes pour les composants **EO** (partie II : les algorithmes à configuration unique).

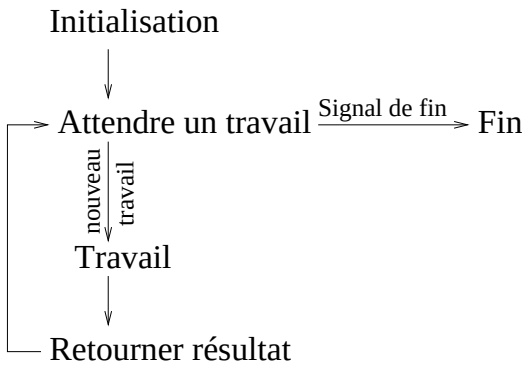
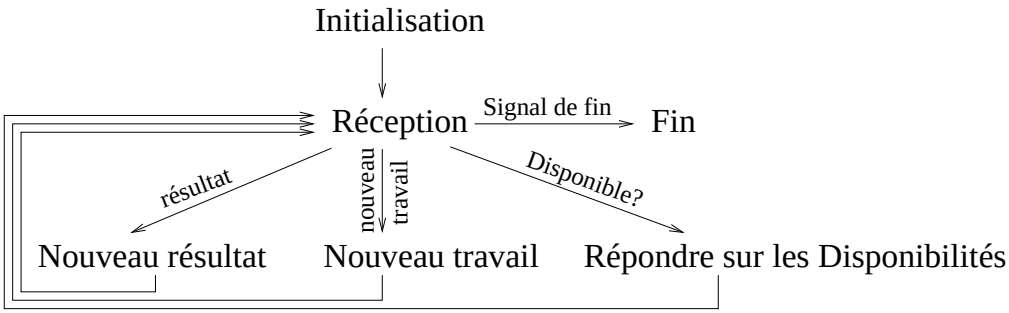
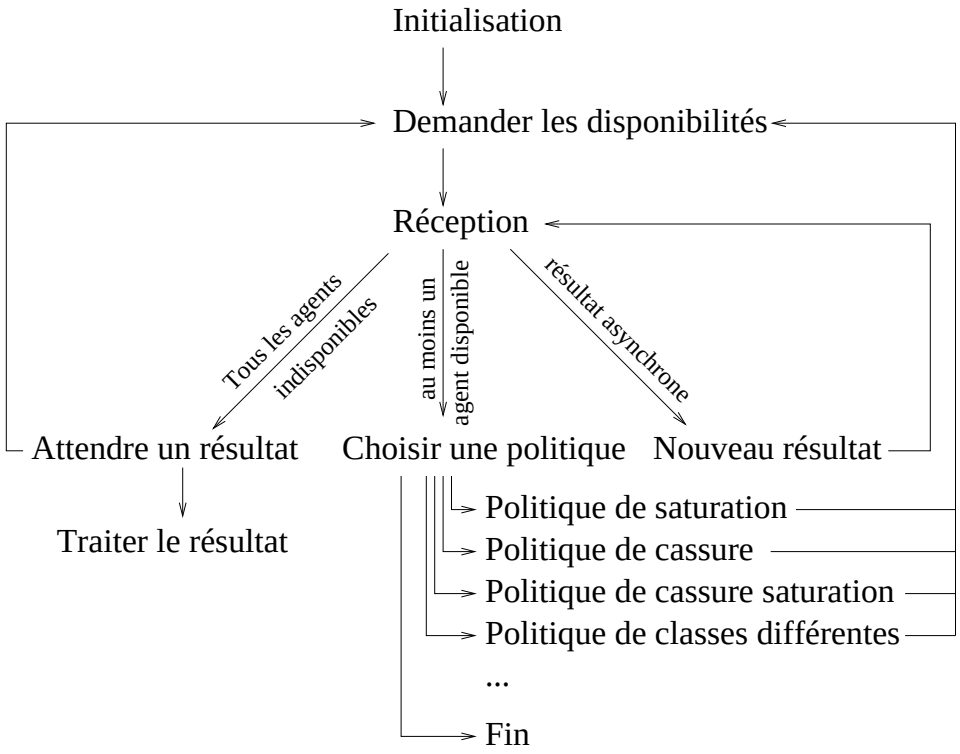


FIG. 5.9 – Automate de contrôle d'un agent.



Un manager de flotte d'agents agit comme une porte asynchrone. Les nouveaux travaux sont soumis aux travailleurs dès qu'au moins un agent est disponible. Dès qu'un résultat est obtenu, il est remonté jusqu'à la mémoire en passant par le manager.

FIG. 5.10 – Automate de contrôle d'un manager de flotte.



Dans l'automate de contrôle de la mémoire de trouve trois états principaux :

- *tous les agents travaillent. La mémoire attend donc l'obtention d'un résultat pour continuer.*
- *Un résultat est obtenu de manière asynchrone. L'automate l'intègre à la mémoire pour prendre une décision avec les informations les plus récentes.*
- *Au moins un agent est disponible. Un travail est fourni à un des agents disponibles.*

Les autres états servent à la synchonisation des différentes tâches.

FIG. 5.11 – Automate de contrôle de la mémoire adaptative.

Généralement, quand un agent termine son travail, il retourne son résultat. La mémoire lui fournit alors une nouvelle tâche à exécuter (vol de cycle). Or au début de l'algorithme et en de rares occasions pendant la résolution[9], des agents de différents types sont disponibles. L'automate de la mémoire choisit une politique par un tirage aléatoire uniforme entre les différents agents disponibles. Par exemple, si seuls des agents Casses et X_ρ sont sans travail alors la mémoire adaptative choisit entre ces deux politiques. Plus simplement, les politiques sature, casse sature, force identique, force différent et GPX ne pourront être choisies.

5.3 Expérimentations

Dans une première partie, nous présentons succinctement les benchmarks utilisés lors de nos évaluations. Puis nous évaluons la performance des agents d'intensification pris séparément. Pour finir, nous montrons comment nous avons instancié (assemblé) **Cosearch** avec ces composants.

5.3.1 Jeux de test utilisés

Afin d'évaluer nos algorithmes, nous avons repris les benchmarks évoqués dans la partie 4.3. Nous avons sélectionné les benchmarks présentant la plus forte amplitude, lors des expérimentations sur la classification des instances. Ainsi, nous avons des graphes fournissant peu de bonnes informations pour les algorithmes de recherche locale.

Le tableau 5.1 fournit des caractéristiques pour les graphes choisis (le nombre de sommets, la densité ...).

5.3.2 Impact des opérateurs d'intensification

Nous avons évalué l'efficacité de ces opérateurs grâce à un schéma **EO** identique pour tous les opérateurs (voir figure 5.12), dont les instanciations s'effectuent grâce aux spécifications de l'interface "eoMonOp".

Avec les opérateurs que nous avons développés, il est difficile de savoir si la solution obtenue ne peut être améliorée puisqu'il est facile de changer la configuration courante sans changer de fitness. En d'autres termes, nous ne pouvons savoir si nous sommes en train de parcourir un plateau ou si nous sommes dans un optimum local et qu'il faille augmenter le nombre de couleurs pour ensuite pouvoir rediminuer celui-ci.

[9] Grâce à la réception des résultat de manière asynchrone.

nom	nb sommets	densité	χ	meilleur	classe
school1_nsh	352	0.12	14	14	1
school1	385	0.13	14	14	1
DSJC500.9	500	0.90	-	129	2
latin_square_10	900	0.38	-	98	2
DSJC250.9	250	0.90	-	72	1
flat300_20_0	300	0.24	20	20	2
flat1000_76_0	1000	0.25	76	83	2
flat1000_60_0	1000	0.25	60	60	2
le450_5d	450	0.05	5	5	1
DSJC125.9	125	0.90	-	44	1
DSJC1000.5	1000	0.50	-	84	2
flat1000_50_0	1000	0.25	50	50	2
le450_5c	450	0.05	5	5	1
DSJC250.5	250	0.50	-	28	2
DSJR500.5	500	0.50	-	123	2
DSJC500.5	500	0.50	-	46	2
DSJR500.1c	500	0.10	-	85	0
DSJC125.5	125	0.50	-	17	1
flat300_28_0	300	0.24	28	31	2
flat300_26_0	300	0.24	26	26	2

Ce tableau fournit pour chaque benchmark utilisé le nom, le nombre de sommets du graphe, la densité, une indication sur le nombre chromatique (le - signifiant que cette valeur est inconnue), le meilleur résultat obtenu par un algorithme de coloration et la classe paysage constatée en 4.3.

TAB. 5.1 – Jeux de tests utilisés.

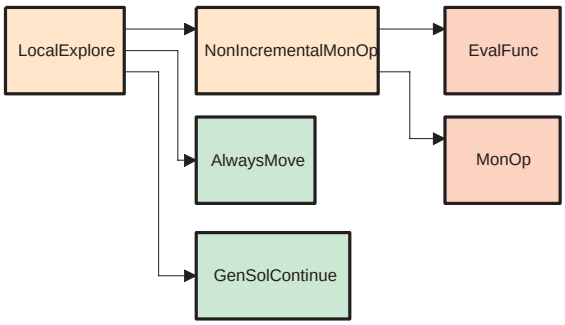


FIG. 5.12 – Schéma général pour tester un opérateur unaire.

Nous avons donc comparé les différents opérateurs en les exécutant n fois sur les différents graphes sélectionnés, où n est le nombre de sommets du graphe utilisé. Le tableau 5.2 reporte les résultats obtenus.

Nous constatons premièrement que ces opérateurs seuls retrouvent pour cinq graphes le "best known". Cependant, nous ne pouvons affirmer qu'un opérateur est vraiment meilleur qu'un autre, puisqu'ils sont meilleurs à tour de rôles sur les différents graphes.

Dans les expériences suivantes nous avons supprimé les graphes school1, school1_nsh, flat300_20_0 et le450_5c car les optima étaient atteints. Pour le graphe DSJR500.1c, nous ne pouvons savoir si l'optimum est atteint donc nous le gardons même si nous avons obtenu le "best known".

5.3.3 Discussion

Dans cette partie, nous avons présenté des mécanismes pouvant servir de briques élémentaires pour l'heuristique **Cosearch** : les opérateurs d'intensification par exemple peuvent être utilisés dans des algorithmes de type recherche locale.

Nous avons évalué les différents composants d'intensification, pour constater qu'ils avaient des performances différentes selon les graphes étudiés. L'implémentation sous **EO** des différents algorithmes nous a permis de reproduire facilement un schéma identique d'évaluation pour tous les mécanismes d'intensification.

Nous avons également proposé quatre opérateurs de diversification.

Il est difficile d'estimer l'apport de composants de diversification car ils introduisent des caractères inexplorés indépendamment du fitness des solutions visitées et *a fortiori* des nouvelles solutions. De plus pour les utiliser, il faut des configurations à diversifier. L'évaluation de la diversification est par conséquent trop dépendante de l'interaction avec l'intensification.

5.3.4 Évaluation de l'algorithme parallèle

Nos algorithmes ont été implémentés grâce à l'environnement C-PVM [GBD+94]. L'exécution a été faite sur un réseau hétérogène de PC Linux (processeurs cadencés entre 1.9 et 3 Ghz et mémoires variant entre 256 Mo et 1Go de RAM).

