Une mesure d'inclusion entre objets structurés. Application à la classification de molécules.

Samuel Wieczorek

Université Joseph Fourier – Grenoble 1

Ecole doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII)

Une mesure d'inclusion entre objets structurés : application à la classification de molécules.

THÈSE

présentée et soutenue publiquement le 7 juillet 2009

pour l'obtention du

Doctorat de l'université Joseph Fourier – Grenoble 1

(spécialité informatique)

par

Samuel Wieczorek

Composition du jury

Président :	Eric Gaussier
Rapporteurs :	Antoine Cornuéjols Jean-Philippe Vert
Examinateurs :	Luc Morin-Allory Gilles Bisson Mirta B. Gordon
Invité :	Sylvaine Roy

Laboratoire des Techniques de l'Ingénierie Médicale et de la Complexité

Mis en page avec la classe thloria.

Remerciements

Je tiens tout d'abord à remercier mon employeur, le Commissariat à l'Energie Atomique (CEA) et plus particulièrement l'Institut de Recherches en Technologies et Sciences pour le Vivant (IRTSV), pour m'avoir permis, depuis le début, de poursuivre ma formation jusqu'à cette dernière étape qu'est la thèse.

Je remercie Monsieur Eric Gaussier pour l'honneur qu'il m'a fait d'avoir présidé mon jury de thèse. J'exprime ma reconnaissance à Messieurs Antoine Cornuéjols et Jean-Philippe Vert, pour avoir accepté d'être rapporteurs de ce travail. Vos remarques ont été d'un très grand intérêt et m'ont permis de prendre un certain recul, nécessaire par rapport à mon propre travail. Je tiens également à remercier Monsieur Luc Morin-Allory pour avoir accepté d'être examinateur lors de ma soutenance. J'ai beaucoup apprécié votre opinion de chimiste et vos remarques qui ont enrichi de façon très fructueuse les discussions lors de la soutenance et élargi les champs d'applications de mon travail.

J'exprime ma très sincère reconnaissance à Mirta B. Gordon et Gilles Bisson pour m'avoir encadré, soutenu, encouragé tout au long de cette thèse.

Lorsque j'ai démarré ce travail, j'avais tendance à m'éparpiller, à envisager un grand nombre d'approches sans méthodologie, et souvent au détriment de la rigueur nécessaire à une activité scientifique. Dès nos premières rencontres, Mirta, tu m'as beaucoup cadré par ton exigence, ton souci du détail, des définitions précises, l'éternelle chasse à l'à peu près. Tout doit être rigoureux, exact, mathématique. La thèse est aussi un temps pour se former et je dois dire que j'ai bien progressé avec toi. Les débuts ont parfois été difficiles mais tu m'as toujours encouragé, expliqué et poussé dans le sens d'un travail de qualité. Pour toutes ces raisons, je te remercie très sincèrement.

Gilles, je te remercie ici mais mon ressenti va bien au-delà de ces quelques mots. Je me rappelle encore le jour où nos routes se sont croisées, au cours d'apprentissage artificiel que tu dispenses au cycle ingénieur du Conservatoire National des Arts et Métiers (CNAM). Ce domaine m'a tout de suite passionné et j'ai très tôt voulu m'orienter vers cette thématique ; ce fut mon objectif en m'inscrivant en DEA puis en thèse pendant lesquels tu m'as suivi, formé et encouragé. Tout comme Mirta, tu m'as forcé à acquérir un regard critique sur mon propre travail en m'apprenant à ne pas avoir une confiance aveugle dans les résultats, à toujours m'interroger, me remettre en questions, creuser, chercher, approfondir. Autant d'aspects qui constituent finalement l'essence même d'un travail de recherche. Merci pour le temps que tu m'as consacré, que ce

soit pour ma thèse ou tout simplement pour m'écouter, autour d'un verre, lorsque j'en avais besoin. J'espère vivement que nous continuerons à travailler ensemble sur d'autres projets. Et maintenant que j'ai l'esprit plus libre, je t'attends pour une petite virée en montagne, à défaut d'une via ferrata ! ;-)

A mon arrivée au CEA (voilà bientôt 10 ans !), j'ai eu le plaisir de travailler avec Mme Sylvaine Roy qui m'a guidé dans le nouveau métier que j'apprenais à l'époque. Tout au long de cette thèse, tu as suivi mon travail avec un grand intérêt ; tes idées et tes remarques m'ont permis de prendre du recul et d'avoir un oeil critique. Tout comme Gilles et Mirta, tu as su recentrer mon travail au moment opportun et toujours dans le but d'améliorer la qualité et la pertinence de ce travail. J'ai trouvé lors de cette thèse la même façon de t'impliquer dans les projets, passionnée, entière, enthousiaste. Merci de m'avoir fait confiance durant toutes ces années où nous avons travaillé ensemble.

