Académie de Montpellier

Université Montpellier II

- sciences et techniques du languedoc -

Diplôme d'études approfondies

- INFORMATIQUE -

Rapport de stage de recherche

Coloration de classes et de propriétés

Étude algorithmique et heuristique

Pascal TAKHEDMIT

Date de soutenance : 25 Juin 2003

Responsables de stage : Roland Ducournau et Christophe Paul

Remerciement

Je tiens à remercier Roland Ducournau pour toute l'aide qu'il a pu m'accorder pendant ces quatre mois de stage et pour sa très grande disponibilité. En effet, il a toujours su avoir une oreille attentive et patiente pour moi, même à des horaires tardifs. Je remercie également Christophe Paul pour m'avoir aiguillé dans la bonne direction au moment opportun.

Table des matières

Chapitre 1

Introduction et problématique

Le paradigme objet occupe une place de plus en plus importante dans l'informatique moderne (représentation de connaissance, programmation par objets, base de données, etc.). Le succès de ce paradigme est dû, en grande partie, aux fonctionnalités des objets (polymorphisme, généricité, encapsulation, etc.) mais ces fonctionnalités ont un coût. C'est dans ce contexte que l'implémentation des objets est devenu une question cruciale puisqu'elle conditionne, en partie, l'efficacité et l'espace occupé des systèmes informatiques.

La problématique implémentatoire des objets se situe à trois niveaux : le test de sous-typage, la sélection de méthodes et l'accès aux attributs. Ces problèmes peuvent être résolus très efficacement en héritage simple et en typage statique. Malheureusement, en héritage multiple ou en typage dynamique, les problèmes se compliquent et les nombreuses recherches menées ont apporté des solutions pas toujours satisfaisantes.

Dans ce cadre, la coloration est une solution simple et élégante aussi bien en héritage simple et en typage statique, où elle est optimale, qu'en héritage multiple ou typage dynamique, où le surcoût qu'elle engendre est très faible. Cette technique peut être utilisée pour compiler les langages objets à typage statique comme C++ ou Eiffel où elle se révèle meilleure que l'implémentation standard. La globalité de la coloration en constitue le seul handicap, mais elle peut être repoussée à l'édition de liens [Duc02b]. Ce dernier point ne sera pas abordé par la suite.

L'objectif de ce travail est de formaliser le problème de la coloration d'une hiérarchie. Pour cela, nous nous plaçons dans le contexte général des ensembles partiellements ordonnés. Ensuite, nous démontrons que le problème de la coloration d'une hiérarchie et ses diverses variantes appartiennent à la classe des problèmes NP-Difficiles (ce résultat étant encore inédit à ce jour). Enfin, nous proposerons nos propres heuristiques de coloration d'une hiérarchie et comparerons grâce à des fichiers communs (benchmarks) les performances respectives des différentes heuristiques proposées.

Chapitre 2

État de l'art

Dans les langages à objets, l'implémentation d'un objet doit répondre aux trois questions suivantes :

- Comment déterminer si un objet de type A est un objet de type B. (Problème de test de sous-typage)
- Comment sélectionner la méthode correspondant à un message envoyé à un objet. (Problème de sélection de méthodes)
- Comment extraire la valeur d'un attribut d'un objet. (Problème de l'accès aux attributs)

Avec l'héritage simple en typage statique où une classe ne peut hériter directement que d'une seule autre classe, l'implémentation standard se révèle optimale en temps et en espace. Par contre en héritage multiple ou typage dynamique, les solutions se compliquent et entraînent des surcoûts considérables en temps et/ou en espace. Cette partie dresse un état de l'art des trois principaux problèmes et montre que la coloration comme technique d'implémentation est une solution simple et efficace aussi bien en héritage simple et en typage statique où elle est optimale qu'en héritage multiple ou en typage dynamique où le surcoût qu'elle engendre est faible.

2.1 Test de sous-typage

Dans les langages orientés objet, le respect de la théorie des types impose une redéfinition contravariante des paramètres d'une méthode. Cette redéfinition est en contradiction avec une modélisation naturelle du monde réel qui privilégie une redéfinition covariante. Un moyen de contourner ce problème est d'effectuer un casting descendant systématique des arguments vers le type souhaité qui est un sous type du type des paramètres de la méthode. Le problème est alors de vérifier que le type dynamique de l'objet est bien un sous-type du type souhaité. Le problème sous-jacent au test de sous typage est le suivant : étant donné un ordre partiel, il s'agit de déterminer, le plus efficacement possible et avec une taille de codage minimale, s'il existe une relation de comparabilité entre deux éléments. Ce problème est un problème classique de l'algorithmique combinatoire.

2.1.1 Numérotation de Schubert [SPT83]

La numérotation de Schubert est une technique simple qui consiste à définir une double numérotation des éléments de l'ordre qui représente, pour chaque élément, les bornes de l'intervalle contenant les entiers associés à ses descendants. Le test de sous typage $x \leq y$ consiste alors à vérifier que l'entier associé à x appartient à l'intervalle de y.

Cette technique s'applique aux hiérarchies en héritage simple (cf. *Range compression* pour l'héritage multiple) et la double numérotation est obtenue en effectuant un parcours en profondeur de l'arborescence. Pour une classe, la borne inférieure de l'intervalle est obtenue à la descente, après incrémentation du compteur, et la borne supérieure à la remontée, sans incrémentation.

Cette méthode garantit donc un test de sous typage en temps constant et un codage en $2 * log(n)$ bits.

2.1.2 Range compression [ABJ89]

Range compression est une généralisation de la numérotation de Schubert aux hiérarchies quelconques. La double numérotation des types est obtenue en effectuant un parcours en profondeur d'une certaine forêt recouvrante. Dans ce cas, le test de sous typage n'est plus en temps constant comme précédemment mais dépend du nombre d'arbres dans la forêt. Toutefois, cette méthode donne de bons résultats car le nombre d'intervalles est souvent faible. Dans certains cas extrêmes, la recherche peut être réalisée via un arbre de recherche.

2.1.3 Méthode de Cohen [Coh91]

Le principe de cette méthode est de trouver un partitionnement de l'ordre telle que chaque partie ne contienne pas deux éléments possédant un descendant commun. A l'intérieur de chaque partie, chaque élément possède un identifiant unique et un tableau indexé par les parties qui contient les ancêtres du type considéré. Le test de sous typage $x \leq y$ revient alors à déterminer si l'identifiant de y appartient au tableau associé à x. Le partitionnement en niveaux de l'arborescence est une décomposition minimale de l'ordre.

2.1.4 Packed Encoding (PE) [VHK97]

Packed Encoding est une généralisation de la méthode de Cohen aux hiérarchies quelconques. L'algorithme issu de cette méthode traite séparément les éléments du cœur et les éléments constituant la frontière et la couronne [1]. L'algorithme applique ensuite une heuristique gloutonne qui consiste à sélectionner à chaque étape le sommet appartenant au coeur (resp. à la couronne ou à la frontière) possédant le plus petit (resp. le plus grand) nombre de sous-classes et à l'insérer dans la partie autorisée de plus petit rang possible. Si l'insertion échoue, une nouvelle partie est crée contenant l'élément courant. La recherche du nombre minimal de parties est un problème NP-difficile dans le cas général.

[1] cf. définition chap. 3

2.1.5 PQ-Encoding [ZG01]

Cette technique combine les deux approches précédentes. En effet, comme dans l'algorithme de Cohen, cette méthode consiste à partitionner l'ordre de sorte que chaque partie admette une numérotation de Schubert optimale c'est-à-dire une numérotation qui ne nécessite qu'un seul intervalle par élément. Une telle numérotation va être obtenue en utilisant une structure de donnée particulière, les arbres PQ. Dans ce cadre, le test de sous-typage $x \leq y$ consiste à déterminer l'arbre PQ contenant les descendants de y puis de déterminer si l'identifiant de x dans cette arbre appartient à l'intervalle associé à y.

2.1.6 Bit-Vector Encoding

Bit-Vector Encoding est la technique qui a suscité le plus d'intérêt [VHK97, Cas93, HN94]. Elle consiste à associer à chaque type un sous-ensemble d'un ensemble à k éléments tels que la relation de sous-typage soit équivalente à la relation d'inclusion des sous-ensembles. Typiquement, un sous-ensemble d'un ensemble à k éléments peut être représenté par un vecteur de bits de longueur k. Le test de sous-typage $x \leq y$ consiste alors à tester si le vecteur de y est inclus dans le vecteur de x. La minimisation de k est un problème NP-Difficile et non approximable.

2.2 Sélection de méthode et accès aux attributs

Dans les langages à objets, les actions susceptibles d'être effectuées par un objet sont invoquées par envoi de message. Le comportement attendu dépend du type dynamique de l'objet et non pas du type statique dont la seule fonction est de permettre au compilateur d'effectuer des vérifications de base. Le problème est alors de trouver une technique efficace pour sélectionner la méthode la plus spécifique vis-à-vis du type dynamique implémentant le message invoqué. L'accès aux attributs est quasiment équivalent à la sélection de méthodes modulo le fait que les tables sont dynamiques et qu'il n'y a pas de redondances dans la matrice correspondante. Toutefois, des techniques spécifiques ont été développées.

2.2.1 Implémentation standard

L'implémentation standard, bien qu'assez naturelle, peut se révéler inefficace et coûteuse en espace mémoire. En héritage simple, un objet est implémenté comme l'implémentation des objets de son unique super classe à laquelle s'ajoute ses propres propriétés. Cette implémentation est optimale en espace et, de plus, elle bénéficie d'une totale invariance par rapport aux types statiques. La sélection de méthode s'effectue via l'indice invariant du message invoqué.

En héritage multiple, deux solutions sont possibles : soit l'invariance des indices est respectée mais alors deux messages en conflits ne peuvent pas occuper le même indice (cf. coloration de messages) soit l'invariance des indices n'est plus vraie et, dans ce cas, la sélection de méthode ne peut plus être effectuée par une seule indirection. C'est cette dernière solution qui a été retenu pour l'implémentation standard.

La sélection des méthodes est rendu possible en indiquant, à des positions invariantes, les informations nécessaires afin d'effectuer les décalages de position. Ces décalages peuvent être factorisés par super-classe. Tout se passe comme si les objets étaient constitués de sous-tables (ou sous-objets), une par super-classe. Une optimisation permet de traiter l'implémentation d'une partie de la hiérarchie comme en héritage simple ce qui permet de limiter la navigation entre sous-objets. Cependant, ce type d'optimisation n'est rendu possible que grâce à une connaissance complète des super classes d'un objet à la compilation.

2.2.2 Matrice de sélection

Une solution naïve au problème de la sélection de méthodes consiste à construire une matrice qui, pour un type et un message donné, contient la méthode la plus spécifique. La sélection d'une méthode $o.m$ revient alors à appeler la méthode contenue à l'intersection de la ligne o et de la colonne m. Cette méthode garantit une sélection de méthode en temps constant. Malheureusement, la taille de la matrice est beaucoup trop importante (e.g. 500 Mo pour IBM SF) pour qu'elle soit envisageable en pratique.

Néanmoins, du fait de la nature de la matrice, de nombreuses optimisations peuvent être effectuées. Ces optimisations se basent sur deux observations. Tout d'abord, un type donné n'a connaissance que d'un sous-ensemble des messages. Ensuite, un type ne redéfinit qu'une petite partie des méthodes qu'il hérite. Le premier phénomène introduit un grand nombre de trous (i.e. cases vides) tandis que le second produit des informations redondantes dans la matrice.

2.2.3 Table d'association

Cette méthode vise à éliminer les trous de la matrice de sélection en associant pour chaque message la liste ordonnée des types l'implémentant. La taille du codage est donc linéaire en le nombre d'entrées non vides de la matrice. Malheureusement, la sélection n'est plus en temps constant mais au minimum en log de la taille de la liste des messages invocables.