La charge variable et l'incertitude de la disponibilité des machines ne nous permet pas d'effectuer des mesures de temps d'exécution de nos algorithmes.

Nous avons équilibré les différents agents : les agents unaires (d'intensification et de diversification) appliquent 100 fois leur opérateur ; les agents quaternaires appliquent un crossover puis une série de 50 applications de l'opérateur de casse sur les deux colorations résultantes.

	SATURE				CASSE				CASSE_SATURE			
	min	max	E	E. T.	min	max	E	E.T.	min	max	E	E.T.
school1_nsh	**14**	19	15,9	1,663	**14**	**14**	**14**	**0**	**14**	15	14,2	0,42
school1	**14**	16	14,7	0,67	**14**	15	14,1	0,32	**14**	**14**	**14**	**0**
DSJC500.9	140	146	142,9	1,66	138	141	139,1	0,88	137	142	139,4	1,43
latin_square_10	114	118	116	1,25	108	111	109,5	0,85	109	111	110	0,67
DSJC250.9	78	81	79,3	0,95	77	78	77,3	0,48	76	79	77,6	0,84
flat300_20_0	36	39	37,8	1,03	**20**	39	31,9	8,85	**20**	39	34,4	5,54
flat1000_76_0	108	110	109,4	0,7	110	112	111,3	0,67	111	112	111,4	0,52
flat1000_60_0	109	110	109,7	0,48	109	112	110,8	1,03	110	112	111	0,82
le450_5d	6	7	6,7	0,48	6	7	6,6	0,52	6	7	6,7	0,48
DSJC125.9	45	47	46	0,94	45	45	45	0	46	46	46	0
DSJC1000.5	110	112	111	0,82	111	114	112,5	0,85	111	113	112,2	0,63
flat1000_50_0	108	110	109,1	0,88	109	111	110	0,47	110	112	110,7	0,67
le450_5c	6	7	6,7	0,48	6	7	6,8	0,42	**5**	7	6,6	0,84
DSJC250.5	35	37	35,8	0,63	35	36	35,6	0,52	35	36	35,2	0,42
DSJR500.5	132	134	132,7	0,67	131	133	132,3	0,67	132	133	132,3	0,48
DSJC500.5	61	64	62,3	0,95	61	64	62,7	1,06	61	64	62,4	0,84
DSJR500.1c	**85**	86	85,1	0,32	85	86	85,1	0,32	**85**	**85**	**85**	**0**
DSJC125.5	21	21	21	0	21	21	21	0	21	21	21	0
flat300_28_0	39	41	39,8	0,63	39	41	40	0,47	39	40	39,6	0,52
flat300_26_0	39	40	39,8	0,42	39	41	40	0,47	39	41	40	0,47

Ce tableau fournit pour les trois opérateurs unaires (Sature, Casse et Casse_Sature) le minimum, le maximum, la moyenne et l'écart type sur le nombre de couleurs trouvé.

TAB. 5.2 – Tableau récapitulatif des performances des opérateurs unaires.

Pour toutes les expériences, nous disposons du même nombre total d'agents, soit 12 agents de chaque type pour l'architecture complète (les sept types d'agents) de **Cosearch**, 21 agents de chaque type pour l'architecture n'en comportant que quatre type (les trois agents intensifiant et une technique de diversifications), et 28 pour une architecture avec uniquement les 3 agents d'intensification. L'algorithme est itéré pendant 1000 choix de politique.

Les tableaux 5.3 et 5.4 regroupent les résultats pour l'ensemble des combinaisons de l'architecture de **Cosearch**.

5.3.5 Discussion

La première remarque porte sur les graphes utilisés. Nous constatons que les graphes les moins discriminants, à savoir le450_5d, DSJC125.9, DSJR500.1c et DSJC125.5, appartiennent aux classes 0 et 1 identifié dans la partie 4.3. De plus, on constate que les résultats sont proches des "best known" (seul les résultats de DSJC125.5 diffèrent avec une couleur de plus).

Nous constatons également que pour tous les schémas de coopérations les nombres couleurs obtenus sont meilleurs qu'en utilisant les opérateurs d'intensification seules. De la même façon la combinaison des trois opérateurs d'intensification est insuffisante pour obtenir les meilleurs résultats.

Nous remarquons que l'algorithme mettant en œuvre tous les composants ne présente pas les meilleurs résultats. Au mieux il atteint la même qualité de résultat qu'un autre schéma et dans de nombreux cas l'absense totale de diversification est plus intéressante que l'utilisation de toutes les diversifications. Ceci peux s'explique par deux raisons :

- l'architecture complète étant lourd donc elle nécessite plus de temps pour converger.
- il est possible que la "sur-diversification" de la recherche perde l'algorithme. En fait en diversifiant trop, **Cosearch** parcourerait l'espace de recherche sans détecter les régions intéressantes.

Aucun opérateur de diversification n'est meilleur sur tous les graphes sélectionnés. Cependant, on constate que les opérateurs FM et X_ρ sont plus efficaces que les autres. En effet, FM obtient les meilleurs résultats pour 12 graphes et X_ρ obtient les meilleurs résultats pour 9 graphes. De plus à eux deux, ils couvrent l'ensemble des meilleurs résultats obtenus pour tous les graphes.

Afin de valider les résultats, nous proposons d'instancier **Cosearch** utilisant les agents X_ρ et FM en plus des trois agents d'intensification (voir tableau 5.5). Les résultats obtenus par la combinaison des deux opérateurs de diversification sont globalement meilleurs que ceux obtenues séparément. Bien que nous n'obtenons pas non plus la combinaison des meilleurs résultats précédents, la combinaison n'est

	S+C+CS+FD+FM+GPX+X_ρ				S+C+CS+FD			S+C+CS+FM			
	min	max	E	E. T.	min	max	E	E.T.	min	max	E	E.T.
DSJC500.9	131	135	133,4	1,67	132	133	132,8	0,45	131	134	132,6	1,14
latin_square_10	104	107	105,2	1,30	104	106	105	1	104	105	104,6	0,55
DSJC250.9	73	74	73,8	0,45	73	74	73,8	0,45	74	75	74,2	0,45
flat1000_76_0	107	109	108,2	0,84	105	107	105,8	0,84	105	106	105,2	0,45
flat1000_60_0	106	109	107,6	1,34	105	106	105,8	0,45	104	106	105	0,71
le450_5D	5	5	5	0	5	5	5	0	5	5	5	0
DSJC125.9	44	44	44	0	44	44	44	0	44	44	44	0
DSJC1000.5	108	110	109	0,71	107	108	107,6	0,55	105	108	106,6	1,14
flat1000_50_0	106	107	106,6	0,55	102	105	104	1,22	98	104	101,4	2,30
DSJC250.5	33	33	33	0	32	32	32	0	31	32	31,8	0,45
DSJR500.5	130	131	130,2	0,45	128	129	128,6	0,55	129	129	129	0
DSJC500.5	59	60	59,6	0,55	58	60	58,6	0,89	58	60	58,8	0,84
DSJR500.1c	85	85	85	0	85	85	85	0	85	85	85	0
DSJC125.5	18	19	18,4	0,55	18	18	18	0	18	19	18,2	0,45
flat300_28_0	36	38	37,4	0,89	36	37	36,4	0,55	35	36	35,8	0,45
flat300_26_0	37	38	37,6	0,55	36	37	36,4	0,55	36	37	36,6	0,55

La méthode S+C+CS+FD+FM+GPX+X_ρ correspond à l'architecture complète avec les sept types d'agents (satures, casses, casse satures, force classes différentes, force même classes, GPX et X_ρ). La méthode S+C+CS+FD correspond à l'architecture utilisant des agents de types satures, casses, casse satures et force classes différentes. La méthode S+C+CS+FM correspond à l'architecture utilisant des agents de types satures, casses, casse satures et force même classes.

TAB. 5.3 – Tableau récapitulatif des performances de différentes combinaisons possibles de **Cosearch**(1).

	S+C+CS+GPX				S+C+CS+X_ρ				S+C+CS			
	min	max	E	E. T.	min	max	E	E.T.	min	max	E	E.T.
DSJC500.9	132	135	133,6	1,14	133	134	133,6	0,55	132	133	132,2	0,45
latin_square_10	104	105	104,8	0,44	104	106	104,8	0,84	105	105	105	0
DSJC250.9	73	75	74,2	0,84	73	75	74	0,71	73	75	73,8	1,1
flat1000_76_0	105	107	106	0,71	106	107	106,2	0,45	105	106	105,4	0,55
flat1000_60_0	104	106	105,4	0,89	105	106	105,4	0,55	105	105	105	0
le450_5D	5	5	5	0	5	5	5	0	5	5	5	0
DSJC125.9	44	44	44	0	44	44	44	0	44	44	44	0
DSJC1000.5	107	108	107,2	0,45	106	108	107	1	106	107	106,6	0,55
flat1000_50_0	101	105	102,8	1,48	104	105	104,4	0,55	102	104	102,8	0,84
DSJC250.5	32	33	32,2	0,45	32	33	32,2	0,45	32	32	32	0
DSJR500.5	129	129	129	0	128	130	129	0,71	128	129	128,6	0,55
DSJC500.5	58	59	58,8	0,45	57	59	58,2	0,84	58	59	58,2	0,45
DSJR500.1c	85	85	85	0	85	85	85	0	85	85	85	0
DSJC125.5	18	18	18	0	18	19	18,2	0,45	18	19	18,2	0,45
flat300_28_0	36	37	36,6	0,55	36	37	36,2	0,45	36	36	36	0
flat300_26_0	36	36	36	0	35	37	36,2	0,84	35	37	36	0,71

La méthode S+C+CS+GPX correspond à l'architecture utilisant des agents de types satures, casses, casse satures et GPX. La méthode S+C+CS+X_ρ correspond à l'architecture utilisant des agents de types satures, casses, casse satures et X_ρ. La méthode S+C+CS correspond à l'architecture utilisant uniquement les trois types d'agents d'intensification (satures, casses, casse satures).

TAB. 5.4 – Tableau récapitulatif des performances de différentes combinaisons possibles de **Cosearch**(2).

	S+C+CS+FM+X_ρ			
	min	max	E	E.T.
DSJC500.9	131	134	132,4	1,34
latin_square_10	104	105	104,8	0,45
DSJC250.9	74	74	74	0
flat1000_76_0	104	106	105	0,71
flat1000_60_0	**103**	106	105	1,22
le450_5d	5	5	5	0
DSJC125.9	44	44	44	0
DSJC1000.5	105	107	106	0,71
flat1000_50_0	102	105	103,4	1,52
DSJC250.5	32	32	32	0
DSJR500.5	128	129	128,8	0,45
DSJC500.5	57	58	57,6	0,55
DSJR500.1c	85	85	85	0
DSJC125.5	18	19	18,2	0,45
flat300_28_0	36	36	36	0
flat300_26_0	36	37	36,6	0,55

TAB. 5.5 – Tableau récapitulatif des performances de **Cosearch** avec les diversifications FM et X_ρ (3).

jamais moins bonne que le pire de ces composants. Pour finir, nous constatons que pour un graphe (flat1000_60_0) la combinaison de FM et X_ρ fournit un meilleur résultat.