2.2.4 Jalapeño [AAB+99]

Cette technique peut être considérée comme une variante de la technique précédente [2]. En effet, l'idée sous-jacente à cette technique est de décomposer l'ensemble des messages en parties disjointes permettant ainsi de réduire le nombre de listes ordonnées, une par partie. En fixant la taille du partitionnement, cette technique permet un chargement dynamique des classes.

2.2.5 Déplacement de ligne *(Row displacement)* [Dri93, DH95]

Cette méthode consiste à trouver une disposition des lignes de la matrice de sélection dans un tableau qui minimise le nombre de cases vides. Toutefois, dans les langages à typage dynamique, cette méthode n'est pas directement applicables. En

[2] à propos de cette technique, une interprétation abusive de la coloration est faîte par [ZG02]

effet, cette technique introduit des méthodes à des emplacements vides ce qui, dans les langages à typages dynamiques qui ne peuvent pas bénéficier des vérifications du compilateur, ne permet plus de déterminer si le message est connu ou non du type dynamique du receveur. Une solution consiste à effectuer les vérifications à l'exécution.

2.2.6 Table de sélection compacte *(CT Compact Dispatch Table)* [VH96]

Cette méthode est une variante de la technique précédente qui permet de réduire le nombre de doublons. Dans la méthode par déplacement de ligne, les seules réutilisations possibles sont réservées aux lignes superposables (i.e. une classe sans redéfinition). Dans cette méthode, l'astuce consiste à découper la matrice augmentant ainsi la probabilité d'obtenir des lignes identiques dans les sous matrices.

2.2.7 Intervalle contenant [ZG02]

Pour un message m, cette technique consiste à partitionner la hiérarchie de sorte que, pour chaque partie, il existe une numérotation des classes qui permette de représenter par un seul intervalle les classes d'équivalence des classes possédant une même méthode implémentant m. Comme dans la technique PQ-encoding, la numérotation des classes va être obtenu en utilisant des arbres PQ. Dans ce cadre, la sélection de méthode $o.m$ revient à appelé la méthode u associé à la classe d'équivalence contenant le type dynamique de o. Cette méthode permet de réduire le nombre de doublons mais la sélection de méthode n'est plus en temps constant. En effet, la représentation des intervalles est assurée par un tableau associatif dans lequel les clés sont les bornes inférieures des intervalles. Dans ce cadre, l'appartenance d'un élément x à un intervalle consiste à trouver la clé y tel que $y = \max(y \mid y \leq x)$ ce qui peut être réaliser en $O(\log n)$. En typage statique, une optimisation peut être réalisée en démarrant la recherche de la méthode à partir du type statique du receveur.

2.2.8 SmallTalk-80 [BI82]

Bien que le langage soit en typage dynamique, l'accès aux attributs s'effectue comme en typage statique. En effet, l'accès aux attributs s'effectue uniquement sur *self*.

2.3 Coloration

Historiquement, elle a été introduite par [DMSV89] pour la coloration de messages et étendue aux langages à typage dynamique par [AR92], mais par la suite, elle a relativement peu été étudiée. Cela est probablement dû aux temps de calcul prohibitifs qu'elle obtient dans l'article de P. André et J.C. Royer. Pourtant, appliquée aux objets, cette technique a la même efficacité que l'implémentation standard de l'héritage simple en typage statique et les expérimentations menées par [PW90, Duc97, Duc02b] montrent que la taille de codage et les temps de calculs sont relativement faibles.

2.3.1 Principe

Le principe de cette technique est de maintenir les invariants de l'implémentation standard de l'héritage simple en typage statique. Une numérotation injective suffit à satisfaire ces invariants mais on obtient une matrice de taille beaucoup trop importante (la matrice de sélection dans le cas des méthodes). Le problème est donc de trouver une numérotation qui minimise le nombre de positions nécessaires. Ce problème se ramène au problème de la coloration d'un graphe quelconque, bien connu pour être très difficile à traiter (NP-Difficile et non approximable). De plus, le critère de minimisation (i.e. minimisation du nombre de couleurs) donne des résultats très médiocres en termes d'espace. Examinons comment cette technique peut être effectivement appliquée à l'implémentation des objets.

Coloration de messages et d'attributs

La coloration de messages permet de maintenir l'invariance des positions des messages vis-à-vis du type statique ce qui permet d'avoir une sélection de méthodes en temps constant. Dans ce cadre, deux messages ne peuvent pas occuper la même couleur si ils sont connus d'une même classe. La coloration de messages peut être transposée à la coloration des attributs mais l'existence de trous dans les tables d'attributs est beaucoup plus gênante. En effet, le nombre de trous dans les tables des méthodes est relativement maîtrisable puisqu'il dépend de la hiérarchie de classes (informations disponible à la compilation) et les classes factorisent les tables des méthodes des objets. Par contre, dans le cas des attributs, cette factorisation n'est plus possible, les trous se trouvent donc multiplié par les instances dont le nombre n'est connu qu'à l'exécution.

Coloration de classes

La coloration a encore une troisième cible : les classes, pour l'implémentation du test de sous-typage. En effet, appliquée aux classes, cette technique est équivalente à la technique introduite par [Coh91] et généralisé par [VHK97]. Elle consiste à colorer les classes telle que deux classes de même couleur n'aient pas de sous-classe commune, exactement comme si les classes étaient des méthodes. Le test de sous-typage $a \leq b$ consiste à vérifier que l'identifiant de b apparaisse à la position $\chi(b)$ du vecteur T_a. La coloration de classes est exactement le problème de la coloration du graphe constitué par l'union du graphe de comparabilité et d'un graphe de conflit défini comme le graphe d'incomparabilité restreint aux classes ayant une sous-classe commune. [Duc01a] présente et expérimente plusieurs heuristiques.

2.3.2 Améliorations

Deux améliorations permettent de réduire significativement la taille des tables obtenues. D'abord, une représentation par vecteurs de la matrice offre un meilleur taux de compression : le taux de trous étant d'environ 6 % pour la coloration des méthodes de SMALLTALK à comparer au 22 % de [AR92]. Le critére à minimiser devient donc la somme des tailles des vecteurs. Ensuite, la coloration bidirectionnelle, introduite [PW90] et adoptée dans d'autres approches [Mey97, EG00], qui consiste à utiliser aussi bien les indices négatifs que les indices positifs des vecteurs permet de réduire le nombre de trous sans pénaliser l'efficacité des accès. Une généralisation à n dimensions a été proposée par [PW93], mais elle ne semble pas présenter d'intérêt pour l'implémentation des objets à cause des indirections supplémentaires qu'elle impose.

Chapitre 3

Définitions et résultats de complexité

Dans un premier temps, nous traiterons le cas de la coloration de classes que nous généralisons à la coloration des éléments d'un ordre. Cette généralisation a pour objectif de formaliser l'approche de la coloration et de montrer que cette technique ne s'applique pas qu'aux langages à objets.

3.1 Définitions

Définition 1 (Ensemble ordonné)
Un ensemble ordonné ou ordre $P = (X, \leq_P)$ est une relation binaire réflexive, antisymétrique et transitive $\leq_P$ sur un ensemble X.

Un sous-ordre de P est un ordre $P' = (X', \leq_P \cap X'^2)$ tel que $X' \subset X$.

La relation binaire antisymétrique, transitive mais non réflexive associée à $\leq_P$ est $<_P$. Deux éléments x et y sont *comparables* si x $\leq_P$ y ou y $\leq_P$ x. Dans le cas contraire, ils sont *incomparables*, ce que l'on note par x $\|_P$ y.

Enfin, tout ordre strict $P = (X, <_P)$ peut être vu comme le graphe orienté sans circuit G_{tc} = (X,E) avec E = (x,y) tel que $y <_P x$. Ce graphe admet une unique réduction transitive G_{tr} appelé graphe orienté de couverture.

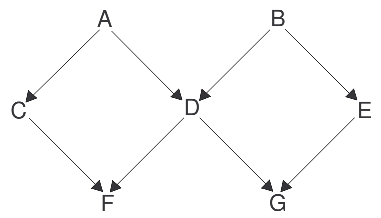


FIG. 3.1 – Graphe orienté de couverture d'un ordre.

Soit un élément $x \in X$, nous définissons les ensembles suivants :

$$descendants(x) = \{\, y \in X \mid y <_P x\}$$
$$ancetres(x) = \{\, y \in X \mid x <_P y\}$$
$$successeurs(x) = \{\, y \in X \mid y < x \wedge \nexists z : y < z \wedge z < x\}$$
$$predecesseurs(x) = \{\, y \in X \mid x < y \wedge \nexists z : x < z \wedge z < y\}$$

Définition 2 (Matrice d'adjacence)
Soit un ensemble ordonné $P = (X, \leq)$ où $X = \{x_1, \ldots, x_n\}$, une matrice d'adjacence M(P) est une matrice a_{ij} de dimension $(n \times n)$ définie par :
- $a_{ij} = 1$ *si* $x_j \leq_P x_i$
- $a_{ij} = 0$ *sinon*

	A	B	C	D	E	F	G
A	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0
C	1	0	1	0	0	0	0
D	1	1	0	1	0	0	0
E	0	1	0	0	1	0	0
F	1	1	1	1	0	1	0
G	1	1	0	1	1	0	1

FIG. 3.2 – La matrice d'adjacence du graphe de la figure 3.1.

La représentation d'un ordre par sa matrice d'adjacence permet un test de comparabilité en $O(1)$. En effet, le test de comparabilité nécessite l'exécution d'une seule instruction, malheureusement la taille d'une telle matrice est considérable. Toutefois, la coloration des éléments de l'ordre permet de réduire la taille de cette matrice tout en conservant l'efficacité du test de comparabilité.

Définition 3 *Le cœur d'un ordre $P = (X, \leq_P)$ est le sous-graphe $Core(H) = (X', E')$, constitué par sa restriction aux éléments qui possèdent un descendant (i.e. une sous-classe) avec au moins deux prédécesseurs (i.e super-classe directe).*

Définition 4 *La couronne de l'ordre $P = (X, \leq_P)$ est le sous-graphe $Crown(H) = (X", E")$, constitué par sa restriction aux éléments dont les descendants ont au plus un prédécesseurs (i.e. une seule super-classe directe).*

Définition 5 *La frontière est constituée par les éléments minimaux du cœur.*

Enfin, nous rappelons le concept de réduction polynomiale entre problème de décision qui nous servira à determiner la complexité de la coloration et de ses variantes.

La réduction polynomiale ou Karp-réduction est notée $L_1 \ll_K L_2$ et signifie que le probléme L_1 est plus facile que le problème L_2 ou encore que le problème L_2

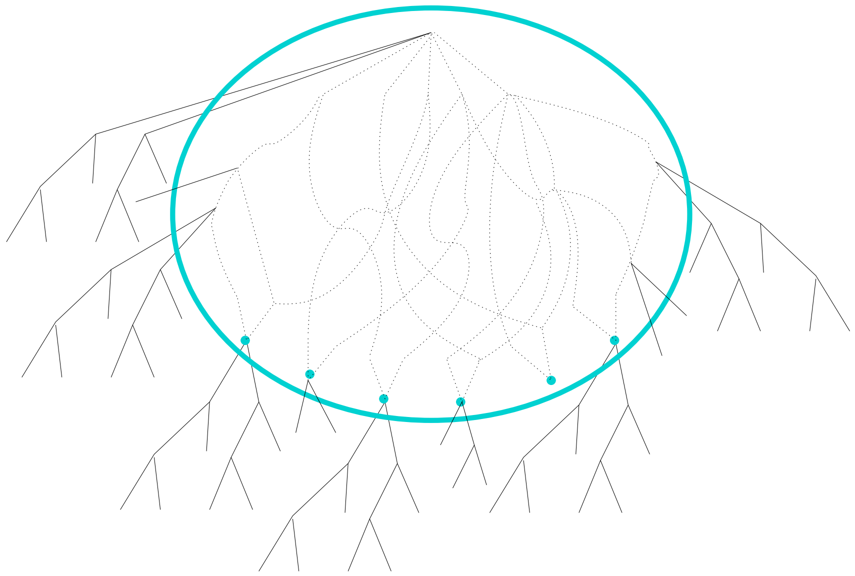


FIG. 3.3 – Le cœur d'un ordre, dans l'ellipse, la frontière et la couronne (d'après Ducournau).

permet de résoudre le problème L_1. Cette réduction n'est définie qu'entre problèmes de décision.