5.4 Conclusion

Dans ce chapitre, nous avons présenté une métaheuristique coévolutionnaire appelée **Cosearch**. Pour implémenter cette stratégie générique, il est nécessaire de décrire de nombreux mécanismes du fait de son haut niveau d'abstraction. Cependant, nous avons précisé les caractères intrinsèques de **Cosearch** :

- utilisation d'une population de configurations pour mémoriser les régions intéressantes de l'espace de recherche ;
- détection des caractères (ou régions) insuffisamment explorés ;
- équilibrage *explicite* de l'intensification et de la diversification ;
- exploitation des ressources parallèles.

Afin de mieux communiquer sur la plateforme nous proposons une schématisation simple faisant apparaître à la fois la hiérarchie de classes et la composition des composants **EO**. Nous avons étendu la plateforme logicielle **EO** pour une réflexivité plus importante. Cette réflexivité était indispensable pour la conception

d'heuristiques hybrides.

Pour finir, nous avons proposé une instanciation de **Cosearch** pour le problème de coloration de graphe. Nous avons instancié plusieurs agents d'intensification et de diversification. Dans ce cadre, nous avons observer que la diversification était un moyen efficace pour améliorer les résultats. Par ailleurs, nous avons également monté que la sur-diversification était préjudiciable à la convergeance de l'algorithme. Nous avons constaté que deux opérateurs de diversification se dénotent particulièrement parmi les opérateurs proposés :

- X_ρ : un opérateur de croisement déduit de nos travaux sur la symétrie.
- FM : un opérateur unaire forçant la recherche de coloration en imposant deux sommets à appartenir à la même classe.

L'amélioration globale des résultats d'utilisation de ces deux agents de diversification dans **Cosearch** et la présence d'un graphe pour lequel la combinaison est meilleure que les éléments séparés est encourageant. Cependant, nous n'avons pas produit la panacée. En effet, nous constatons une détérioration de l'union des résultats sur d'autres graphes.

Pour finir, nous n'avons pas pu améliorer les meilleurs résultats obtenus dans la communauté. Ceci peut s'expliquer par le choix *a priori* que nous avons fait de ne chercher des solutions uniquement parmi les colorations valides et en s'interdisant les violations de contraintes contrairement aux autres travaux.

Perspective

Dans ce mémoire, nous avons abordé de nombreux aspects de l'optimisation combinatoire et nous avons obtenu des résultats théoriques intéressants. Cependant, il nous semble prometteur d'explorer ces nouvelles pistes pour prolonger nos travaux.

La notion de PLS-complétude définit un problème d'optimisation comme un composé d'un espace de recherche, d'une fonction objectif et d'un opérateur de voisinage. Il existe une notion de PLS-complétude comparable à la notion de NP-complétude. La corrélation entre la preuve de structure dans le problème de k-coloration de graphe et sa notion classique de voisinage nous fait croire que cette notion peut servir à étendre notre résultat sur la notion de structure de problème à d'autres problèmes. Il sera alors question de vérifier si un mécanisme de réduction PLS peut permettre de prouver que tous les problèmes PLS-complets sont structurés.

La plateforme **EO** nous semble une voix à prendre pour la concéption d'heuristisques et de *nouvelles* heuristiques. Nous pensons qu'il est intéressant de rassembler toutes les critiques émises par les utilisateurs, afin de générer une nouvelle version de la plateforme.

- Évolutionnaires et locales. Les algorithmes évolutionnaires sont avant tout des recherche locales travaillant sur des configurations complexes (les parties de l'espace de recherche).
- Hybrides (ou coopératif). Augmenter la reflexivité de la plateforme permettra d'avoir plus de souplesse pour développer de nouvelles méthodes hybrides.
- Parallèle. La plateforme devra exploiter la puissance de calcul disponible de manière transparante et portable.

Parmi les problèmes d'optimisation, certains sont considérés comme des problèmes déceptifs. Cependant, ces problèmes présentent des propriétés fortes comme, par exemple, l'existence d'un attracteur déceptif et sa valeur par rapport à l'optimum global. Nous croyons que ces propriétés peuvent justement être utilisées pour améliorer les approches heuristiques. D'autre part, certaines instances (de problèmes connus) ont été prouvées comme étant des problèmes déceptifs [Pap94]. Nous pensons qu'il serait donc intéressant de reprendre une activité de recherche dans ce domaine en traitant des problèmes du type SAT en cherchant à identifier des sous-problèmes déceptifs d'un problème donné.

Chapitre 6

Conclusion

Dans cette thèse, nous avons traité plusieurs aspects de l'optimisation combinatoire. La difficulté des problèmes à résoudre incite l'utilisation d'heuristiques. En particulier, les métaheuristiques sont devenues d'usages fréquents. Or, la variation d'efficacité de ces techniques combinée avec la parution du théorème du No Free Lunch (**NFL**) nécessitait, à notre sens, une clarification. En effet, nous avions d'une part les métaheurisiques qui ont fait leurs peuvres et qui présentent l'avantage d'être réutilisable ; d'autre part le théorème **NFL** qui affirme que toutes les heuristiques se valent appuyé, par la constatation que toutes les métaheuristiques sont plus ou moins efficaces selon le problème étudié.

Dans un premier temps, nous nous sommes donc positionné par rapport au théorème du **NFL**, justifiant ainsi l'utilisation de méthodes génériques pour traiter les problèmes d'optimisation : métaheuristiques et outils statistiques pour l'analyse de paysage. Pour cela, nous avons défini une notion de structure qui empêche l'application du théorème. Puis, nous avons prouvé que le problème de k-coloration de graphe vérifie cette notion de structure. Malheureusement, notre tentative pour étendre le résultat sur les problèmes de k-coloration à d'autres problèmes par la réduction polynomiale, s'est soldé par un échec.

Cet echec nous incite à croire qu'il existe également une partie spécifique à tous les problèmes. Nous pensons qu'il est alors judicieux de l'identifier pour pouvoir utiliser au mieux les mécanismes génériques de résolution mais aussi les méthodes d'analyse. C'est ce que nous avons dû faire pour le problème de coloration de graphe que nous avons utilisé pour illustrer nos travaux.

Pour le problème de coloration de graphe, comme pour tous les problèmes de partitionnements, il existe une forte symétrie. La présence de cette symétrie était un frein à de nombreuses techniques dont des opérateurs de recombinaisons et un

calcul de distance. Notre analyse nous a permis de supprimer les effets de cette symétrie. Nous avons ainsi pu développer un test d'égalité entre deux colorations en temps de calcul linéaire, et deux notions de distance entre deux colorations. Bien que les deux distances soit calculables polynômialement la seconde "d_ρ" est plus économique en temps de calcul et présente un biais intéressant pour le problème de coloration.

Grâce aux outils que nous avons développés, il a été possible d'analyser les problèmes de colorations par des méthodes statistiques. Nous avons pu classifier les problèmes du 2^{me} *Challenge DIMACS* en trois catégories (de la plus simple à la plus difficile à optimiser) :

- massif central ;
- multi-massif ;
- uniforme.

Le développement de nos algorithmes a été effectué sur la plateforme **EO**. Nous avons proposé un système de schématisation pour communiquer les compositions de **EO**. Cette schématisation inspiré d'UML permet, grâce à un code de couleur, de visualiser facilement les différentes tâches nécessaires à l'instanciation d'une métaheuristique. À l'heure d'aujourd'hui **EO** présente de nombreux points positifs comme sa modularité due à la forte réification de la plateforme. Nous avons proposé dans ce cadre l'ajout d'un ensemble de composants (HEA) permettant la conception d'heuristiques hybriques plus facilement.

Nous avons instancié plusieurs agents d'intensification et de diversification pour le problème de coloration de graphe. Dans ce cadre, nous avons observer que la diversification était un moyen efficace pour améliorer les résultats. Par ailleurs, nous avons également montré que la sur-diversification était préjudiciable à la convergeance de l'algorithme. Nous avons remarqué deux opérateurs de diversification particulièrement prometteurs :

- X_ρ : un opérateur de croisement déduit de nos travaux sur la symétrie.
- FM : un opérateur unaire forçant la recherche de coloration en imposant deux sommets à appartenir à la même classe.

Malheureusement, nous n'avons pas pu améliorer les meilleurs résultats obtenus dans la communauté. Ceci peut s'expliquer par le choix volontaire de prendre le contre pied de la plupart des études en explorant uniquement les colorations valides pour trouver une solution.

En résumé, nous avons proposé une légitimation théorique à l'usage des approches génériques (par métaheuristiques ou analyse du paysage) pour résoudre les problèmes difficiles. Sur le problème de coloration de graphe, nous avons proposé des outils formels et appliqués permettant d'utiliser de telle méthodes génériques. Pour appuyer notre thèse, nous avons évalué la pertinence de nos outils, aboutissant à une classification des problèmes de colorations de graphe et à la validation d'opérateurs à l'aide d'un algorithme coopératif parallèle.

Chapitre 7

Annexe B : Glossaire

À propos du NFL

- $\mathcal{X}$: espace de recherche.
- $\mathcal{Y}$: ensemble des coûts.
- $\prec$: relation d'ordre sur $\mathcal{Y}$. On dit que u est meilleure que v si $u \prec v$.
- $\mathcal{Y}^{\mathcal{X}}$: ensemble des problèmes d'optimisation définit par $\mathcal{X}$ et $\mathcal{Y}$.
- Un algorithme A est caractérisé par la séquence des configurations visitées $(a_0, a_1, a_2, \dots a_m, \dots)$.
- comparaison d'algorithmes A, B à l'itération m : comparaison du meilleur élément d'indice inférieur ou égal à m (i.e. A est meilleur que B s'il existe $i \leq m$ tel que pour tout $j \leq m$, $f(a_i) \prec f(b_j)$).

Espace de recherche

- V : ensemble des sommets du graphe considéré.
- E : ensemble des arcs du graphe considéré.
- $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$: des colorations i.e. des applications de V dans $\{1 \dots |V|\}$.

- $\mathcal{R}$: ensemble des colorations (i.e. ensembles des représentants).
- $\sim$: la relation d'équivalence entre les colorations.

- $\mathcal{A}_\sim$, $\mathcal{B}_\sim$, $\mathcal{C}_\sim$: des partitionnements de V (i.e. des classe d'équivalence de coloration).

- $\mathcal{O}$: ensemble des partitionnements (i.e. $\mathcal{O} = \mathcal{R}_{/\sim}$).

Métriques de l'espace de recherche

- $\mathcal{M}^{\mathcal{AB}}$: matrice de raffinement entre les colorations $\mathcal{A}$ et $\mathcal{B}$ (i.e. $m_{ij}^{\mathcal{AB}} = |\mathcal{A}^{-1}(i) \bigcap \mathcal{B}^{-1}(j)|$).

- $s(\mathcal{A}, \mathcal{B}, \tau)$: indicateur de ressemblance entre $\mathcal{A}$ et $\mathcal{B}$ relativement à l'application τ

- $e(\mathcal{A}, \mathcal{B}, \tau)$: indicateur de dissemblance entre $\mathcal{A}$ et $\mathcal{B}$ relativement à l'application τ

- parties communes $\mathcal{C}$ de deux colorations $\mathcal{A}$ et $\mathcal{B}$ relativement à l'application τ : $\mathcal{C}^{-1}(i) = \bigcup_{j \in \tau^{-1}(i)} (\mathcal{A}^{-1}(j) \bigcap \mathcal{B}^{-1}(i))$.
- $\sigma^{\mathcal{AB}}$: bijection de ressemblance maximale pour $\mathcal{A}$ et $\mathcal{B}$.

- $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim)$: distance déduite de $\sigma^{\mathcal{AB}}$ (i.e. $d_\sigma(\mathcal{A}_\sim, \mathcal{B}_\sim) = e(\mathcal{A}, \mathcal{B}, \sigma^{\mathcal{AB}})$).

- $\rho^{\mathcal{AB}}$: application de ressemblance maximale pour $\mathcal{A}$ et $\mathcal{B}$.

- $d_\rho(\mathcal{A}_\sim, \mathcal{B}_\sim)$: distance déduite de $\rho^{\mathcal{AB}}$ (i.e. $d_\rho(\mathcal{A}_\sim, \mathcal{B}_\sim) = max(e(\mathcal{A}, \mathcal{B}, \rho^{\mathcal{AB}}), e(\mathcal{B}, \mathcal{A}, \rho^{\mathcal{BA}}))$.

Terminologie objet, application à EO

- **Instancier** : créer une instance, un exemple. En programmation orienté objet, on utilise se terme pour parler de la création d'un objet. L'instanciation nous fait passer du schéma de classes (schéma des possibles) au schéma d'instance (schéma effectif ou schéma d'execution). C'est à ce moment qu'on constate le partage des objets. Notons que dans **EO**, l'instanciation d'une classe spécifie l'assemblage de l'heuristique par les paramètres des constructeurs mais précis

également le problème traité par la spécification des paramétres “templates” de la classe.

- **Réflexivité** : se dit d'un système qui s'applique à lui même. Nous utilisons ce terme pour signifier que les composants **EO** ont une vue sur des objets instanciant le même interface (ou un interface plus générique encore). Par exemple, un eoHEA est un eoAlgo particulier mais posséde également une vue sur un (autre) eoAlgo, par exemple un eoCycle. Ainsi la notion d'algorithme évolutionnaire (eoAlgo) est réflexive car elle peut avoir une vue sur elle même.

- **Réification** : matérialiser un concepte par un objet. Nous utilisons ce terme pour signifier que la quasi-totalité de la plateforme **EO** et constituée d'objet. En particulier, nous n'utilisons pas de fonctions ou des procédures mais nous déclarons des objets *agissant comme* des fonctions. Il est alors possible de déplacer facilement une fonctionnalité puisqu'il s'agit d'un objet.

- **Flexibilité** : Décrit la souplesse d'utilisation d'un outil. Nous utilisons ce terme pour préciser la modularité de **EO**. Pour **EO**, la flexibilité provient de sa réification, sa réflexivité mais aussi de l'évolutivité induite par le fait qu'elle soit open source.

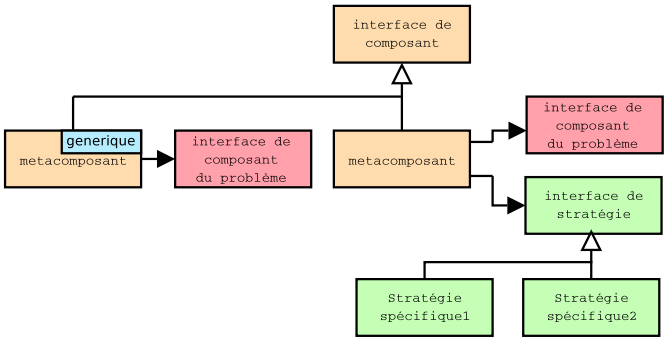
Schématisation EO

- **boîtes vertes** : interfaces et composants génériques (utilisable indépendamment du problème).

- **boîtes oranges** : interfaces et composants d'assemblage (utilisateur doit choisir un assemblage pour une résolution efficace).

- **boîtes rouges** : interfaces des composants spécifiques aux différents problèmes (configurations, evaluateurs, opérateurs).

- **boîtes bleues** : paramètres de “templétisation” différents de EOT (utilisé pour certains composants des recherches locales).



Cette figure représente un schéma possible pour des composants **EO**. *Une métaheuristique ou un métacomposant est spécifié par son interface. Il est possible de l'instancier de deux méthodes. La première en utilisant un composant "templatisé" par générique et utilisant un composant propre à l'application. La seconde en instanciant un composant nécessitant, lui même, deux composants. le premier dépend du problème, le second est une stratégie générique dont deux formes spécialisées existent déjà dans les bibliothèques* **EO**.

FIG. 7.1 – Rappel de schémas **EO**.

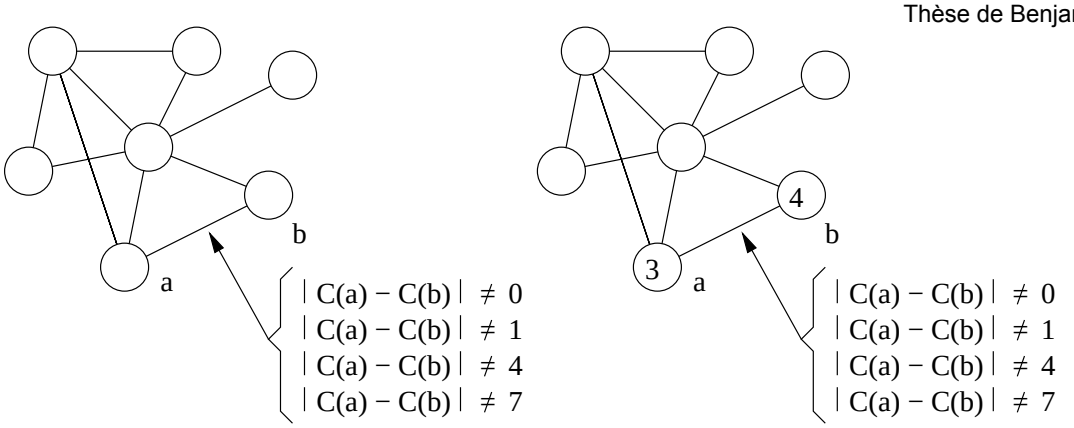
Chapitre 8

Annexe A : Résolution du problème d'affectation de fréquences avec polarités

Le problème d'affectation de fréquences avec polarités a été proposé dans le cadre du challenge ROADéF 2001. Pour résoudre cette classe de problèmes, nous proposons l'algorithme Split Iterated Greedy (SIG), dont le principal avantage est d'être économique en temps de calcul. Nous avons également défini des variables complexes - appelées connexions - réduisant l'espace de recherche, sur lesquelles nous avons fait opérer SIG. Pour un meilleur guidage dans l'espace de recherche, nous proposons l'utilisation de fonctions heuristiques pour distinguer des configurations de même coût. Plusieurs critères pour guider l'algorithme sont présentés et comparés à la fois du point de vue de la meilleure solution trouvée mais aussi au niveau de leur robustesse. SIG pouvant toujours, malgré ces mécanismes, être bloqué dans un optimum local, nous lui ajoutons des mécanismes inspirés de la recherche taboue. Ces derniers ont montré leurs efficacités en terme de qualité des solutions fournies.
Ces Travaux ont été effectués pour **le challenge ROADéF 2001** et ont été présenté à **FRANCORO III**, Quebec, mai 2001.

8.1 Introduction

Les réseaux hertziens se sont largement développés pour assurer la communication entre des sites distants. Ces réseaux permettent la connexion de sites sans fils. Cependant la conception de tels réseaux nécessite de respecter de nombreuses



Sur ces exemples, nous montrons les contraintes liées aux somments a et b et à l'arc (a,b). Des contraintes similaires existent sur les sommets et les arcs.

À gauche, un exemple de problème de T-coloration. Sur cet exemple l'arc (a,b) impose quatre contraintes sur les valeurs affectées aux sommets a et b.

À droite, un exemple de problème de TE-coloration. Ici, il est question d'affecter un ensemble de trois valeurs différentes au sommet a.

FIG. 8.1 – Illustration des problèmes de T-coloration et de TE-coloration de graphe.

contraintes pour garantir la qualité des messages transmis. Un réseau hertzien est constitué d'émetteurs et de récepteurs capables de communiquer par onde radio. Lors de la communication, les messages échangés sont transmis sur une fréquence précise.

L'affectation des ressources électromagnétiques telles que les fréquences et les polarités des signaux aux différents transmetteurs est une étape importante du *design*. En effet, une mauvaise affectation entraîne une diminution de la qualité des communications : interférences sur les messages échangés voire même une perte de certains signaux. Le modèle utilisé pour le problème d'affectation de fréquences a évolué au cours des dernières années. À l'origine, il était modélisé comme un problème de coloration de graphe [Gam86], où les transmetteurs proches devaient utiliser des fréquences différentes. Puis, progressivement, plusieurs spécificités furent remarquées. Le problème d'affectation de fréquences fut ensuite présenté comme un cas particulier des problèmes de T-coloration de graphe et de TE-coloration de graphe. Ces deux problèmes proches sont en fait des généralisations du problème de coloration de graphe [Akr94]. Le problème de T-coloration spécifie plusieurs contraintes sur un même arc (figure 8.1 partie gauche). Le problème de TE-coloration ajoute des besoins de plusieurs fréquences sur un sommet du graphe (figure 8.1 partie droite). Ce type de modélisation est communément utilisé pour les réseaux GSM (Global System for Mobile communication) [Dor98].

Dans les années 1993-1995, le CELAR[1] fournit une formalisation complète pour le projet CALMA[2]. Ce modèle prenait déjà en compte différents types de contraintes telles que les contraintes de domaine, les contraintes de séparation de fréquences et les contraintes d'égalité de fréquences. La définition et les fichiers de description sont accessibles depuis la page de T. Schiex[3]. Plusieurs métaheuristiques comme la recherche Taboue [BBD+95], les colonies de fourmis [MC99] et les algorithmes génétiques [VT98] furent utilisées pour le résoudre. Pour intégrer la polarité des signaux, une extension est proposée dans le challenge ROADéF[4] : le problème d'affectation de fréquences avec polarités (FAPP). L'information de polarité permet de moduler certaines contraintes entre les fréquences. En effet, des messages émis sur des polarités différentes ont moins tendance à s'interférer, ce qui autorise l'utilisation de fréquences plus proches. Par ailleurs, le problème présente deux types de contraintes : les contraintes dites impératives (CI) et les contraintes électromagnétiques (CEM). Comme leur nom l'indique les CI doivent être respectées pour que l'affectation soit valide. En ce qui concerne les CEM, le FAPP présente onze niveaux de relaxation. Nous indiquons par la lettre k le niveau de relaxation traité. En d'autres termes ces onze niveaux constituent onze problèmes imbriqués, du plus facile ($k = 10$) au plus difficile ($k = 0$). Finalement, les termes du challenge spécifient que le temps de calcul est borné par une heure sur une machine test, à savoir un PIII à 500MHz, avec 128Mo de RAM.

Cette partie vise à présenter la méthode SIG économique en temps de calcul, pour utiliser au mieux les ressources autorisées par les termes du challenge. Nous présentons une implémentation de SIG spécialement dédiée au FAPP, utilisant des variables complexes, appelées liens ou connexions. Nous comparons également différents "add-ons" au SIG de base. Nous présentons dans la partie 8.2 le framework général de SIG. Puis, nous présentons dans la partie 8.3 les spécificités de SIG pour le problème : la terminologie du FAPP et le codage. Dans la section 8.4 figure l'ensemble des résultats expérimentaux sur les différentes variantes de notre algorithme.