On écrit que $L_1 \ll_K L_2$, s'il existe une transformation polynomiale A qui transforme toute instance I de L_1 en une instance $A(I)$ de L_2 telle que la réponse est "oui" pour I pour L_1 si et seulement si la réponse est "oui" pour $A(I)$ pour L_2.

3.2 Coloration d'une hiérarchie

La coloration d'une hiérarchie, appelée aussi coloration unidirectionnelle ou plus simplement coloration, consiste à conserver l'invariance des positions des éléments de l'ordre. Dans ce cadre, plusieurs éléments peuvent occuper la même position si et seulement s ils n'ont pas de sous-classes communes.

La définition suivante énonce rigoureusement la condition de conflit entre deux éléments de X.

Définition 6 (Graphe de conflits, Graphe de dépendance)
Soit $P = (X, \leq_P)$, le graphe de conflits G(P)=(X,E) est un graphe non orienté tel que :

$(x, y) \in E \Leftrightarrow x$ *et* y *sont incomparables et* $\exists z \in X$ *tel que* $z \leq_P x$ *et* $z \leq_P y$.

Le graphe de dépendance de P, noté D(P), est l'union du graphe non orienté de couverture de P et du graphe de conflits G(P).

Nous définissons la coloration d'une hiérarchie comme la coloration du graphe de dépendance.

Définition 7 (Coloration d'une hiérarchie)
Soit un ensemble ordonné $P = (X, \leq_P)$, une coloration $\chi : X \rightarrow \mathbb{N}$ de P est une fonction telle que $(x, y) \in E_{D(P)} \Rightarrow \chi(x) \neq \chi(y)$.

La figure suivante illustre un graphe de dépendance muni d'une coloration de ses sommets. Les arêtes en traits continus sont les arêtes du graphe non orienté de couverture de l'ordre tandis que les arêtes en pointillés sont les arêtes du graphe de conflits associé à l'ordre.

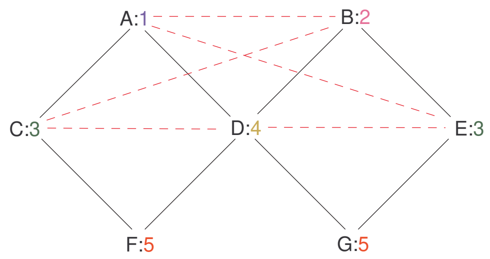


FIG. 3.4 – Un graphe de dépendance muni d'une coloration

Définition 8 (Matrice colorée)
Soit un ensemble ordonné $P = (X, \leq_P)$ où $X = \{x_1, \ldots, x_n\}$ et une coloration χ, une matrice colorée, notée $MC(D(P), \chi)$, est une matrice a_{ij} de dimension $(|X| \times k)$ telle que :

- $a_{ij} = x$ *si* $x \leq_P x_i$ *et* $\chi(x) = j$
- $a_{ij} = \perp$ *sinon*

	A	B	C,E	D	F,G
A	A				
B	$\perp$	B			
C	A	$\perp$	C		
D	A	B	$\perp$	D	
E	$\perp$	B	E		
F	A	B	C	D	F
G	A	B	E	D	G

FIG. 3.5 – Matrice colorée associée à la coloration du graphe de la figure 3.4. Les éléments $\perp$ représentent les trous de la coloration.

Problème 1 *Nombre chromatique d'une hiérarchie*
Donnée : *Un ensemble ordonné $P = (X, \leq)$*

Résultat : *Trouver une coloration χ qui minimise le nombre de couleurs*

C'est le problème classique de coloration de graphe appliqué au graphe de dépendance.

On peut montrer aisément que le calcul du nombre chromatique d'une hiérarchie est aussi difficile que le calcul du nombre chromatique d'un graphe quelconque. Pour s'en convaincre, il suffit de remarquer que le graphe de conflits n'appartient à aucune classe particulière. En effet, à partir d'un graphe quelconque G, on peut construire un ensemble ordonné P tel que G(P) = G en associant à chaque sommet de G un élément dans P et à chaque arête un élément z telle que $z \leq x$ et $z \leq y$ ce qui induit l'arête (x, y) dans $G(P)$.

Toutefois, la minimisation du nombre de couleurs ne tient pas compte de la représentation en mémoire de la matrice colorée. En effet, une représentation par vecteurs est plus efficace. Pour une matrice de dimension $n \times n$, elle nécessite n vecteurs dont la taille totale est majorée par $n(n+1)/2$ au lieu de n^2. Un critère de minimisation plus pertinent consiste donc à minimiser la somme des tailles des vecteurs.

Problème 2 *Coloration d'une hiérarchie minimale*
Donnée : *Un ensemble ordonné $P = (X, \leq_P)$*

Résultat : *Trouver une coloration χ qui minimise $\sum_{x \in X} \max_{x \leq_P y} \chi(y)$.*

Définition 9 (Coloration d'une hiérarchie parfaite)
Soit $P = (X, \leq_P)$, une coloration χ est une coloration unidirectionnelle parfaite si $\forall x \in X$ tel que $\chi(x) > 1, \exists y \in X$ tel que $x \leq_P y$ et $\chi(y) = \chi(x) - 1$.

Proposition 1 *Soit $P = (X, \leq_P)$, les trois propositions suivantes sont équivalentes :*

1. *il existe une coloration d'une hiérarchie parfaite de P.*
2. *le graphe de conflits G(P) est vide.*

3. le graphe orienté de couverte de P est une arborescence.

Preuve :

$1 \Rightarrow 2$: Raisonnons par l'absurde et supposons qu'il existe une coloration parfaite de P et que le graphe de conflits de G(P) possède au moins une arête e entre x et y.

Il existe donc z tel que $x <_P z$ et $y <_P z$. Sans perte de généralité, supposons que $\chi(x) < \chi(y)$, comme le vecteur associé à y est sans trou, il existe w tel que $\chi(w) = \chi(x)$ et $x \neq w$ (x et y sont incomparables) ce qui est absurde puisque x et w sont aussi en conflit.

$2 \Rightarrow 3$: immédiat.

$3 \Rightarrow 1$: On définit une fonction χ de P par :

- $\chi(r) = 0$ si r est la racine de l'arborescence.
- $\chi(x) = \chi(y) + 1$ où y est un successeur de x sinon.

Tout d'abord, il est clair que la fonction χ est une coloration de P. Soit x un élément quelconque de E, il existe un unique chemin de r vers x. Par construction, les couleurs des éléments constituant ce chemin forment une suite continue croissante qui peut être trivialement représentée par un vecteur sans trou.

□

Dans le cas où une coloration χ n'est pas parfaite, on dit qu'elle contient des trous.

En langage mathématique, il y a un trou à la position (i,j) ssi $\nexists z$ tel que $x_i \leq z$ et $\chi(z) = j$ avec $j < \max\limits_{x_i \leq_P y} \chi(y)$.

Proposition 2 *Soit $P = (X, \leq_P)$, le nombre de trous engendré par une coloration unidirectionnelle optimale est inférieure ou égale à $|X|^2 - |X|$.*

Preuve :

La coloration unidirectionnelle induite par un codage par matrice d'adjacence de l'ordre P engendre $|X|^2 - ft$ trous et $ft \geq |X|$ où ft est la taille de la fermeture transitive.

□

3.3 Résultats de complexité

Le problème de coloration d'une hiérarchie est clairement un problème d'optimisation dont nous pouvons donner le problème de décision associé :

Problème 3 *K-Coloration d'une hiérarchie*

Donnée : *Un ensemble ordonné $P = (X, \leq)$ et un entier K*

Résultat : *Existe-t-il une coloration χ qui ait au plus K trous ?*

Jusqu'à présent la complexité du problème de la K-Coloration d'une hiérarchie n'était pas connue. Dans cette section, nous établissons la NP-complétude de ce problème en exhibant une réduction polynomiale depuis la variante décisionnelle du problème Max2Sat [GJ79]. Le problème Max2Sat est un problème d'optimisation bien connu en logique et qui consiste, à partir d'une formule constituée de clauses avec au plus deux littéraux, à trouver une interprétation qui permette de satisfaire le maximum de clauses.

Problème 4 *Maximum 2 Satisfiability (Max2Sat)*

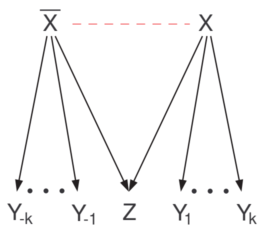
Donnée : *Un ensemble de variables booléenes $x_1, \ldots, x_n$, une collection de clauses $C = \{C_1, \ldots, C_m\}$ où chaque clause possède au plus deux littéraux et un entier l.*

Résultat : *Existe-t-il une affectation de valeurs de vérité qui permet de satisfaire au moins l clauses ?*

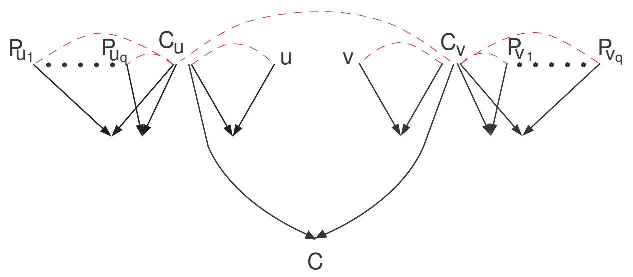
Nous définissons ici la construction d'un ensemble ordonné à partir d'une instance de Max2Sat.

Définition 10 *Soit une instance F de Max2Sat, on associe à F l'ensemble ordonné, noté H(F), suivant :*

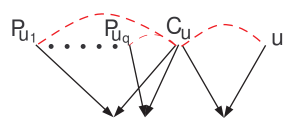
- *Soit $X = \{x_1, \ldots, x_n\}$ l'ensemble des variables apparaissant dans F, on associe à chaque variable x le motif suivant :*



- *Soit $C = \{C_1, \ldots, C_m\}$ l'ensemble des clauses de F, on associe à chaque clause binaire $C = (u \vee v)$ le motif suivant :*



et à chaque clause unaire $C = u$ le motif suivant :



Le lien entre interprétation d'une formule F de Max2Sat et coloration de l'ensemble ordonné H(F) est donné dans la définition suivante :

Définition 11 *Coloration associée à une interprétation*
Soit I une interprétation d'une instance F de Max2Sat, une coloration associée à l'interprétation I est une fonction $\chi : X_{H(F)} \to \mathbb{N}$ telle que, pour toute littéral u (i.e. x ou $\bar{x}$) :

$$\chi(u) = 1 \text{ et } \chi(\bar{u}) = 0 \Leftrightarrow I(u) = faux.$$

Définition 12 *Coloration minimale associée à une interprétation*
Soit I une interprétation d'une instance F de Max2Sat, une coloration minimale associée à l'interprétation I est une coloration associée à l'interprétation I telle que :
- *chaque motif associé à la clause $C = u \vee v$ binaire :*
  - $I(C) = faux \Leftrightarrow \forall i \leq q : \chi(P_{u_i}) = 0$ *et* $\chi(P_{v_i}) = 1$, $\chi(C_u) = 2$, $\chi(C_v) = 0$ *et* $\chi(C) = 1$
  - $I(C) = vrai \Leftrightarrow \forall i \leq q : \chi(P_{u_i}) = \chi(P_{v_i}) = 0$, $\chi(C) = 0$, $\chi(C_u) = 1$ *et* $\chi(C_v) = 2$
- *chaque motif associé à la clause $C = u$ unaire :*
  - $I(C) = faux \Leftrightarrow \chi(C_u) = 0$ *et* $\forall i \leq q : \chi(P_u) = 1$.
  - $I(C) = vrai \Leftrightarrow \chi(C_u) = 1$ *et* $\forall i \leq q : \chi(P_u) = 0$.

Proposition 3 *Soit F une instance de Max2Sat avec n variables et m clauses dont m_1 clauses unaires, une coloration minimale associée à une interprétation I de F qui satisfait l clauses est une coloration de H(F) qui engendre $3l+(q+2)(m-l)+n-2m_1$ trous avec $q > 3$.*

Preuve :

On laisse le soin au lecteur de vérifier que, pour chaque élément de H(F), les couleurs associées à ses ancêtres sont toutes 2 à 2 distinctes. De plus, l'ensemble des motifs associés aux variables engendre n trous, les motifs associés aux clauses binaires (resp. unaires) vérifiées engendrent 3 trous (resp. 1 trous), les motifs associés aux clauses binaires (resp. unaires) falsifiées engendrent $q+2$ trous (resp. q trous).
□

Proposition 4 *Toute coloration minimale χ de $H(F)$ est la coloration minimale associée à une interprétation I_χ de F.*

Preuve :

Tout d'abord, supposons, sans perte de généralité, que le nombre de trous engendré par une coloration minimale χ soit p.

Montrons que χ est une coloration associée à une interprétation.

Raisonnons par l'absurde. Il existe donc un sommet x associé à un littéral tel que $\chi(x) \geq 2$. Le nombre de trous engendrés par le motif associé à la variable x est alors $(\chi(x)-1)(k+1)+(\chi(x)-1)$ trous où k est le nombre de prédécesseurs (i.e. sous-classes) de x. Pour $k > (q+2)m+n \geq p$, ce nombre est supérieur au nombre de trous engendrés par une coloration minimale.

Montrons que χ est une coloration minimale associée à I.

Soit une clause binaire quelconque $C = u \vee v$, supposons que la coloration χ ne soit pas une coloration minimale associée à I et examinons les interprétations possibles des littéraux u et v. Pour simplifier l'énoncé, on peut remarquer que les motifs associés aux clauses binaires sont symétriques. On peut donc se contenter de traiter les trois cas suivant :

- $I(u) = I(v) = vrai$. On a $\chi(u) = \chi(v) = 0$.
  Supposons que $\forall i \leq q : \chi(P_{u_i}) \geq 1$, le nombre de trous engendrés est égale au moins à q. Pour q > 3, le nombre de trous est supérieur à une coloration minimale de $H(F)$. On a donc $\forall i \leq q : \chi(P_{u_i}) = 0$ *et* $\chi(P_{v_i}) = 0$. La coloration minimale des éléments C_u, C_v *et* C se déduit immédiatement et respecte les règles de coloration minimale associée à l'interprétation I.
- $I(u) = faux$ *et* $I(v) = vrai$. On a $\chi(u) = 1$ *et* $\chi(v) = 0$.
  De même que dans le cas précédent, on a $\forall i \leq q : \chi(P_{u_i}) = 0$ *et* $\chi(P_{v_i}) = 0$ et la coloration minimale des autres éléments se déduit immédiatement et verifie la construction de la coloration minimale associée à l'interprétation I.
- $I(u) = I(v) = faux$. On a $\chi(u) = \chi(v) = 1$.
  Supposons que $\forall i \leq q : \chi(P_{u_i}) \geq 2$, le nombre de trous engendrés est égale au moins à $2q$. Pour $q > 3$, le nombre de trous est supérieur à une coloration minimale de $H(F)$. Supposons que $\forall i \leq q : \chi(P_{u_i}) = 0$, on a alors $\chi(C_u) = 2$ et $\chi(C_v) = 3$ puisque $\chi(C_u) \neq \chi(C_v)$. La coloration χ engendre alors $q + 7$ trous ce qui est supérieur au nombre de trous de la coloration minimale associée à l'interprétation I.

Les clauses unaires se traitent de la même manière.
□

Proposition 5 *K-Coloration unidirectionnelle est NP-complet.*
Preuve :
Max2Sat $\ll_K$ K-Coloration unidirectionnelle.

D'après les propositions précédentes, il existe une interprétation d'une instance F de Max2Sat qui permet de satisfaire au moins l clauses ssi la coloration minimale de $H(F)$ engendre au plus $3l + (q + 2)(m - l) + n - 2m_1$ trous.

On a donc prouvé que Max2Sat se réduit à K-Coloration unidirectionnelle. Cette réduction est manifestement polynomiale et K-Coloration unidirectionnelle est dans NP de façon triviale.
□

La NP-complétude du problème de la coloration d'une hiérarchie va nous permette de construire, dans les chapitres suivants, des réductions polynomiales de ce problème vers la coloration bidirectionnelle [PW90] et la coloration des propriétés [DMSV89, Duc01b]. Ces résultats sont inédits à ce jour et généralisent les résultats obtenus par [PW93].

3.4 Coloration n-dimensionnelle

Dans cette partie, la coloration d'une hiérarchie est étendue à une coloration n-dimensionnelle d'une hiérarchie. Cette généralisation, introduite par [PW90, PW93], permet de réduire le nombre de trous. Pour $n = 2$ [PW90], l'efficacité des accès peut être conservé en utilisant une représentation bidirectionnelle (indices négatifs et positifs des vecteurs), on parle alors de coloration bidirectionnelle. Par contre, pour $n > 2$, l'accès aux éléments s'effectue via une indirection supplémentaire ce qui rend cette technique beaucoup moins intéressante pour l'implémentation des objets.

Définition 13 (Coloration n-dimensionnelle)
Soit $P = (X, \leq_P)$, une coloration n-dimensionnelle ψ_n de P est un couple de fonction $dir : X \to \{1, \ldots, n\}$ et $\chi : X \to \mathbb{N}$ telle que :
$(x, y) \in E_{D(P)} \Rightarrow (dir(x) \neq dir(y)) \vee (\chi(x) \neq \chi(y))$.

La relation d'équivalence suivante traduit la décomposition en dimension de l'ordre :

Définition 14 (Relation d'équivalence) *Soit $P = (X, \leq_P)$ et une coloration n-dimensionnelle ψ_n, la relation d'équivalence R est définie par :*
$x \sim y \Leftrightarrow dir(x) = dir(y)$.

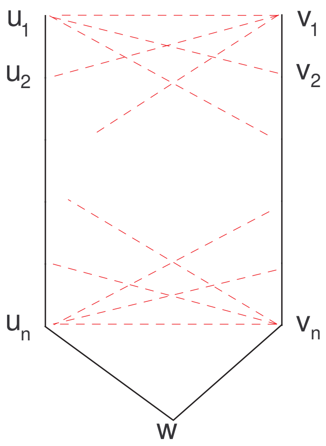
Problème 5 *Coloration n-dimensionnelle minimale*
Donnée : *Un ensemble ordonné $P = (X, \leq_P)$*
Résultat : *Trouver une coloration n-dimensionnelle $\psi_n =< dir, \chi >$ qui minimise* $\sum_{x \in X} \sum_{i=1}^{n} \max_{x \leq_P y \,\wedge\, dir(y)=i} \chi(y)$.

Définition 15 (Coloration n-dimensionnelle parfaite)
Soit $P = (X, \leq_P)$, une coloration n-dimensionnelle ψ_n est dite parfaite si elle ne contient pas de trous.

Proposition 6 *Soit $P = (X, \leq_P)$, il existe une coloration n-dimensionnelle parfaite ssi le graphe de conflits G(P) est partitionnable en n stables.*
Preuve : Le partitionnement en n stables du graphe de conflits induit un partitionnement de P. Chaque partie obtenue admet une coloration parfaite, donc P admet une coloration n-dimensionnelle parfaite.
La réciproque est vraie puisque, s'il existe une coloration n-dimensionnelle parfaite, alors il existe un partitionnement induit par la relation d'équivalence R qui, d'après la proposition 1 et la définition 14, constitue un partitionnement en n stables du graphe de conflits. □

Problème 6 *K-Coloration n-dimensionnelle*
Donnée : *Un ensemble ordonné $P = (X, \leq_P)$ et un entier K*
Résultat : *Existe-t-il une coloration n-dimensionnelle ψ_n qui est au plus K trous ?*

Lemme 1 *Soit un ordre $P = (X, \leq_P)$ appartenant à la classe des ordres définie par le schéma ci-dessous, toute coloration bidirectionnelle de P dont la bipartition induite n'est pas optimale engendre au moins n trous.*



Preuve :

La coloration bidirectionnelle optimale de P induit un partitionnement de X en deux sous ensembles $\{u_1, \ldots, u_n\}$ et $\{v_1, \ldots, v_n\}$.

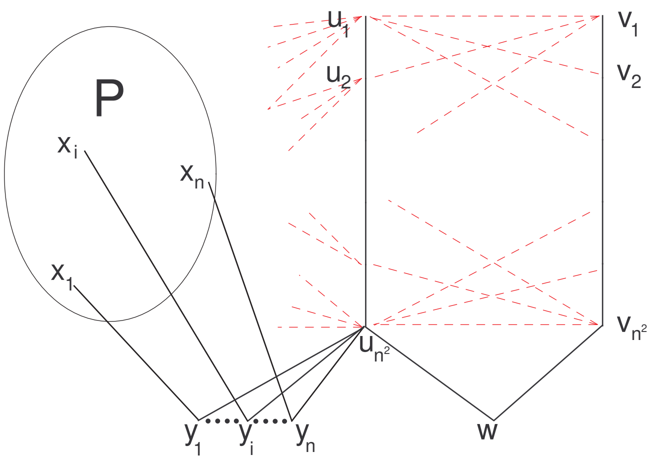
Une bipartitionnement non optimale de P est donc de la forme suivante $\{u_{i_1}, \ldots, u_{i_p}, v_{j_1}, \ldots, v_{j_q}\}$ qui contient pq arêtes de conflits et $\{u_{i_{p+1}}, \ldots, u_{i_n}, v_{j_{q+1}}, \ldots, v_{j_n}\}$ qui contient $(n-p)(n-q)$ arêtes de conflits avec $n > p, q > 0$. On a donc $pq + (n-p)(n-q)$ arêtes de conflits dans les deux sous ordres. Clairement, chaque arête de conflits provoque au minimum un trou (i.e. la couleur occupée par une des extrémités de l'arête ne peut pas être occupée par l'autre) et donc le nombre de trous engendré par une coloration bidirectionnelle non optimale est au minimum $pq + (n-p)(n-q)$. On laisse le soin au lecteur de vérifier que $pq + (n-p)(n-q) \geq n$ avec $n > p, q > 0$.

$\square$

Proposition 7 *K-Coloration bidimensionnelle (ou bidirectionnelle) est NP-complet.*

Preuve : K-Coloration d'une hiérarchie $\ll$ K-Coloration bidimensionnelle.

Soit un ensemble ordonné $P = (X, \leq)$ où $X = \{x_1, \ldots, x_n\}$, on associe à P l'ensemble ordonné suivant :



Tout d'abord, le nombre de trous engendrés par une coloration bidirectionnalle optimale de cet ensemble est inférieur ou égale au nombre de trous engendré par une coloration unidirectionnelle optimale de l'ensemble P. En effet, on pose $dir(X) = dir(V) = +$ et $dir(U) = -$ (i.e. $dir(E) = d \Leftrightarrow \forall e \in E, dir(e) = d$). D'après la proposition 2, la coloration bidirectionnelle optimale respectant cette bipartition engendre donc au maximum $n^2 - n$.

Montrons que, pour toute coloration bidirectionnelle optimale, la bipartition précedente est optimale.

Supposons que $dir(V) \neq dir(U)$ soit faux, alors d'après le lemme 1, le nombre de trous engendrés serait n^2 ce qui est supérieur au nombre de trous engendrés par la coloration bidirectionnelle optimale.
On a donc $dir(V) \neq dir(U)$.