8.2 Schéma général de SIG

Les algorithmes gloutons sont une des méthodes les plus simples pour résoudre les problèmes difficiles. Culberson *et al.* présentent une méthode appelée Iterated Greedy (IG) [Cul92] pour le problème de coloration de graphe. L'idée principale de cette méthode est qu'une configuration (pour le problème qu'il étudie une coloration valide) induit des ordres spéciaux sur les sommets : à l'aide d'un de ces ordres, il est possible de reconstruire une configuration de meilleure qualité (au sens large).

[1] Centre d'ÉLectronique et d'ARmement.
[2] Combinatorial ALgorithms for Military Applications.
[3] http ://www.inra.fr/bia/T/schiex/Doc/CELARE.html
[4] http ://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2001/challenge2001.html

Par ailleurs, Potter *et al.* utilisent différentes espèces coopérantes [PD94] dans un algorithme génétique. Techniquement, ils considèrent indépendamment les différentes variables de leurs problèmes et utilisent la fonction objectif pour choisir localement la valeur à affecter.

Split Iterated Greedy (SIG) combine les deux idées précédentes. Premièrement, on passe d'une configuration à une autre en remettant éventuellement en cause la valeur de toutes les variables. Deuxièment, on attribue pour chaque variable la valeur qui semble être la plus profitable. Pour décrire SIG, nous utilisons les notations suivantes :

- $\{conf.\}$ représente l'ensemble des configurations. On considère que cet ensemble posséde une relation d'ordre notée $<$.
- $\{variables\}$ représente l'ensemble des variables du problème. Nous verrons que dans notre approche du FAPP, cet ensemble est non-trivial (voir section 8.3).
- $\{values\}_v$ représente l'ensemble des valeurs que peut prendre la variable v.
- Pour $X \in \{conf.\}$, on note $X(v)$ la valeur affectée à la variable v dans la configuration X.

SIG itère une amélioration de la configuration courante (voir algorithme 8) tant qu'un critère d'arrêt n'est pas atteint (prédicat $not - ended?()$). Pour le FAPP le critère d'arrêt consiste seulement à déterminer s'il reste encore du temps.

Algorithme 8 Split Iterated Greedy.

Require: $eval, best \in \{conf.\}$
Require: $w_{best} \in \bigcup_v \{values\}_v$
1: $eval \leftarrow best$
2: **while** $not - ended?()$ **do**
3: **for** $v \in \{variables\}$ **do**
4: **for** $w_v \in \{values\}_v$ **do**
5: $eval(v) \leftarrow w_v$ // modification de la valeur de v dans la configuration $eval$
6: **if** $eval < best$ // test de la qualité de la modification **then**
7: $w_{best} \leftarrow w_v$
8: **end if**
9: **end for**
10: $best(v) \leftarrow w_{best}$ // mis à jour de $best$: la valeur de v dans $best$ est la meilleure
11: $eval(v) \leftarrow w_{best}$ // mis à jour de $eval$: $best$ et $eval$ coïncide
12: **end for**
13: **end while**

Comparons maintenant SIG avec les autres algorithmes de recherche locale. Généralement, dans les méthodes de descente (type Hill Climbing (HC) ou Recherche Taboue (TS)), on observe tous les mouvements possibles avant d'en choisir un[5].

[5] Vers l'une des meilleures configurations par exemple.

Cette étape est répétée jusqu'à ce qu'un "critère d'arrêt" soit atteint. Par conséquent, des mouvements précédemment écartés peuvent être exécutés par la suite dans une nouvelle investigation d'un voisinage. SIG décide donc plus vite d'exploiter les bons mouvements, puisqu'il peut effectuer jusqu'à "nombre de variables" mouvements améliorant la configuration. Dans le recuit simulé (SA), seul un mouvement est choisi, puis l'heuristique décide s'il est appliqué à la configuration courante. Si peu de mouvements améliorent la configuration, alors SA rencontre des difficultés à les trouver et donc à les exploiter ; contrairement à SIG qui trouve toujours au moins un mouvement améliorant la configuration s'il en existe un, puisqu'il visite le cas échant tous les voisins d'une configuration.

Quoiqu'il en soit, le SIG basique peut être bloqué dans un optimum local. En effet, si aucune modification locale ne peut améliorer la configuration courante, SIG n'effectuera aucun changement sur la configuration et sera donc bloqué de la même façon que HC. Nous verrons dans la section 8.4 des mécanismes inspirés de TS pour modifier SIG.

8.3 Réalisation

Dans cette section, nous présentons les mécanismes du SIG dédiés au FAPP. Pour cela, nous utilisons les notations suivantes :

- $\{path\}$ est l'ensemble des variables élémentaires d'un problème. Pour ne pas confondre les éléments de $\{path\}$ avec la notion de variable utilisée par notre algorithme, nous reprenons le terme de l'énoncé du challenge : un élément de cet ensemble est appelé un chemin du réseau.
- k représente le niveau de relaxation courant.
- pour une configuration C et un chemin v, on note $C(v).f$ (resp. $C(v).p$) la fréquence (resp. la polarité) affectée au chemin v dans la configuration C.
- pour deux chemins p_1 et p_2 et pour un niveau de relaxation k, on note $\gamma^k_{p_1,p_2}$ (resp. $\delta^k_{p_1,p_2}$) l'écart nécessaire entre les fréquences affectées à p_1 et p_2 en cas d'égalité de polarité (resp. de polarités différentes).
- $\{link\}$ est une partition de l'ensemble des chemins, construite en regroupant les chemins connectés directement ou indirectement par des contraintes d'égalité. On appelle une telle réunion de chemins un lien ou une connexion. Il y a généralement deux voire quatre chemins dans un lien.

Notre algorithme a pour objectif principal de réduire le niveau de relaxation k. Il procède par étapes successives. On initialise k à 10 puis on cherche une configuration respectant toutes les contraintes à partir d'une configuration quelconque. Quand la recherche réussit, on décrémente k, et on réitère le mécanisme à partir de la même configuration. Ce procédé est répété tant qu'il nous reste du temps. Notons que pour le moment, nous ne distinguons pas les CEM des CI dans le calcul de la fonction

coût. La fonction H définie par 8.1 sera notre fonction d'évaluation au cours de chaque étape.

$$\begin{aligned} H : \{conf.\} &\rightarrow \mathbb{N} \\ c \mapsto H(c) &= \sum_{\substack{p_1 \in \{path\} \\ p_2 \in \{path\}}} co(p_1, p_2) \end{aligned} \tag{8.1}$$

où $co(p_1, p_2)$ est le nombre de violations de contraintes entre p_1 et p_2.

Dans la présentation de SIG, nous avons parlé des variables du problème sans spécifier ce qu'il en était sur le FAPP. Dans notre implémentation de SIG pour le FAPP, une variable n'est pas un unique chemin. En fait, nous utilisons comme variable un groupe de chemins reliés par une contrainte d'égalité de distance. De telles contraintes font partie des CI et doivent donc être vérifiées dans la solution proposée. Nous avons constaté qu'il est difficile de les réparer sans augmenter largement le nombre de contraintes violées. De plus, affecter une valeur à un chemin d'une connexion restreint considérablement le domaine des autres chemins de la connexion : il y a donc au plus deux valeurs possibles pour les autres chemins[6], ce qui réduit d'autant l'espace de recherche. En d'autres termes pour une contrainte $|C(i).f - C(j).f| = \epsilon_{ij}$, fixer la valeur $C(i).f$ contraint $C(j).f$ à prendre ses valeurs dans $\{C(i).f + d_{ij}, C(i).f - \epsilon_{ij}\} \bigcap F_{\phi(j)}$, où $F_{\phi(j)}$ est le domaine de j.

Le dernier point nécessaire pour l'implémentation de SIG pour le FAPP est l'ordre de parcours des variables (ligne 5 de l'algorithme 8). Pour plus d'équité entre les variables, nous avons choisi de mélanger l'ordre de parcours des liens entre chaque itération.

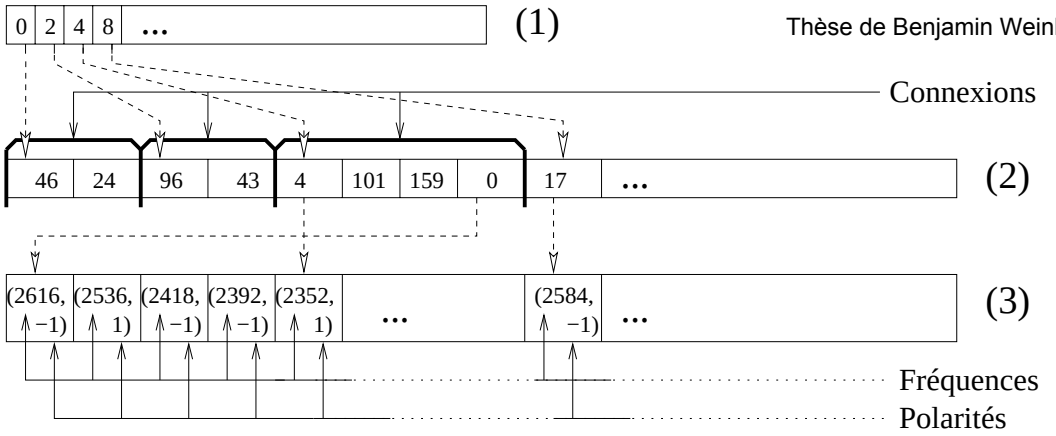
Pour coder une configuration, nous regroupons les chemins des liens à l'aide d'une permutation. Les chemins appartenant à une même connexion sont rangés consécutivement dans le tableau (2) de la figure 8.2. Nous avons encore besoin d'un tableau supplémentaire pour pointer sur les débuts de chaque lien (voir figure 8.2).

Notons que pour le FAPP, la fonction coût peut être calculée de manière incrémentale :

$$\begin{aligned} h_l : \{conf.\} \times \{link\} &\rightarrow \mathbb{N} \\ c, l \mapsto h_l(c, l) &= \sum_{\substack{p_1 \in l \\ p_2 \in \{path\}}} co(p_1, p_2) \end{aligned} \tag{8.2}$$

La fonction objectif est adaptée pour bien réduire le nombre de violations de contraintes, mais s'avère insuffisante pour guider la méthode vers de bonnes régions. Dans un tel cas, Duvivier *et al.* [DPF+98] proposent d'utiliser une seconde fonction objectif pour discriminer les configurations de même fitness. Cette méthode est utilisée pour franchir les plateaux. Nous avons testé plusieurs fonctions, toutes basées sur la différence entre la séparation requise et la séparation effective des valeurs

[6] Cette technique est présenté dans le site http ://www.inra.fr/bia/T/schiex/Doc/CELARE.html



Le tableau (2) regroupe les chemins par connexion. La plupart des connexions sont constituées de deux chemins. Dans le cas contraire, ils restent malgré tout de petites tailles (4 ou 6). Le tableau (1) est nécessaire pour traiter ces rares exceptions, en indiquant les indices du début de chaque lien.