Supposons qu'il existe $x \in X$ tel que $dir(x) = -$ et $\chi(x) = c$, par construction, pour tout $j \leq n^2$: $\chi(x) \neq \chi(u_j)$. Cette coloration engendre alors $(n^2 - c + 1) + c$ trous dans la partie $-$ (i.e. c trous dans le vecteur de x et un trou pour chaque élément de U qui a une couleur supérieure à $\chi(x)$) ce qui est supérieur au nombre de trous engendrés par la coloration bidirectionnelle optimale.

On a donc montré que résoudre le problème de la coloration bidirectionnelle permet de résoudre le problème de la coloration unidirectionnelle et la construction de l'ordre est clairement polynomiale.

□

Proposition 8 *K-Coloration n-dimensionnelle d'une hiérarchie est NP-complet.*

Preuve : Pour n = 1 et n = 2, la proposition a déjà été établie.

Pour $n \geq 3$, il existe une réduction triviale qui est une conséquence immédiate de la proposition 6 [PW93]. □

3.5 Représentation partielle

Dans ce chapitre, nous allons nous intéresser à une classe d'ensembles ordonnés qui ne nécessite qu'une représentation partielle de ses éléments. Cette variante est issue des hiérarchies de classes dans lesquelles certaines classes, appelées classes abstraites, ne possèdent pas d'instances et ne sont pas à colorer. Les principaux résultats de cette variante sont dus à Pugh et Weddell [PW93].

Définition 16 (Matrice colorée partielle)
Soit un ensemble ordonné $P = (X, \leq_P)$ où $X = \{x_1, \ldots, x_n\}$, une coloration χ et $X' = \{x_{l_1}, \ldots, x_{l_m}\} \subset X$, une matrice colorée partielle, notée $MC(P, \chi, X')$, est une matrice a_{ij} de dimension $(|X'| \times k)$ telle que :
– $a_{ij} = x$ *si* $x \leq_P x_{l_i}$ *et* $\chi(x) = j$
– $a_{ij} = \perp$ *sinon*

Proposition 9 *Matrice colorée partielle est NP-complet.*
Preuve : k-coloration $\ll_K$ Matrice colorée partielle
Soit un graphe G=(V,E) et un entier k, on construit l'ensemble ordonné $P = (X, \leq_P)$ suivant :
– à tout sommet x de G est associé l'élément x' de X.
– à tout arête e = (x,y) de G sont associés k-2 éléments $e'_3, \ldots, e'_k$ de X avec $e'_3 \leq_P x'$ et $e'_3 \leq_P y'$; $e'_4 \leq_P e'_3; \ldots ; e'_k \leq_P e'_{k-1}$.

Montrons qu'il existe une coloration χ_k parfaite pour $MC(P, \chi_k, \{e'_k | e \in E\})$ ssi le graphe G est k-colorable.
Si le graphe est k-colorable alors, par construction, il existe une partition en k stables de l'ensemble des x' associés aux sommets de G. Pour chaque arête e, les éléments de l'ensemble $\{e'_3, \ldots, e'_k\}$ peuvent être associés arbitrairement aux k-2 couleurs laissées libres par les éléments x' et y'. Il est clair que cette coloration est une coloration parfaite si on considère la matrice colorée partielle induite par le sous-ensemble $\{e'_k | e \in E\}$ de X.
La réciproque est vraie puisque s'il existe une coloration parfaite de P, alors la coloration associée à la matrice colorée partielle (de dimension $|E| * k$) est une k-coloration de G. $\square$

Proposition 10 *Matrice colorée bidirectionnelle partielle est NP-complet.*
Preuve : k-coloration $\ll_K$ Matrice colorée bidirectionnelle partielle
Soit un graphe G=(V,E) et un entier k, on construit l'ensemble ordonné $P = (X, \leq_P)$ suivant :
– deux éléments *droite* et *gauche.*
– à tout sommet x de G est associé l'élément x' de X.
– à tout arête e = (x,y) de G est associée k éléments $e'_3, \ldots, e'_k, e'd, e'g, e'dg$ de X avec $e'_3 \leq_P x'$ et $e'_3 \leq_P y'$; $e'_4 \leq_P e'_3; \ldots ; e'_k \leq_P e'_{k-1}; e'd \leq_P e'_k, droite; e'g \leq_P e'_k, gauche; e'dg \leq_P ed, eg$.

Montrons qu'il existe une coloration ψ_2 parfaite pour $MC_2(P, \psi_2, \{e'd, e'g, e'dg | e \in E\})$ ssi le graphe G est k-colorable.

Si la coloration est parfaite les éléments *droite* et *gauche* ne peuvent pas avoir la même direction ce qui force les couleurs des autres éléments a être compris entre 1 et k, ce qui constitue une k-coloration de G.
Si le graphe G est k-colorable, il existe une coloration parfaite pour MC_2. (Idem preuve proposition précédente) $\Box$

3.6 Coloration des propriétés

Dans cette partie, les éléments de l'ensemble ordonné sont munis de propriétés. Il nous faut donc définir une méthode afin d'accéder à ces propriétés. La coloration constitue, là aussi, une solution efficace mais qui peut être coûteuse en mémoire.

Définition 17 (fonction prop) *Soit* $P = (X, \leq_P)$ *et un ensemble de propriété* M*, on définit une fonction* $prop_P : X \rightarrow 2^M$ *telle que* $x \neq y \Rightarrow prop_P(x) \cap prop_P(y) = 0$.

La fonction $prop_P$ fait correspondre à un élément x l'ensemble des propriétés $prop_P(x)$ qu'il a introduit.

Définition 18 (Coloration des propriétés) *Soit une fonction* $prop_P$*, une coloration des propriétés* $\chi : M \rightarrow \mathbb{N}$ *de* $prop_P$ *est une fonction telle que* $((x, y) \in E_{D(P)}) \vee (x = y) \Rightarrow \chi(p) \neq \chi(q)$ *où* $p \in prop_P(x)$ *et* $q \in prop_P(y)$.

La définition de la matrice des propriétés colorées découle de la définition précédente ainsi que les problèmes de minimisation et la généralisation à une coloration n-dimensionnelle des propriétés.

Proposition 11 (K-Coloration n-dimensionnelle des propriétés est NP-difficile)
Preuve : Soit $P = (X, \leq_P)$, on définit une fonction $prop_P$ par : $prop_P(x) = \{m_x\}$. Autrement dit, chaque élément dispose d'une et une seule propriété. Dans ce cadre, la coloration n-dimensionnelle d'une hiérarchie est clairement équivalente à la coloration n-dimensionnelle des propriétés.
□

Chapitre 4

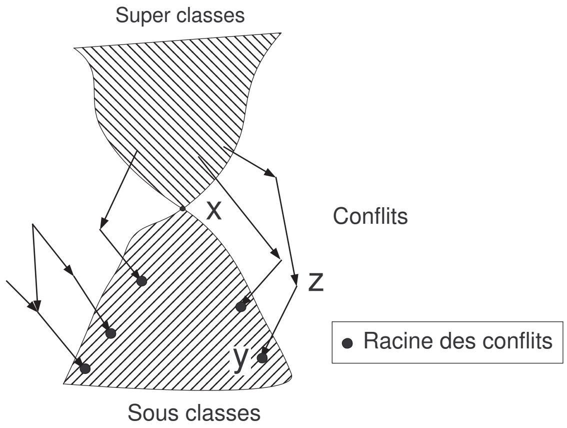
Heuristique

Nous avons montré dans les parties précédentes qu'il n'existe pas d'algorithme polynomial permettant de calculer une coloration n-dimensionnelle d'une hiérarchie. Pourtant, de nombreuses heuristiques ont été proposées ces dernières années [Duc01a, VHK97, PW90]. Nous proposons dans cette partie nos propres heuristiques de coloration qui adoptent une approche différente. Dans ce cadre, nous proposons un algorithme de calcul du graphe de conflits efficace. Nous avons pu programmer nos algorithmes et les exécuter sur la même série de données que nos prédécesseurs, les résultats sont satisfaisants et améliorent ceux obtenus par Ducournau qui, eux-mêmes, amélioraient ceux de Vitek [Duc01a]. Les résultats sont disponibles dans l'annexe de ce rapport.

4.1 Algorithme de calcul du graphe de conflits

Dans cette section, nous proposons un algorithme pour calculer le graphe de conflits d'un ensemble ordonné en $O(n(n+m))$.

Définition 19 (Racines des conflits) *Soit un élément x de l'ensemble ordonné $P = (X, \leq)$, on appelle racines des conflits de x l'ensemble défini par $Racine(x) = \{y \in descendants(x) \text{ tel que } \exists z, z \in predecesseur(y) \text{ et } z \notin descendant(x) \text{ et } z \neq x\}$*



Proposition 12 *Soit un sommet x, le calcul des racines des conflits de x s'effectue en O(n+m).*

Preuve : L'algorithme de calcul des racines des conflits de x consiste à simuler la suppression de l'ensemble des descendants de x. Trivialement, les descendants possédant au moins un successeur après cette suppression sont les racines des conflits de x. La simulation de la suppression est rendue possible grâce à un vecteur d'entiers s qui, à un élément, fait correspondre le nombre de ses successeurs dans le sous-graphe induit par les suppressions.

```
Données : P = (X, ≤) et un élément x ∈ X
Résultat : Racine(x)
pour chaque y ∈ descendants(x) faire
| s(y) ← |successeurs(y)|
fin
pour chaque y ∈ descendant(x) faire
| pour chaque z ∈ predecesseurs(y) faire
| | s(z) ← s(z) − 1
| fin
fin
pour chaque y ∈ descendants(x) faire
| si s(y) > 0 alors
| | Racine(x) ← Racine(x) ∪ {y}
| fin
fin
```

Algorithme 1: Calcul des racines des conflits de x

□

Soit un sommet x, on note conflits(x) l'ensemble des éléments en conflits avec x.

Proposition 13 *Soit x et y deux sommets tels que $x \parallel y$, $y \in conflits(x)$ ssi $\exists z \in racine(x)$ tel que $z \leq y$.*

Preuve : Par définition, il existe w tel que $w < x$ et $w < y$, de plus, on a x $\parallel$ y. Soit $w = x_1, \ldots, x_k = y$ le chemin menant de w à y, il existe un élément x_i tel que $x_i \not\leq x$ et $x_{i-1} \leq x$. x_{i-1} appartient donc à $Racine(x)$.

La réciproque est trivialement vraie.

□

Proposition 14 *Le calcul du graphe de conflits s'effectue en O(n (n + m)).*

Preuve :

```
Données : P = (X, ≤)
Résultat : G(P) = (X, E)
Initialiser le graphe G(P) par le graphe vide;
pour chaque x ∈ X faire
    pour chaque y ∈ descendants(x)∪ancetres(x) faire
        comparable[y] := id_x ;
    fin
    calculer Racine(x);
    pour chaque y ∈ancetres(Racine(x)) faire
        si comparable[y] ≠ id_x alors
            E = E ∪ {(x, y)}

        fin
    fin
fin
```

Algorithme 2: Calcul du graphe de conflits

□

Optimisation

D'après la proposition 1, le graphe de conflits induit par un ordre dont le graphe orienté de couverture est une arborescence est vide. Autrement dit, seuls les éléments du cœur sont susceptibles d'être en conflit. On peut donc restreindre la recherche des conflits aux éléments constituant le cœur de l'ordre.

4.2 Coloration

4.2.1 Coloration unidirectionnelle

Comme nous l'avons vu dans la section 3.2, la coloration unidirectionnelle est un problème NP-Difficile et il est aussi fort probable qu'il ne soit pas non plus approximable en temps polynomial. Toutefois, il ressort des expérimentations menées d'une part par Pugh et Weddell pour la coloration des propriétés et d'autre part par Ducournau pour la coloration de classes que le taux de trous est relativement faible ($\sim$ 15 %) et dans le pire des cas inférieur à 50 %. Ces expérimentations sont issues d'heuristiques privilégiant une approche locale. L'heuristique présentée dans cette section est issue d'une heuristique gloutonne qui consiste à construire un partitionnement en stables pondérés du graphe de dépendance. En effet, si l'on considère la matrice colorée, un principe simple pour réduire le nombre de trous, est de maximiser le nombre de cases non vides des premières colonnes. Plus formellement, l'idée est de rechercher pour chaque colonne de la matrice colorée un stable de poids maximal. Les colonnes seront ensuite ordonnées de manière décroissante suivant le poids du stable associé. Le problème du stable max pondéré est un problème NP-difficile et non approximable mais, en pratique il se révèle plus simple de traiter ce problème plutôt que d'attaquer de front le problème de la coloration. L'heuristique est définie par :

Données : $P = (X, \leq)$
Résultat : Coloration χ
Calculer le graphe de dépendance D(P) ;
Attribuer à chaque élément x un poids w ;
Ordonner l'ensemble X en une séquence $x_1, \ldots, x_n$ décroissante ;
S $\leftarrow \mathbb{P}^0$; *// S est une séquence de stables triée par poids décroissants*
l $\leftarrow$ 1 ; *// indice du premier stable*
pour $j = 1, \ldots, |X|$ **faire**
  pour $k = 1, \ldots, l$ **faire**
    si $S[k] \cup \{x_i\}$ *est un stable* **alors**
      $S[k] \leftarrow S[k] \cup \{x_i\}$;
      Repositionner$(S, S[k])$;
      exit
    fin
  fin
  sinon
    l $\leftarrow$ l + 1 ;
    $S[l] \leftarrow \{x_i\}$; *Repositionner*$(S, S[l])$;
  fin
fin
// Affecter une couleur à chaque stable $S[k]$
pour $k = 1, \ldots, l$ **faire**
  pour chaque $x \in S[k]$ **faire**
    $\chi(x) \leftarrow k$;
  fin
fin

Algorithme 3: Heuristique de coloration unidirectionnelle.

Cette heuristique peut être implémentée en $O(n^2)$. Dans le cadre de notre heuristique, le poids w associé à l'élément x est logiquement le nombre de descendants (i.e. de sous-classes) de x, c'est-à-dire le nombre de cases occupées par l'identifiant x dans la colonne $\chi(x)$.

Proposition 15 *Soit $P = (X, \leq_P)$, si le graphe de conflits G(P) est vide alors la coloration obtenue par l'heuristique de coloration unidirectionnelle est parfaite.*

Preuve : Les poids affectés aux sommets induisent un parcours en largeur de la hiérarchie A chaque étape, l'heuristique insère le sommet dans la première partie autorisée qui, dans le cas d'une arborescence et d'un parcours en largeur, est la partie composée des éléments du même niveau. L'heuristique de coloration unidirectionnelle induit une décomposition en niveaux de l'arborescence où chaque niveau occupe la première couleur laissée libre par les niveaux supérieurs ce qui est clairement optimale. □

Optimisation (Correction de partition)

Une optimisation possible consiste à effectuer une correction de la partition obtenue par l'heuristique gloutonne. En effet, à chaque étape, l'heuristique gloutonne sélectionne l'élément de poids maximal. Malheureusement, ce choix n'est pas toujours optimal.

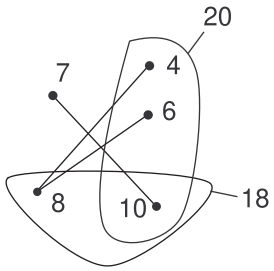


FIG. 4.1 – Une configuration non optimale.

Pour diminuer les effets de ces erreurs, une optimisation possible consiste, pour un stable donné, à rechercher dans les stables de poids inférieurs des configurations non optimales. L'idée est alors d'effectuer des permutations des sommets participants à de telles configurations. Dans le cadre de notre heuristique, la recherche de ces configurations est limitée aux composantes connexes des graphes induits par chaque paire de stables.

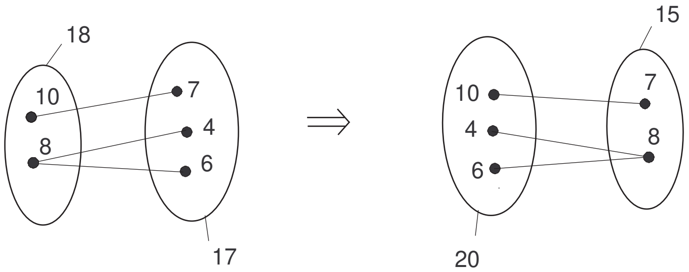


FIG. 4.2 – Permutation des sommets d'une composante connexe non optimale.

Optimisation (Coloration restreinte au cœur)

Une optimisation, visant à réduire les temps de calcul, consiste à colorer la hiérarchie en deux temps. En effet, à partir d'une coloration des éléments du cœur, la coloration des éléments constituant la couronne et la frontière est immédiate. En effet, comme ces éléments ne sont pas en conflit, l'algorithme de coloration consiste simplement à affecter à ces éléments les couleurs laissées libres par leurs ancêtres, en privilégiant, bien entendu, les trous de la coloration. Cet algorithme est simple et efficace puisqu'il consiste uniquement à mettre à jour les couleurs libres lors de la descente de la hiérarchie à partir des éléments du cœur.

4.2.2 Coloration n-dimensionnelle

Une coloration n-dimensionnelle peut être vue comme n colorations unidirectionnelles des n sous-graphes de dépendance induits par la relation d'équivalence R. Le problème consiste alors à trouver la relation d'équivalence R qui minimise la somme des trous des n colorations unidirectionnelles. Intuitivement, le partitionnement du graphe de dépendance élimine les conflits situés entre des sommets appartenant à des parties différentes. L'idée est donc de trouver le partitionnement qui maximise le nombre de conflits éliminés. Ce problème, connu sous le nom de MAX K-CUT, est un problème classique de la théorie des graphes. Il appartient à la classe des problèmes NP-Complet et est .878-approximable [GW94]. Toutefois, l'heuristique issue de ce résultat d'approximabilité, bien que polynomial, est relativement inefficace sur des ordres comptant plusieurs milliers d'éléments.

L'heuristique présentée dans cette section est issue de la famille des metaheuristiques GRASP (Greedy Randomized Adaptive Search Procedure) [FPRR]. Une heuristique GRASP opère en deux étapes : une phase de construction et une phase de recherche locale. La phase de construction attribue aléatoirement une partition à chaque élément. A partir de cette partition, la procédure de recherche locale tente de l'améliorer en produisant pour chaque sommet une solution optimale locale. La phase de construction et de recherche locale sont répétées un certain nombre de fois et la meilleure solution est conservée.

Données : $D(P) = (X, F)$
Résultat : $U^* = S, \bar{S}$
pour $i = 1, \ldots, MAX\_ITER$ **faire**
  Construire une solution aléatoire U
  $U \leftarrow$Recherche locale(U)
  si $i = 1$ **alors** $U^* \leftarrow U$ **else**
    // $w(U)$ = *somme des poids des arêtes coupées par la solution* U
    si $w(U) > w(U^*)$ **alors** $U^* \leftarrow U$
  end
fin

Algorithme 4: Metaheuristique GRASP

```
change ← VRAI
tant que change faire
    change ← FAUX
    pour v = 1, ..., |X| faire
        // σ_S(x_v) = poids des arêtes incidentes à x_v et appartenant à S
        si x_v ∈ S and δ(x_v) = σ_S(x_v) − σ_S̄(x_v) > 0 alors
            S ← S − {x_v} ; S̄ ← S̄ ∪ {x_v} ; change ← VRAI ;
        fin
        si x_v ∈ S̄ and δ(x_v) = σ_S̄(x_v) − σ_S(x_v) > 0 alors
            S̄ ← S̄ − {x_v} ; S ← S ∪ {x_v} ; change ← VRAI ;
        fin
    fin
fin
```

Procédure: `Recherche locale(`$U = S, \bar{S}$`)`

Dans le cadre de notre heuristique, chaque arête $e = (x, y)$ est valuée par un poids qui est le minimum du nombre de sous-classes de x et de y. Intuitivement, cette valuation pénalise les éléments possédant le moins de sous-classes ce qui permet de limiter la propagation des trous. Dans ce cas, le critère d'optimisation de la bipartition est la maximisation de la somme des poids des arêtes coupées.

Remarque

Le partitionnement réalisé par l'heuristique ne traite par les sommets qui n'ont pas de conflits. L'idée est alors d'effectuer une coloration en deux temps, comme dans l'optimisation de coloration restreinte au cœur, qui dans ce cas sera restreinte aux éléments en conflit.

4.2.3 Coloration des propriétés

Comme nous l'avons vu dans la section 3.6, la coloration des propriétés est une généralisation de la coloration de classes. En effet, la coloration des n propriétés d'une classe peut être vue comme la coloration de n classes.

L'heuristique de coloration unidirectionnelle et bidirectionnelle de classes s'adapte facilement à la coloration des propriétés. Cependant, la correction des partitions engendre un surcoût non négligeable du fait du nombre parfois élevé de parties. Ce surcoût est moindre dans la coloration bidirectionnelle puisque le nombre de parties pour chaque direction est plus petit.

Remarque

La bipartition effectuée dans la coloration bidirectionnelle des propriétés est réalisée suivant l'ensemble des classes. Autrement dit, les méthodes d'une même classe appartiennent à la même partie. Un bipartitionnement des propriétés serait plus judicieux mais il nécessiterait un critère de partitionnement qui traduise qu'il est préférable que deux propriétés d'une même classe appartiennent à des parties différentes.

Chapitre 5

Conclusion et perspectives

Dans ce rapport, nous avons étudié et formalisé la coloration d'une hiérarchie et ses variantes. Cette technique permet de traiter les principaux aspects des langages à objet à typage statique de manière simple et efficace. Il s'agit donc d'une technique assez générale dont les applications dépassent probablement les frontières des objets.

Les apports de ce travail se situent à plusieurs niveaux. Tout d'abord, la formalisation de cette technique a été une étape cruciale dans la bonne comprehention du problème. Ce travail avait été, en partie, réalisé par [Duc02a]. Ensuite, nous proposons une réduction polynomiale du problème Max2Sat vers le problème de la coloration d'une hiérarchie. Ce résultat de complexité nous permet ainsi de déterminer la complexité des différentes variantes (coloration bidirectionnelle, coloration des propriétés). Enfin, nous proposons plusieurs heuristiques basées sur une heuristique de coloration unidirectionnelle originale. Ces heuristiques améliorent les performances de nos prédécesseurs dans ce domaine.

De futures recherches sont envisageables et devraient montrer que la coloration d'une hiérarchie est non approximable. Toutefois, de nouvelles heuristiques de recherche d'un stable pondéré maximum qui exploitent les particularités du graphe de dépendance permettraient certainement de réduire le nombre de trous. Par contre, le cas de la bidirectionnalité est plus délicat. En effet, le critère de bipartition utilisé ne tient pas compte de la disposition des arêtes non coupées. Autrement dit, deux bipartitions différentes obtenues par l'heuristique de partitonnement peuvent avoir la même valeur (i.e. la somme des poids des arêtes coupées) sans que les colorations bidirectionnelles associées engendrent le même nombre de trous.

Bibliographie

[AAB+99] B. Alpern, C.R. Attanasio, J.J. Barton, A. Cocchi, S.F. Hummel, D. Lieber, T. Ngo, M. Mergen, J.C. Shepherd, and S. Smith. Implementing Jalapeño in Java. In *Proc. OOPSLA'99*, SIGPLAN Notices, 34(10), pages 314–324. ACM Press, 1999.

[ABJ89] R. Agrawal, A. Borgida, and H.V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the ACM/SIGMOD International Conference on the Management of Data, Portland (OR), USA*, ACM SIGMOD Record, 18(2), pages 253–262, 1989.

[AR92] P. André and J.-C. Royer. Optimizing method search with lookup caches and incremental coloring. In *Proc. OOPSLA'92*, SIGPLAN Notices, 27(10), pages 110–126, Vancouver, 1992. ACM Press.