FIG. 8.2 – Codage d'une configuration.

affectées aux chemins. Nous les présentons plus précisément et nous les comparons dans la partie suivante.

8.4 Expérimentations

Dans cette partie, nous présentons nos résultats sur FAPP obtenus grâce à SIG ainsi que ses différentes variations. En particulier, nous présentons des relevés obtenus par l'algorithme de base puis par trois méthodes reposant sur l'usage de fonctions discriminantes. Pour finir, nous présentons deux modifications de SIG utilisant des mécanismes tabous pour améliorer la diversification. La présentation des différentes méthodes est faite de manière incrémentale.

Pour comparer les différentes méthodes, nous avons effectué cinq exécutions sur un échantillon choisi parmi le jeu de tests du challenge. Chaque exécution était limitée à une demi-heure sur un PC Pentium III cadencé à 933 MHz et ayant 128Mo de RAM. Pour chaque expérience, nous rassemblons les résultats pour les différentes instances en précisant :

- k : le meilleur niveau de relaxation obtenu.
- $\left|V^{k-1}\right|$: le nombre de violations de CEM au niveau inférieur pour la meilleure configuration trouvée.
- échec : le nombre d'exécutions où le niveau de relaxation 10 n'a pas été résolu. Nous utilisons cette information comme un indicateur de la robustesse de la

méthode. Rappelons que le premier objectif du problème est de trouver une configuration ne présentant aucune violation de contrainte sur les CI.

instance	k	$\lvert V^{k-1}\rvert$	échec
fapp01 (200)	5	19	0
fapp04 (300)	4	29	1
fapp07 (600)	9	125	4
fapp10 (900)	10	163	4
fapp15 (3000)	11	19	5
fapp17 (300)	4	4	2
fapp19 (350)	11	0	5
fapp23 (1800)	11	0	5
fapp29 (2900)	11	0	5
test04 (1250)	10	282	4

Dans ce tableau nous avons ajouté entre parenthèses, après les noms des instances, le nombre de chemins.

TAB. 8.1 – Résultats pour SIG de base.

Les résultats présentés dans la table 8.1 constitue la référence pour toutes nos comparaisons des différentes variantes de SIG. Le niveau de relaxation des réseaux de fapp19, fapp23 et fapp29 reste à 11 malgré la valeur nulle de $\left|V^{k-1}\right|$; ceci signifie qu'il reste des CI non résolues dans la meilleure configuration trouvée.

8.4.1 Fonctions discriminantes

Il est possible de discriminer de plusieurs façons les configurations présentant le même nombre de violations de contraintes. Nous en avons retenues trois. Il est important de noter que ces critères sont heuristiques et que rien ne prouve *a priori* qu'il vont effectivement fournir de meilleurs résultats. Le premier critère vise à économiser les ressources électromagnétiques. Pour cela, les valeurs affectées à des variables connectées par des contraintes doivent être aussi proches que possible. Par conséquent, SIG cherchera à minimiser la fonction h_2 suivante lors de l'affectation de chaque variable :

$$\begin{aligned} h_2 : \{conf.\} \times \{link\} &\to \mathbb{N} \\ C, l \mapsto h_2(C,l) &= \sum_{\substack{p_1 \in l \\ p_2 \in \{path\}}} s(C, p_1, p_2) \end{aligned} \tag{8.3}$$

où

$$s(C,p_1,p_2) = \begin{cases} |C(p_1).f - C(p_2).f| - \gamma^k_{p_1,p_2} & \text{si } C(p_1).p = C(p_2).p \\ |C(p_1).f - C(p_2).f| - \delta^k_{p_1,p_2} & \text{si } C(p_1).p \neq C(p_2).p \end{cases} \tag{8.4}$$

	(a)			(b)			(c)		
instance	k	$\lvert V^{k-1}\rvert$	échec	k	$\lvert V^{k-1}\rvert$	échec	k	$\lvert V^{k-1}\rvert$	échec
fapp01	5	21	0	4	20	2	5	26	0
fapp04	2	21	1	3	22	0	2	20	0
fapp07	9	123	4	10	159	4	9	123	2
fapp10	6	106	2	7	114	4	8	133	4
fapp15	11	25	5	11	6	5	11	18	5
fapp17	11	0	5	4	4	4	4	4	1
fapp19	11	47	5	11	0	5	9	16	4
fapp23	11	0	5	11	0	5	11	0	5
fapp29	11	1	5	11	0	5	11	15	5
test04	9	198	2	10	270	2	9	235	2

– *(a) : guidage de SIG par minimisation de h2 ;*
– *(b) : guidage de SIG par maximisation de h2 ;*
– *(c) : guidage de SIG par minimisation de la variance de h2.*

TAB. 8.2 – Tableau récapitulatif des différentes fonctions discriminantes.

Les résultats concernant cette variante sont présentés dans le tableau 8.2 (a). Nous obtenons uniquement de meilleurs résultats pour les réseaux fapp04 et fapp10. Par ailleurs, la qualité de certains réseaux décroît, en particulier pour le fapp17. On observe également que minimiser h_2 n'améliore pas la robustesse de notre algorithme. En somme, nous obtenons des résultats moins bons que prévu.

Dans la seconde expérience nous avons utilisé h_2 d'une autre façon. En effet, puisque h_2 mesure la marge entre la contrainte et l'affectation effective, nous pouvons maximiser cette marge afin d'accroître la flexibilité de l'affectation pendant la recherche. D'ailleurs quand un niveau de relaxation est gagné, il est intéressant de disposer de telles marges sur l'ensemble des CEM.

Le tableau 8.2 (b) présente les résultats obtenus. Nous constatons que cette utilisation de h_2 est plus efficace. Seule le fapp07 présente plus de difficultés à être résolu avec cette méthode qu'en utilisant SIG. Par ailleurs, le niveau de relaxation est amélioré pour cinq réseaux. D'un point de vue de la robustesse, les résultats ne présentent guère de progrès significatifs.

La dernière manière d'utiliser h_2 que nous avons testée consiste à essayer d'équilibrer les différentes marges sur l'ensemble du réseau. Pour cela, nous cherchons à réduire la variance des marges. Nous procédons de la manière suivante : nous commençons par estimer la marge totale H_2 en sommant toutes les marges sur les liens. Puis Nous choisissons la valeur qui minimise la fonction suivante :

$$\left| H_2(C) - \frac{nbPaths \times h_2(C,v)}{|v|} \right| \tag{8.5}$$

Le coefficient $\frac{nbPaths}{|v|}$ tend à équilibrer le poids des différents liens, puisque les liens ne contiennent pas toujours le même nombre de chemins.

Les relevés pour cette méthode apparaissent dans le tableau 8.2 (c). On constate que tous les réseaux obtiennent un meilleur niveau de relaxation avec la variante avec diminution de la variance que la version de base de SIG. De plus, on constate également une diminution du niveau de relaxation sur quatre réseaux et les meilleures améliorations sur fapp04 et fapp19. Du point de vue de la robustesse on constate aussi que cette technique est intéressante, en particulier sur fapp07 et fapp17 où les deux versions précédentes y présentaient des faiblesses. En somme, cette méthode améliore plus souvent l'algorithme SIG basique.

8.4.2 Mécanismes de diversification

Jusqu'à présent les mécanismes additionnels guident la recherche mais SIG reste une recherche locale. Par conséquent, il peut rester bloqué dans un optimum local. Pour permettre de quitter ces optima, nous utilisons le mécanisme taboue [GL93].

Dans les recherches taboues classiques (TS), on considère tous les mouvements possibles indépendamment, ce qui n'est pas le cas pour SIG. Par conséquent dans TS, on empêche de défaire un mouvement en inscrivant le mouvement contraire dans une liste taboue. Or dans SIG, on sait que le mouvement suivant traitera une autre variable ; par conséquent, on sait que le mouvement ne sera pas défait à l'itération suivante. Cette interdiction tient pendant un nombre d'iterations appelé "la longueur de liste taboue" ($|lt|$ en abrégé). Plus précisément, on sait que deux mouvements traitant la même variable sont séparés par un nombre de mouvements égal au nombre de variables du problème (ici le nombre de liens). Donc Tabu Split Iterated Greedy (TSIG) doit prendre en compte ce fait et doit avoir une longueur de liste taboue suffisamment courte pour permettre éventuellement de réaffecter des bonnes valeurs qui auraient changé.

Dans notre cas, nous avons utilisé comme liste taboue un tableau de dates tridimensionnel (indicé par les chemins, les fréquences et les polarités). Quand un mouvement est appliqué, la procédure $setTabu()$ marque, pour tous les chemins d'une connexion et leurs anciennes ressources électromagnétiques affectées (fréquences et polarités), la case de la liste taboue d'indice c, f, p avec la valeur $c\_date + |lt|$, où c_date est la valeur courante de la date. La fonction $isTabu?()$ retourne vrai si les valeurs indicées dans la liste taboue par tous les chemins du lien et les ressources qu'on souhaite affecter sont inférieures à c_date. L'algorithme 9 présente comment on intègre les mécanismes tabous à SIG pour obtenir TSIG.

Les résultats obtenus par TSIG sont présentés dans 8.3 (a), pour $|lt|$ fixée expérimentalement à 3. Nous avons bien évidemment utilisé la troisième version de la fonction discriminante pour nos expérimentations.

Algorithme 9 Tabu Split Iterated Greedy.

Require: $eval, current, best \in \{config.\}$
Require: $c\_date \in \mathbb{N}$
Require: $w_{best} \in \{values\}$
1: $eval \leftarrow current$
2: $c\_date \leftarrow 0$
3: **while** $not - ended?()$ **do**
4: **for** $v \in \{variable\}$ **do**
5: **for** $w_v \in \{value\}$ **do**
6: **if** $not\ isTabu?(v, w_v)$ **then**
7: $eval(v) \leftarrow w_v$ // modification au niveau de la variable v
8: **if** $eval < current$ // de la modification **then**
9: $w_{best} \leftarrow w_v$
10: **end if**
11: **end if**
12: **end for**
13: $setTabu(v, w_{best})$ // le retour est marqué comme taboue
14: $current(v) \leftarrow w_{best}$ // modification est validé sur la configuration courante
15: $eval(v) \leftarrow w_{best}$ // eval est mis à jour pour coïncider avec la configuration courante
16: **if** $current < best$ **then**
17: $best \leftarrow current$ // mémorisation de la meilleure configuration obtenue
18: **end if**
19: **end for**
20: $c\_date \leftarrow c\_date + 1$
21: **end while**

Les résultats que nous avons obtenus sont moins bons que tous ceux que nous avions déjà eus. Ce phénomène s'explique par le fait que TSIG change l'affectation de chaque lien à chaque itération ; ceci malgré le fait que certaines valeurs affectées puissent être optimales. En d'autres termes TSIG diversifie trop et donc la configuration n'a pas la possibilité de converger vers une bonne solution. Afin de pallier à ce défaut, nous utilisons un critère d'aspiration. On modifie pour cela TSIG en ATSIG (Aspiration Tabu Split Iterated Greedy), en remplaçant les lignes 9 à 14 de l'algorithme précédent par les instructions suivantes :

9': $eval(v) \leftarrow w_v$
10': **if** $not\ isTabu?(v, w_v)\ OR\ eval < best$ **then**
11': **if** $eval < current$ **then**
12': $w_{best} \leftarrow w_v$
13': **end if**
14': **end if**

Ici le critère d'aspiration ne prend en compte que la fonction fitness h_1. Les résultats obtenus sont présentés dans la partie (b) du tableau 8.3. Pour nos expérimentations nous avons augmenter $|lt|$ jusqu'à 7.