[BI82] A.H. Borning and D.H.H. Ingalls. Multiple inheritance in Smalltalk-80. In *Proceedings of AAAI'82, Pittsburgh (PA), USA*, pages 234–237, 1982.

[Cas93] Y. Caseau. Efficient handling of multiple inheritance hierarchies. In *Proc. OOPSLA'93*, SIGPLAN Notices, 28(10), pages 271–287. ACM Press, 1993.

[Coh91] N.H. Cohen. Type-extension type tests can be performed in constant time. *Programming languages and systems*, 13(4) :626–629, 1991.

[DH95] K. Driesen and U. Hölzle. Minimizing row displacement dispatch tables. In *Proc. OOPSLA'95*, SIGPLAN Notices, 30(10), pages 141–155. ACM Press, 1995.

[DMSV89] R. Dixon, T. McKee, P. Schweitzer, and M. Vaughan. A fast method dispatcher for compiled languages with multiple inheritance. In *Proc. OOPSLA'89*, New Orleans, 1989. ACM Press.

[Dri93] K. Driesen. Selector table indexing and sparse arrays. In *Proc. OOPSLA'93*. ACM Press, 1993.

[Duc97] R. Ducournau. La compilation de l'envoi de message dans les langages dynamiques. *L'Objet*, 3(3) :241–276, 1997.

[Duc01a] R. Ducournau. La coloration : une technique pour l'implémentation des langages à objets à typage statique. i. la coloration de classes. Technical Report 01-225, L.I.R.M.M., 2001.

[Duc01b] R. Ducournau. Spécialisation et sous-typage : thème et variations. Technical Report 01-013, L.I.R.M.M., 2001. *TSI*.

[Duc02a] R. Ducournau. La coloration pour l'implémentation des langages à objets à typage statique. In M. Dao and M. Huchard, editors, *Actes LMO'2002* in *L'Objet vol. 8*, pages 79–98. Hermès, 2002.

[Duc02b] R. Ducournau. "Real World" as an argument for covariant specialization in programming and modeling. Technical Report 02-083, L.I.R.M.M., 2002.

[EG00] N. Eckel and J. Gil. Empirical study of object-layout and optimization techniques. In E. Bertino, editor, *Proc. ECOOP'2000*, LNCS 1850, pages 394–421. Springer-Verlag, 2000.

[FPRR] P. Festa, P. M. Pardalos, M. G. C. Resende, and C. C. Ribeiro. Randomized heuristics for the max-cut problem.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco (CA), USA, 1979.

[GW94] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and statisfiablity problems using semidefinite programming. *26th Annual ASM Symposium on Theory of Computing, Montréal*, pages 422–431, 1994.

[HN94] M. Habib and L. Nourine. Bit-vector encoding for partially ordered set. *ORDAL'94, LNCS 831*, pages 1–12, 1994.

[Mey97] B. Meyer. *Object-Oriented Software Construction.* The Object-Oriented Series. Prentice-Hall, Englewood Cliffs (NJ), USA, second edition, 1997.

[PW90] W. Pugh and G. Weddell. Two-directional record layout for multiple inheritance. In *Proc. ACM Conf. on Programming Language Design and Implementation (PLDI'90)*, ACM SIGPLAN Notices, 25(6), pages 85–91, 1990.

[PW93] W. Pugh and G. Weddell. On object layout for multiple inheritance. Technical Report CS-93-22, University of Waterloo, 1993.

[SPT83] L.K. Schubert, M.A. Papalaskaris, and J. Taugher. Determining type, part, color and time relationship. *Computer*, 16 :53–60, 1983.

[VH96] J. Vitek and R.N. Horspool. Compact dispatch tables for dynamically typed programming languages. In *Proc. of Intl. Conf. on Compiler Construction*, 1996.

[VHK97] J. Vitek, R.N. Horspool, and A. Krall. Efficient type inclusion tests. In *Proc. OOPSLA'97*, SIGPLAN Notices, 32(10), pages 142–157. ACM Press, 1997.

[ZG01] Y. Zibin and J. Gil. Efficient subtyping tests with PQ-encoding. In *Proc. OOPSLA'01*, SIGPLAN Notices, 36(10), pages 96–107. ACM Press, 2001.

[ZG02] Y. Zibin and J. Gil. Fast algorithm for creating space efficient dispatching tables with application to multi-dispatching. In *Proc. OOPSLA'02*, SIGPLAN Notices, 37(10). ACM Press, 2002.

Annexe A

Résultats : Coloration de classes

Ce chapitre présente les résultats obtenus par nos heuristiques et par les heuristiques de Ducournau [?]. Pour mesurer le caractère aléatoire de nos heuristiques, nous les avons exécutés un grand nombre de fois sur divers benchmarks. Ces benchmarks sont des exemples de hiérarchies prises dans les langages à objets comme C++, Java ou Eiffel. De plus, nous avons mesuré la vitesse de nos heuristiques et avons constaté que les temps d'exécutions étaient relativement faibles. Ces expérimentations ont été réalisées sur un P4 2Ghz.

Système	nb. classe	ferm. trans.	t (ms)	nb trous				pourc.
				coeur	mini	moy	maxi	moy
dylan	925	5097	82	29	87	87	87	1.7
harlequin	666	4493	72	1044	1491	1508	1523	33.6
vortex3	1954	14146	411	1020	1440	1457	1480	10.3
MI-jdk1.3.1	7401	32480	4466	1736	4754	4893	5051	15.1
MI-Orbix	2716	7560	468	183	223	224	228	3.0
MI-Orbacus	1379	6244	197	495	562	578	584	9.3
MI-IBM-XML	145	640	6	46	52	52	52	8.1
MI-IBM-SF	8793	80860	6764	6878	12340	12411	12709	15.3
MI-HotJava	736	3768	81	479	1330	1333	1337	35.4
MI-Corba	1699	6551	263	408	513	521	528	8.0
Java-a	226	883	8	14	15	15	15	1.7
JDK.1.0.2	604	2802	49	79	99	111	126	4.0
Java-1.18	1704	7411	286	263	428	430	433	5.8
Java-1.22	4339	18941	1688	914	2700	2703	2707	14.3
Java-1.30	5438	23772	2929	1195	3622	3624	3627	15.2
IDL	66	253	1	3	3	3	3	1.2
Unidraw	613	1854	51	15	15	15	15	0.8
Lov-obj-ed	436	3707	59	936	1701	1709	1733	46.1
Geode	1318	18442	385	4529	7985	8057	8155	43.7
SmartEiffel	396	3031	29	24	39	44	50	1.5
Eiffel	1999	17559	570	3760	6946	7006	7068	39.9
Laure	293	2392	21	60	84	85	86	3.6
Self	1801	53837	618	1234	1079	1237	1386	2.3
Cecil	932	6032	114	379	563	574	584	9.5
Total	46579	322755	19618	25723	48071	48677	49567	15.1

FIG. A.1 – Statistiques de coloration unidirectionnelle pour notre heuristique.

Système	nb. classe	ferm. trans.	t (ms)	nb trous				pourc.
				coeur	mini	moy	maxi	moy
dylan	925	5097	11	4	4	4	5	0.1
harlequin	666	4493	109	468	594	611	682	13.6
vortex3	1954	14146	193	315	521	544	562	3.8
MI-jdk1.3.1	7401	32480	529	699	1319	1514	1600	4.7
MI-Orbix	2716	7560	46	79	78	79	81	1.0
MI-Orbacus	1379	6244	86	176	155	191	263	3.1
MI-IBM-XML	145	640	4	12	12	12	12	1.9
MI-IBM-SF	8793	80860	3342	1468	2453	2605	2707	3.2
MI-HotJava	736	3768	39	166	410	414	423	11.0
MI-Corba	1699	6551	67	102	116	121	129	1.8
Java-a	226	883	2	0	0	0	0	0.0
JDK.1.0.2	604	2802	9	10	10	10	10	0.4
Java-1.18	1704	7411	40	68	84	87	93	1.2
Java-1.22	4339	18941	206	335	747	816	900	4.3
Java-1.30	5438	23772	316	446	1018	1079	1152	4.5
IDL	66	253	0	0	0	0	0	0.0
Unidraw	613	1854	2	1	1	1	2	0.1
Lov-obj-ed	436	3707	102	487	918	921	925	24.8
Geode	1318	18442	813	2374	4641	4923	5006	26.7
SmartEiffel	396	3031	6	7	12	12	15	0.4
Eiffel	1999	17559	772	1542	2509	2581	2648	14.7
Laure	293	2392	7	14	14	15	19	0.6
Self	1801	53837	167	549	548	549	551	1.0
Cecil	932	6032	45	119	177	180	185	3.0
Total	46579	322755	6913	9441	16341	17269	17970	5.4

FIG. A.2 – Statistiques de coloration bidirectionnelle pour notre heuristique.

Ducournau

nom	rang	nb. trous										fermeture transitive
		optimal		min		moy.		max.		pire		
dylan	99	87	0.02	94	0.02	94.	0.02	94	0.02	2681	0.53	5097
harlequin	43	1319	0.29	1560	0.35	1572.	0.35	1580	0.35	2259	0.50	4493
vortex3	57	1356	0.10	1572	0.11	1665.	0.12	1694	0.12	8901	0.63	14146
MI-jdk1.3.1	27	4328	0.13	4563	0.14	4572.	0.14	4585	0.14	26495	0.82	32480
MI-Orbix	108	182	0.02	403	0.05	405.	0.05	406	0.05	994	0.13	7560
MI-Orbacus	93	442	0.07	807	0.13	817.	0.13	835	0.13	3003	0.48	6244
MI-IBM-XML	173	50	0.08	97	0.15	97.	0.15	97	0.15	136	0.21	640
IBM-SF	113	14905	0.18	22026	0.27	22312.	0.28	22693	0.28	44287	0.55	80860
MI-HotJava	55	1008	0.27	1355	0.36	1359.	0.36	1363	0.36	3982	1.06	3768
MI-Corba	57	531	0.08	704	0.11	712.	0.11	720	0.11	2487	0.38	6551
Java-a	103	12	0.01	14	0.01	14.	0.01	14	0.01	18	0.02	940
JDK.1.0.2	77	96	0.03	142	0.05	142.	0.05	142	0.05	338	0.12	2802
Java-1.18	15	365	0.05	366	0.05	369.	0.05	373	0.05	3163	0.43	7411
Java-1.22	32	2411	0.13	2880	0.15	2888.	0.15	2896	0.15	18420	0.97	18941
Java-1.30	30	3279	0.14	3537	0.15	3559.	0.15	3578	0.15	24095	1.01	23772
IDL	42	3	0.01	3	0.01	3.	0.01	3	0.01	5	0.02	320
Unidraw	86	12	0.00	14	0.01	14.	0.01	14	0.01	18	0.01	2468
Lov-obj-ed	35	1562	0.42	1818	0.49	1829.	0.49	1839	0.50	3594	0.97	3707
Geode	38	8415	0.46	9530	0.52	9555.	0.52	9620	0.52	19880	1.08	18442
SmartEiffel	33	36	0.01	36	0.01	36.	0.01	37	0.01	321	0.09	3428
Eiffel	60	5465	0.31	7668	0.44	7682.	0.44	7700	0.44	16411	0.93	17559
Laure	119	74	0.03	100	0.04	100.	0.04	100	0.04	560	0.23	2392
Self	77	778	0.01	934	0.02	986.	0.02	1021	0.02	4300	0.08	55639
Cecil	78	519	0.09	560	0.09	566.	0.09	572	0.09	2430	0.40	6032
Total	42	47235	0.15	60783	0.19	61348.	0.19	61976	0.19	188778	0.58	325692

FIG. A.3 – Statistiques de coloration undirectionnelle pour l'heuristique de Ducournau.

Ducournau

nom	rang	nb. trous optimal		min		moy.		max.		pire		fermeture transitive
dylan	1	4	0.00	4	0.00	4.	0.00	4	0.00	899	0.18	5097
harlequin	22	441	0.10	465	0.10	606.	0.13	705	0.16	1576	0.35	4493
vortex3	59	363	0.03	511	0.04	537.	0.04	566	0.04	3822	0.27	14146
MI-jdk1.3.1	26	955	0.03	1303	0.04	1354.	0.04	1414	0.04	14451	0.44	32480
MI-Orbix	2	16	0.00	17	0.00	22.	0.00	32	0.00	564	0.07	7560
MI-Orbacus	48	86	0.01	132	0.02	180.	0.03	182	0.03	1402	0.22	6244
MI-IBM-XML	109	9	0.01	13	0.02	14.	0.02	14	0.02	87	0.14	640
IBM-SF	1	3388	0.04	4193	0.05	4659.	0.06	6494	0.08	29659	0.37	80860
MI-HotJava	1	221	0.06	243	0.06	268.	0.07	390	0.10	2385	0.63	3768
MI-Corba	50	89	0.01	113	0.02	115.	0.02	117	0.02	1518	0.23	6551
Java-a	1	0	0.00	0	0.00	0.	0.00	0	0.00	13	0.01	940
JDK.1.0.2	2	9	0.00	10	0.00	10.	0.00	10	0.00	95	0.03	2802
Java-1.18	3	71	0.01	75	0.01	77.	0.01	79	0.01	685	0.09	7411
Java-1.22	28	527	0.03	536	0.03	692.	0.04	759	0.04	9412	0.50	18941
Java-1.30	20	703	0.03	941	0.04	1091.	0.05	1231	0.05	12209	0.51	23772
IDL	1	0	0.00	0	0.00	0.	0.00	0	0.00	1	0.00	320
Unidraw	1	1	0.00	1	0.00	1.	0.00	1	0.00	9	0.00	2468
Lov-obj-ed	157	721	0.19	1005	0.27	1263.	0.34	1470	0.40	2744	0.74	3707
Geode	5	3693	0.20	3975	0.22	4343.	0.24	4849	0.26	15271	0.83	18442
SmartEiffel	1	4	0.00	4	0.00	4.	0.00	4	0.00	280	0.08	3428
Eiffel	51	1961	0.11	2524	0.14	2640.	0.15	2782	0.16	8379	0.48	17559
Laure	215	7	0.00	8	0.00	16.	0.01	22	0.01	137	0.06	2392
Self	101	334	0.01	624	0.01	625.	0.01	625	0.01	8541	0.15	55639
Cecil	141	104	0.02	139	0.02	179.	0.03	218	0.04	1338	0.22	6032
Total	1	13707	0.04	16836	0.05	18700.	0.06	21968	0.07	115477	0.35	325692

FIG. A.4 – Statistiques de coloration bidirectionnelle pour l'heuristique de Ducournau.

Annexe B

Résultats : Coloration d'attributs

Système	nb. classe	intro.	hérités	t (ms)	nb trous				pourc.
					coeur	mini	moy	maxi	moy
SmartEiffel	396	977	1956	3	0	0	0	0	0.0
JDK.1.0.2	604	2231	90864	908	2760	18105	18105	18106	19.9
Unidraw	613	1574	5103	2	5	5	6	7	0.1
Lov-obj-ed	436	1262	3554	36	167	265	268	275	7.5
Geode	1318	2919	14392	486	1339	1625	1639	1651	11.4
Total	3367	8963	115869	1435	4271	20000	20018	20039	17.3

Fig. B.1 – Statistiques de coloration undirectionnelle pour notre heuristique.

Système	nb. classe	intro.	hérités	t (ms)	nb trous				pourc.
					coeur	mini	moy	maxi	moy
SmartEiffel	396	977	1956	3	0	0	0	0	0.0
JDK.1.0.2	604	2231	90864	778	0	0	0	0	0.0
Unidraw	613	1574	5103	3	0	0	0	0	0.0
Lov-obj-ed	436	1262	3554	75	29	37	37	38	1.0
Geode	1318	2919	14392	794	378	516	523	572	3.6
Total	3367	8963	115869	1653	407	553	560	610	0.5

FIG. B.2 – Statistiques de coloration bidirectionnelle pour notre heuristique.

Système	nb classes	Tables SI	coloration unidirectionnelle					coloration bidirectionnelle				
			nb trous			taux trous		nb trous			taux trous	
			max	moy	total			max	moy	total	max	moy
SmartEiffel	397	1956	0	0.0	0	0.0	0.00	0	0.0	0	0.0	0.00
JDK.1.0.2	604	90667	578	30.8	18622	289.0	0.21	0	0.0	0	0.0	0.00
Unidraw	614	5103	1	0.0	5	0.5	0.00	0	0.0	0	0.0	0.00
Lov-obj-ed	436	3554	29	1.7	744	5.0	0.21	7	0.2	68	3.5	0.02
Geode	1318	14392	56	1.8	2338	27.0	0.16	14	0.5	617	3.5	0.04
Total (6/4)	3369	115672	578	6.4	21709	289.0	0.19	14	0.2	685	3.5	0.01

FIG. B.3 – Statistiques de coloration undirectionnelle et bidirectionnelle pour l'heuristique de Ducournau.

Annexe C

Résultats : Coloration de messages

Système	nb. classe	intro.	définies	hérités	t (ms)	nb trous				pourc.
						coeur	mini	moy	maxi	moy
dylan	925	813	1783	70383	131	116	320	354	372	0.5
harlequin	666	416	1016	23167	79	362	696	696	696	3.0
vortex3	1954	933	2496	305755	575	752	1671	1678	1685	0.5
MI-jdk1.3.1	7401	9568	28683	142447	4696	12239	45249	47955	49682	33.7
MI-Orbix	2716	1135	3704	22637	240	1208	1477	1483	1489	6.6
MI-Orbacus	1379	1716	4996	24877	351	3457	3952	4002	4055	16.1
MI-IBM-XML	145	363	945	2338	17	193	353	374	398	16.0
MI-IBM-SF	8793	25000	116152	394375	66157	31291	100706	101037	101536	25.6
MI-HotJava	736	1310	3397	25149	307	4145	13320	13386	13454	53.2
MI-Corba	1699	627	3201	13578	106	1134	1959	1981	1996	14.6
JDK.1.0.2	604	3192	5095	22377	211	732	997	1020	1039	4.6
Unidraw	613	1758	3327	14183	18	97	135	148	157	1.0
Lov-obj-ed	436	3643	5026	37775	1147	6763	14415	14479	14551	38.3
Geode	1318	8078	14214	305560	11806	88642	178417	178724	179001	58.5
SmartEiffel	396	4940	7967	53583	191	311	643	683	743	1.3
Self	1801	26268	29414	1038669	5539	25326	25126	25484	25772	2.5
Cecil	932	2743	4208	73366	405	2383	4281	4323	4370	5.9
Total	32514	92503	235624	2570219	91976	179151	393717	397807	400996	15.5

FIG. C.1 – Statistiques de coloration undirectionnelle pour notre heuristique.

Système	nb. classe	intro.	définies	hérités	t (ms)	nb trous				pourc.
						coeur	mini	moy	maxi	moy
dylan	925	813	1783	70383	134	18	145	145	145	0.2
harlequin	666	416	1016	23167	227	84	133	136	437	0.6
vortex3	1954	933	2496	305755	823	101	209	213	273	0.1
MI-jdk1.3.1	7401	9568	28683	142447	3866	1843	5004	5611	6209	3.9
MI-Orbix	2716	1135	3704	22637	209	42	41	42	45	0.2
MI-Orbacus	1379	1716	4996	24877	411	275	295	320	333	1.3
MI-IBM-XML	145	363	945	2338	19	28	18	28	38	1.2
MI-IBM-SF	8793	25000	116152	394375	62962	7568	18090	18270	18572	4.6
MI-HotJava	736	1310	3397	25149	237	331	726	765	804	3.0
MI-Corba	1699	627	3201	13578	154	386	435	458	484	3.4
JDK.1.0.2	604	3192	5095	22377	168	6	6	6	10	0.0
Unidraw	613	1758	3327	14183	18	0	0	0	0	0.0
Lov-obj-ed	436	3643	5026	37775	854	3554	7183	7276	7362	19.3
Geode	1318	8078	14214	305560	7696	51867	101971	102718	103412	33.6
SmartEiffel	396	4940	7967	53583	142	22	46	47	49	0.1
Self	1801	26268	29414	1038669	3917	11079	10866	11210	11567	1.1
Cecil	932	2743	4208	73366	324	769	1550	1553	1563	2.1
Total	32514	92503	235624	2570219	82161	77973	146718	148798	151303	5.8

FIG. C.2 – Statistiques de coloration bidirectionnelle pour notre heuristique.

Système	nb classes	Tables SI	coloration unidirectionnelle nb trous max	moy	total	taux trous max	moy	coloration bidirectionnelle nb trous max	moy	total	taux trous max	moy
dylan	925	70383	15	0.4	354	0.2	0.01	3	0.2	146	0.0	0.00
harlequin	666	23167	20	1.4	924	0.4	0.04	9	0.6	388	0.3	0.02
vortex3	1954	305755	38	1.1	2149	0.3	0.01	6	0.2	441	0.0	0.00
MI-jdk1.3.1	7401	142445	218	4.4	32668	32.7	0.23	50	0.7	5094	7.3	0.04
MI-Orbix	2716	22637	81	0.5	1473	6.3	0.07	31	0.0	77	2.1	0.00
MI-Orbacus	1379	24877	116	5.6	7728	19.3	0.31	18	0.2	297	3.0	0.01
MI-IBM-XML	145	2338	25	1.4	204	6.0	0.09	7	0.1	17	0.6	0.01
IBM-SF	8793	394375	212	15.0	131910	34.0	0.33	41	3.9	34069	6.8	0.09
MI-HotJava	736	25149	130	17.3	12769	19.2	0.51	67	1.2	912	11.0	0.04
MI-Corba	1699	13578	27	1.3	2257	6.0	0.17	12	0.1	162	2.2	0.01
JDK.1.0.2	604	22365	101	1.8	1094	6.1	0.05	3	0.0	6	0.2	0.00
Unidraw	614	14167	27	0.3	195	2.3	0.01	1	0.2	111	0.3	0.01
Lov-obj-ed	436	37436	167	38.4	16740	6.0	0.45	96	22.8	9926	3.4	0.27
Geode	1318	305560	834	145.0	191065	31.3	0.63	496	60.8	80183	13.3	0.26
SmartEiffel	397	53307	46	1.6	642	0.4	0.01	3	0.0	9	0.0	0.00
Self	1802	1038613	764	13.3	23973	535.0	0.02	514	11.4	20587	497.0	0.02
Cecil	932	73366	73	4.3	3977	0.9	0.05	42	1.1	996	0.4	0.01
Total (6/7)	32517	2569518	834	13.2	430122	535.0	0.17	514	4.7	153421	497.0	0.06

FIG. C.3 – Statistiques de coloration undirectionnelle et bidirectionnelle pour l'heuristique de Ducournau.

Résumé

La coloration comme technique d'implémentation permet de traiter les principaux aspects implémentatoires des langages à objets à typage statique de manière simple et efficace. Nous proposons dans ce rapport une formalisation de cette approche. Nous apportons une preuve de la NP-complétude de la coloration comme technique d'implémentation. Ce résultat nous permet de démontrer la complexité des différentes variantes de ce problème. Enfin, nous avons appliqué nos heuristiques de coloration sur des benchmarks déjà utilisées par Vitek [VHK97] et Ducournau [Duc02b] et avons pu montrer que nous améliorons leurs résultats.
Mots clés : langage à objets, implémentation, coloration.

The method coloring as implementation scheme allows us to treat the main implementation aspects of staticaly typed object-oriented programming languages in a simple and effective way. We propose in this report a formalization of this approach. We give a proof of NP-completeness of the coloring as implementation scheme. This result allows us to demonstrate the complexity of the various variants of this problem. Finally, we applied our heuristics of coloring to benchmarks already used by Vitek [VHK97] and Ducournau [Duc02b] and we were able to show that we improve their results.
Keywords : oriented-object language, implementation scheme, coloring.