	(a)			(b)		
instance	k	$\|V^{k-1}\|$	échec	k	$\|V^{k-1}\|$	échec
fapp01	7	16	0	4	15	0
fapp04	9	34	1	1	217	0
fapp07	11	6	5	9	115	2
fapp10	11	7	5	7	90	4
fapp15	11	68	5	11	33	5
fapp17	5	4	3	4	4	1
fapp19	11	9	5	9	11	3
fapp23	11	24	5	11	0	5
fapp29	11	228	5	11	0	5
test04	11	18	5	9	194	2

- *(a) : ajout d'une liste taboue (TSIG) ;*
- *(b) : ajout d'une liste taboue balancée par un critère d'aspiration (ATSIG).*

TAB. 8.3 – Résultats des versions taboues de SIG

Bien qu'il nécessite plus de temps de calcul à chaque itération (limitant d'autant le nombre d'itérations effectuées), on constate que ATSIG fournit de meilleurs résultats que TSIG. En particulier, on constate que ATSIG trouve des affectations valides pour tous les petits réseaux (nombre de chemins inférieurs à 900). En comparant ATSIG avec la troisième variante de SIG, on constate que ATSIG atteint généralement de meilleures configurations. Sur l'ensemble des méthodes développées, seule la première variante de SIG obtient une solution de meilleur niveau de relaxation pour un réseau (le fapp10).

8.5 Conclusion

Dans cette partie, nous avons présenté SIG, une métaheuristique locale efficace et économique en temps de calcul. Afin de guider l'heuristique dans l'espace de recherche, nous avons suggéré l'utilisation de fonctions pour discriminer les configurations ayant la même valeur de fonction objectif. Nous avons comparé trois fonctions discriminantes, pour retenir celle réduisant la variance de la marge sur les différentes contraintes. Pour prévenir la convergence inopinée de SIG vers un optimum local, nous avons testé des combinaisons de SIG avec des mécanismes taboues : TSIG et

ATSIG. Après avoir constaté que TSIG diversifiait trop, nous avons ajouté un critère d'aspiration à l'algorithme (ATSIG). Nous avons montré que l'ajout des deux mécanismes adaptatifs améliorait les meilleures performances de l'algorithme, bien que nous n'ayons pas pu constater d'amélioration significative au niveau de la robustesse.

Bibliographie

[AH77] K. Appel and W. Haken. Every planar map are four colorable. *Illinois J. math*, 21 :429–490, 1977.

[AHZ03] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151 :379–388, 2003.

[Akr94] A. Akrout. Problème d'affectation de fréquences : méthodes basées sur le recuit simulé. Technical report, CNET, 1994.

[Bac99] V. Bachelet. *Métaheuristique parallèle hybride : application au problème d'affectation quadratique*. PhD thesis, Université des sciences et technologies de Lille, cité scientifique Villeneuve d'Ascq 59655, december 1999.

[BB03] N. Barnier and P. Brisset. Coloriage de graphe en programmation par contraintes. In *ROADEF 2003*, pages 348–349. ROADéF, February 2003.

[BBD+95] A. Bouju, J. Boyce, C. Dimitropoulis, G. Scheidt, and J. Taylor. Tabu search for the radio link frequency assignment problem. In *Applied Decision Technologies, [ADT95]*, London, 1995. UNICOM Conference.

[BC01] A. Beacham and J. Culberson. On the complexity of unfrozen problems. In *Workshop on Computational Complexity and Statistical Physics*, Santa Fe, New Mexico, USA, september 2001.

[Bre79] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256, 1979.

[Bri92] P. Briggs. *Register Allocation via Graph Coloring*. PhD thesis, Rice University, April 1992.

[Bri98] P. Briggs. Register allocation via graph coloring. Technical Report TR92-183, Rice University, 24, 1998.

[BRJ99] G. Booch, J. Rumbauch, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Professional, 1999.

[CAC+81] G.J. Chaitin, M. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P. Markstein. *Computer Languages*, chapter Register allocation

via graph coloring, pages 47–57. IBM T. J. Watson Research Center, 1981.

[CG99] J. Culberson and I.P. Gent. Well out of reach : why hard problems are hard. Technical Report APES-13-1999, APES Research Group, 1999.

[CG01] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1–2) :227–264, 2001.

[CH97] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48 :295–305, 1997.

[CHD95] D. Costa, A. Hertz, and O. Dubuis. Embedded of sequential procedure within an evolutionary algorithm for coloring problems in graphs. Technical report, École Polytechnique Fédérale de Lausanne, 1995.

[Cle01] M. Clerc. Optimisation par essaim particulaire et coloriage de graphe. Technical report, France Télécom, 2001.

[CMT04] S. Cahon, N. Melab, and E-G. Talbi. ParadisEO : A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 2004. To appear.

[CMTS03] S. Cahon, N. Melab, E-G. Talbi, and M. Schoenauer. "ParadisEO based design of parallel and distributed evolutionary algorithms". In *Evolutionary Algorithms EA'2003*, Marseille, France, October 2003. LNCS.

[Cro58] G.A. Croes. A method for solving the traveling salesman problems. *Operations Research*, 6 :791–812, 1958.

[CS02] M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring. In D. S. Johnson, A. Mehrotra, and M. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, Ithaca, New York, USA, September 2002.

[CS03] P. Collet and M. Schoenauer. Guide : Unifying evolutionary engines throught a graphical user interface. In *Artificial Evolution EA'03*, pages 208–220, October 2003.

[CT80] G. Carpaneto and P. Toth. Algorithm 548 : Solution of the assignment problem. *ACM Transactions on Mathematical Software (TOMS)*, 6(1) :104–111, 1980.

[CTM03] S. Cahon, E-G. Talbi, and N. Melab. Paradiseo : a framework for parallel and disributed biologically inspired heuristics. In *IPDPS*, Nice, France, April 2003. IEEE.

[Cul92] J. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical report, University of Alberta, June 1992.

[Cul96] J.C. Culberson. On the Futility of Blind Search. Technical Report 96–18, Dept Computer Science, University of Alberta, Edmonton, Alberta, Canada, 1996.

[Dav85] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Inteligence*, pages 162–164, 1985.

[Def00] T. Defaix. Challenge roadef 2001, Octobre 2000. http ://www.ensta.fr/uer/uma/conf/roadef-2001-challenge/index_main.html.

[DJW99] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. In W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Proceedings of the First Genetic and Evolutionary Computation Conference(GECCO '99)*, pages 833–839, San Francisco CA, 13–17 1999. Morgan Kaufmann Publishers, Inc.

[DJW02] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics-the (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1) :131–144, september 2002.

[Dor98] R. Dorne. *Étude des méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquence*. PhD thesis, Université de Montpellier II Science et Technique, May 1998.

[DPF+98] D. Duvivier, P. Preux, C. Fonlupt, D. Robilliard, and E-G. Talbi. The fitness function and its impact on local search methods. In *IEEE System, Man, and Cybernetics*, pages 2478–2483, San Diego, USA, October 1998.

[Eng04] T. English. titi. In J. Gottlieb and G. Raidl, editors, *EVOcop*, Coimbra, April 2004. LNSC.

[Err27] A. Errera. Exposé historique du problème des quatre couleurs. In *Periodico di Matematische*, pages 20–41, juillet 1927.

[EV98] A.E. Eiben and J.K. Van Der Hauw. Adaptive penalties for evolutionary graph coloring. *Lecture Notes in Computer Science*, 1363 :95–106, 1998.

[EvdHvH98] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1) :25–46, 1998.

[Fal98] E. Falkenauer. *Genetic Algorithm and Grouping Problems*. John Wiley & Sons, 1998.

[FF96] J. Fleurent and J.A. Ferland. *Cliques, Coloring and Satisfiability : Second DIMACS Implementation challenge*, volume 26, chapter C :

Object-Oriented Implementation of Heuristic Search Methodes for Graph Coloring Problem, Maximum clique and Staisfiability, pages 619–652. DIMACS series in Discret Mathematics and Theoritical Computer Science, 1996.

[FLS01] D. Fotakis, S. Likothanassis, and S. Stefanakos. An evolutionary annealing approach to graph coloring. In *Proceedings of Applications of Evolutionary Computing*, number 2037 in LNCS, pages 120–129. Evo-Workshops 2001, Springer-Verlag, April 2001.

[FR95] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6 :109–133, 1995.

[FRPT99] C. Fonlupt, D. Robillard, P. Preux, and E-G. Talbi. *Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization*, chapter Fitness Landscape and performance of meta-heuristics, pages 257–268. Kluwer Academic, 1999. ISBN 0-7923-8369-9.

[Gal99] P. Galignier. *Étude des métaheuristique pour la résolution du problème de statisfaction de contraintes et de la coloration de graphes.* PhD thesis, Université de Montpellier II, 1999.

[Gam86] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1) :8–14, 1986.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine, A Users' Guide and a Tutorial Networked Parallel Computing.* The MIT Press, Cambridge, Massachusetts, 1994.

[GH99] P. Galinier and J-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4) :379–397, 1999.

[Gir01] M. Giritli. From 3-sat to {2+p},{3}-sat. Technical report, ILLC, Amsterdam, Nederlands, November 2001.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Interactability. A Guide to the Theory of NP-Completeness.* A Series of Books in the Mathematical Sciences. Freemann And Company, 1979.

[GL85] D.E. Goldberg and R. Lingle. Alleles loci and the tsp. In *Proceeding of the First International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, 1985. Lawrence Erlbaum Associates.

[GL93] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

[Glo77] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1) :156 – 166, 1977.

[Gol94] D E. Goldberg. *Algorithmes génétiques. Exploration, optimisation et apprentissage automatique*. Paris Addison-Wesley, 1994.

[GPB98] C.A. Glass and A. Prügel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. Technical report, Departement of Electronics and Computer Science, University of Southampton, 1998.

[Gre93] J.J. Grefenstette. Deception considered harmful. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, San Mateo, CA, 1993.

[GS03] L. Di Gaspero and A. Schaerf. *Software : Practice and Experience*, volume 33, chapter EASYLOCAL++ : an object-oriented framework for the flexible design of local-search algorithms, pages 733–765. John Wiley & Sons, 2003.

[HdW87] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(2) :345–351, 1987.

[HH01a] J-P. Hamiez and J-K. Hao. An analysis of solution properties of the graph coloring problem. In J. de Sousa, editor, *Metaheuristic International Conference, MIC*, pages 193–198. Kluwer, 2001.

[HH01b] J-P. Hamiez and J-K. Hao. Scatter search for graph coloring. In *Artificial Evolution*, pages 267–278, Le Creusot, France, October 2001.

[HSV98] S. Hurley, D. Smith, and C. Valenzuela. A permutation based genetic algorithm for minimum span frequency assignment. In T Baeck, A Eiben, M Schoenauer, and H Schwefel, editors, *PPSN V : Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 907–916, Amsterdam, The Netherlands, September 1998. Springer-Verlag Publication.

[IT01] C. Igel and M. Toussaint. On classes of functions for which No Free Lunch results hold. Technical report, Institut für Neuroinformatik, Ruhr-Universität Bochum, ND 04 44780 Bochum - Germany, 2001.

[IT04] C. Igel and M. Toussaint. A non-free-lunch theorem for non-uniform distributions target functions. *Journal of Mathematical Modelling and Algorithms*, 2004. to Appear.

[JAMS91] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing : An experimental evaluation ; part II, graph coloring and number partitioning. *Operations Research*, 39(3) :378–406, may-june 1991.

[Jon95] T. Jones. One operator, one landscape. Technical report, Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA, 1995.

[Jou03] L. Jourdan. *Métaheuristique pour l'extraction de connaissances : application à la génomique*. PhD thesis, Université des Sciences et technologie de Lille, 2003.

[JT95] T. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1995.

[Kar72] R. M. Karp. *Complexity of computer computations*, chapter Reducibility among combinatorial problems, pages 85–103. Plenm Press New York, 1972.

[Kem79] A.B. Kempe. On the geographical problem of four colors. *Amer. J. Math*, pages 193–200, 1879.

[KGV83] S. Kirkpatrick, D.C. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, May 1983.

[KMRS01] M. Keijzer, J.J. Merelo, G. Romero, and M. Schoenauer. Evolving objects : a general purpose evolutionary computation library. In *EA-01, Evolution Artificielle, 5th International Conference in Evolutionary Algorithms*, LNCS, pages 231–244. Springer-Verlag, 2001.

[Krz01] W. Krzysztof. Graph coloring using ant algorithms. In *Conference on Computer Recognition Systems KOSYR 2001*, pages 199–204, Miłkow, May 28-31 2001. Wrocław : Oficyna Wydaw. Politechniki Wrocławskiej 2001.

[Lei79] F.T. Leighton. A graph coulouring algorithm for large scheduling problems. *J. Res Natl Bur. Standards*, 84 :489–506, 1979.

[Lov73] L. Lovàsz. Coverings and colorings of hypergraph. In *4th S-E conferenece on Combinatorics, Graph Theory and Computing*, 8, pages 3–12, Boca Raton, 1973.

[LY93] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *25'th IEEE Symposium on The Theory of Computing*, pages 286–293. ACM, 1993.

[Mah95] S. Mahfoud. *Niching Methods for Genetic Algorithm*. PhD thesis, Universty of Illinois, 1995.

[MC99] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *Knowledge and Data Engineering*, 11(5) :769–778, 1999.

[MD00] A. Marino and R.I. Damper. Breaking the symmetry of the graph colouring problem with genetic algorithms. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 240–245, Las Vegas, Nevada, USA, april 2000.

[MEY95] S. Miner, S. Elmohamed, and H. Yau. Optimizing timetabling solutions using graph coloring. In *NPAC REU program, NPAC*, Syracuse University, NY, 1995.

[MH01] Laurent Michel and Pascal Van Hentenryck. Localizer : An open library for local search. Technical Report CS-01-02, Brown University, 2001.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs, third, revised and extended Edition*. Springer-Verlag, Berlin Heidelburg NewYork, 1996. ISBN 3-540-60676-9.

[Mit02] J.E. Mitchell. *Handbook of Applied Optimization*, chapter Branch-and-Cut Algorithm for Combinatorial Optimization Problem, pages 65–77. Oxford University Press, 2002.

[MPBG99] A. Marino, A. Prugel-Bennett, and C. Glass. Improving graph colouring with linear programming and genetic algorithms, 1999.

[MT03] F. Maffray and N. Trotignon. Colorer par contraction : les graphes parfaitement contractiles. In *ROADéF 2003*, pages 321–321, February 2003.

[MV93] L. Ming and P.M.B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

[Oga86] H. Ogawa. Labeled point pattern matching by delaunay triangulation and maximal cliques. In *Pattern Recognition*, number 1 in 19, pages 35–40, 1986.

[OSH87] I.M. Oliver, D.J. Smith, and J.R.C Holland. A strudy of permutation crossover operator on the traveling salesman problem. In *Proceeding of the Second International Conference on Genetic Algorithms*, pages 224–230, Hillsdale, 1987. Lawrence Erlbaum Associates.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Co. - Reading, Mass., 1994. ISBN : 0-201-53082-1.

[Par95] J. Paredis. Coevolutionary computation. *Artificial Life*, 2(4) :355–375, 1995.

[PD94] M.A. Potter and K.A. De Jong. A cooperative coevolutionary approach to function optimization. *LNCS*, 866 :249–257, 1994.

[PS02] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002 : EvoCOP, EvoIASP, EvoSTim*, volume 2279, pages 121–130, Kinsale, Ireland, 3-4 2002. Springer-Verlag.

[PW89] A. Petford and D. Welsh. A randomised 3-colouring algorithm. *Discrete Mathematics*, 74 :253–261, 1989.

[Rob] G. Robins. www.cs.virginia.edu/~robins.cs660/cs660_slides_144_157.

[Rud98] G. Rudolph. Finite Markov chain results in evolutionary computation : A tour d'horizon. *Fundamenta Informaticae*, 35(1–4) :67–89, 1998.

[SVW01] C. Schumacher, M.D. Vose, and L.D. Whitley. The No Free Lunch and problem description length. In L. Spector, E. Goodman, A. Wu, W.B.

Langdon, H-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[Tai91] É. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5) :443–455, July 1991.

[Tri03] M.A. Trick. Color02/03 : Graph coloring and its generalizations, 2003. http ://mat.gsia.cmu.edu/COLORING03/.

[VT98] C. Voudouris and E. Tsang. Solving the radio link frequency assignment problem using guided local search, 1998.

[VW02] S. Voss and D.L. Woodruff, editors. *Optimization Software Class Libraries*. Kluwer Academic Publishers, Boston, Hardbound, April 2002.

[VZ00] A. Vesel and J. Zerovnik. How good can ants color graphs ? *Journal of Computing and Information Technology - CIT*, 8 :131–136, 2000.

[Wal01a] C. Walshaw. A Multilevel Approach to the Graph Colouring Problem. Tech. Rep. 01/IM/69, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, May 2001.

[Wal01b] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. Tech. Rep. 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, June 2001.

[WBT00] B. Weinberg, V. Bachelet, and E-G. Talbi. A coevolutionary metaheuristic for the frequency assignment problem. Frequency Assignment Workshop, July 2000.

[WBT01] B. Weinberg, V. Bachelet, and E-G. Talbi. A co-evolutionnist metaheuristic for the assignment of the frequencies in cellular networks. In *First European workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, pages 140–149, Como, Italy, 2001. LNCS.

[WE02] C. Walshaw and M.G. Everett. Multilevel Landscapes in Combinatorial Optimisation. Tech. Rep. 02/IM/93, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, April 2002.

[Wei90] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, pages 325–336, 1990.

[Whi91] L.D. Whitley. Fundamental principles of deception in genetic search. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 221–241. Morgan Kaufmann, San Mateo, CA, 1991.

[Whi00] D. Whitley. Functions as permutations : Implications for no free lunch, walsh analysis and summary statistics. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 169–178, Berlin, 2000. Springer.

[Wig83] A. Wigderson. Improving the performance for approximate graph coloring. *Journal of the ACM*, 30 :729–735, april 1983.

[WM97] D.H. Wolpert and W.G. MacReady. No Free Lunch for optimization. *IEEE Transactions on Evolutionary Computation*, pages 67–82, 1997.

[WN03] J.R. Woodward and J.R. Neil. No free lunch, program induction and combinatorial problems. In C.Ryan et al., editor, *Genetic programming*, LNCS 2610, pages 475–484, Essex, UK, 4 2003. Springer-Verlag.

[Wri32] S. Wright. The role of mutation, inbreeding, crossbreeding, and selection in evolution. In *the Sixth Congress on Genetics*, volume 1, page 365, 1932.

[WS93] E. Weinberger and P. Stadler. Why some fitness landscapes are fractal, 1993.

[WSF89] D. Whitley, T. Starkweather, and D'A. Fuquay. Scheduling and taveling salesman : the genetic edge recombination operator. In *Proceeding of the Third International Conference on Genetic Algorithms*, pages 133–140, San Mateo, 1989. Morgan Kaufman Publishers.

[WT01] B. Weinberg and E-G. Talbi. Résolution du challenge ROADéF par plouton et recherche locale. Francoro III, Mai 2001.

[WT02] B. Weinberg and E-G Talbi. Fitness landscape for metaheuristic design. International Symposium on Combinatorial Optimisation (CO'02), April 2002.

[WT03] B. Weinberg and E-G Talbi. No doom in no free lunch theorem. European Chapter on Combinatorial Optimisation, June 2003.

[WT04a] B. Weinberg and E-G. Talbi. *Handbook of Bioinspired Algorithms and Applications*, chapter A cooperative parallel metaheuristic applied to the graph coloring problem. CRC Press, USA, 2004. To appear.

[WT04b] B. Weinberg and E-G. Talbi. NFL theorem is unusable on structured classes of problems. In *Congress on Evolutionary Computation (CEC)*, Portland, Oregon, June 2004. To appear.

[WT04c] B. Weinberg and E-G. Talbi. On symmetry of partitionning problems. In J. Gottlieb and G. Raidl, editors, *EvoCOP*, pages 230–240. LNCS, 2004.

[Yan97] M. Yannakakis. Computational complexity. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley & Sons Ltd., 1997.

Analyse et résolution approchée de problèmes d'optimisation combinatoire : application au problème de coloration de graphe

résumé : Nous avons exploré plusieurs aspect théorique et expérimentaux de l'optimisation combinatoire. Premièrement, nous avons défini une notion de structure permettant de s'échapper du résultat du théorème du *No Free Lunch*. Deuxièmement nous avons formalisé la symétrie de l'espace de recherche des problèmes de partitionnements. À l'aide de cette formalisation, nous pûmes concevoir des outils travaillant efficacement sur cette espace. Plus précisément nous avons développé un test d'égalité, un mesure de distance et un nouvel opérateur de *Cross over*. Nous avons utilisé ces résultats pour classifier les benchmarks classique de la coloration de graphe. Pour finir, nous avons développé pour ce problème une métaheuristique parallèle qui équilibre l'intensification et la diversification pendant la recherche.

mots clés : No Free Lunch, problème de partitionnements, problème de coloration de graphe, algorithmes évolutionnaires, algorithmes parallèles, opérateurs génétiques, distance.

Analysis and approximated solving of combinatorial optimisation problems : application to graph coloring de problem

Abstract : We explored several theoretical and experimental aspects of combinatorial optimization. First, to escape from the result of the *No Free Lunch* theorem, we defined a concept of structure and shown its existence of such a structure in some classes of problems. Second, we formalized the symmetry of the search space of partitioning problems. Using this formalization, we were able to design tools working efficiently on their search space. More precisely, we defined a equality test, a distance and a new crossover operator. Using these results, we were able to classify standard benchmarks of graph coloring problem. Finally, we designed for this problem, a parallel metaheuristic, which balances the intensification and the diversification during the search.

keywords : No Free Lunch, grouping problèms, graph coloring problèms, evolutionnary algorithms, parallel algorithms, genetic operators, distance.