Je tiens également à remercier Yves Vandenbrouck, responsable du Laboratoire de Biologie, Informatique et Mathématiques (LBIM), pour m'avoir fait confiance et toujours poussé vers l'autonomie dans ce métier de chercheur auquel je me destine. Je remercie également les permanents, doctorants, post-doctorants et stagiaires pour leur bonne humeur, leur sympathie et leurs encouragements. Je pense à Delphine, Florence, Frédéric, Johan, Samia, Sylvain, qui se sont souvent trouvés face à un ours terré dans sa caverne même à l'heure du déjeuner ! lol

J'en profite également pour remercier Jean-Paul pour son aide précieuse lorsque tu étais mon responsable. Un très grand merci à Vincent car si j'en suis arrivé là aujourd'hui c'est bien grâce à toi (souviens-toi lorsque nous partagions les mêmes bancs aux cours du soir..). T'inviter à ma soutenance était pour moi une marque de sympathie et de sincère remerciement pour ton amitié et ta présence m'a vraiment beaucoup touché.

J'ai également une pensée pour toutes les personnes de mon entourage professionnel, ces personnes qui de près ou de loin m'ont aidé, ont participé à ce travail.

Mais c'est à mes proches, famille et amis, que j'aimerai maintenant adresser toute ma reconnaissance et mes remerciements, à ces personnes qui me sont chères et qui ont participé de près ou de loin à ce travail.

Tout d'abord, Mary, Alain, Emilie, Yannick, amis de longue date ou Isabel, Christian, Marie-Françoise : vous avez toujours été présents pour des sorties ou tout simplement pour discuter, me soutenir lorsque je traversais des moments difficiles.

Je voudrais tout particulièrement remercier Marianne qui n'aime pas trop mes longs discours ! Alors, pour faire court, durant cette dernière année, tu as beaucoup pris sur toi

pour m'aider dans cette entreprise. Je sais que ça n'a pas été sans mal. Je te l'ai déjà dit le soir de la soutenance : finalement, le plus beau cadeau que tu aies pu me faire est ta patience, ton courage et ton abnégation. Je te remercie vraiment du fond du coeur pour avoir cru en moi, pour avoir cru à tout ça.

Je pense aussi à tes enfants Amaury, Guilhem et Enguerrand pour leur joie de vivre, pour leur énergie, leur affection. Chacun à votre façon, vous avez participé à l'achèvement de ce travail et je vous en remercie.

Je ne pourrai clore cette page sans un mot à mes enfants Louis et Ethan. Je me souviens qu'il y a encore quelques semaines, vous me faisiez remarquer que vous m'avez toujours connu avec des "devoirs à la maison". C'est vrai que je n'ai pas été très disponible durant ces années. Cette dernière année en particulier a été difficile, chargée de périodes d'angoisse, de découragement parfois. Mais chaque jour, votre présence me donnait l'allant et la volonté nécessaires pour toujours maintenir le cap. Merci à vous d'être là, merci pour tout...

Grenoble, le 22 juillet 2009

Table des matières

Liste des figures

Liste des tableaux

Liste des Algorithmes

Acronymes

AMC Algorithme de Minimisation des Conflits
APC Arbre Partiellement Couvrant
AO Appariement Optimal
DP de Décomposition Pondérée
SSCM Sous-Structure Commune Maximale
SSC Sous-Structure Commune
FIFO First In First Out
PLI Programmation Logique Inductive
PSC Problème de Satisfaction de Contraintes
CNAM Conservatoire National des Arts et Métiers
CEA Commissariat à l'Energie Atomique
IRTSV Institut de Recherches en Technologies et Sciences pour le Vivant
LBIM Laboratoire de Biologie, Informatique et Mathématiques
IS Indice de Segmentation
IC Indice de Confusion
LPO Logique du Premier Ordre
TP Transition de Phase
AO Appariement Optimal
ISOA Itérative Similarity Optimal Assignment
CAH Classification Ascendante Hiérarchique
NA Non Applicable

Introduction

Un sujet important pour la recherche et ses retombées potentielles sur la santé est la découverte ou la synthèse de petites molécules bio-actives, c'est-à-dire des molécules qui soient effectrices sur des systèmes biologiques. De tels composés qui activent ou inhibent de façon spécifique un système biologique donné peuvent devenir des candidats intéressants dans une perspective de découverte de médicament ou un outil moléculaire pour la recherche en biologie [Stockwell, 2000], [Mayer, 2003], [Inglese *et al.*, 2007], [Maréchal, 2008]. L'identification et la production de ces composés est donc un problème majeur pour la recherche thérapeutique et la recherche en biologie. Les stratégies pour la découverte de petites molécules reposent largement sur le criblage à haut débit de collections de molécules (appelées *chimiothèques*), qui ont traditionnellement été mis en oeuvre dans l'industrie durant les 20 dernières années et est devenu récemment disponible dans les institutions académiques [Stein, 2003], [Fox *et al.*, 2006].

Ces approches à haut débit utilisent des plates-formes robotiques qui permettent de cribler un grand nombre de composés pour sélectionner ceux (appelés "touches") qui produisent l'effet voulu et reproductible sur une cible biologique donnée (une enzyme ou une cellule entière par exemple). La taille des collections de composés disponibles qui peuvent être criblés est assez importante : par exemple, la base ChemNavigator, qui propose des composés disponibles commercialement à partir de fournisseurs internationaux compte actuellement plus de 46,7 millions d'échantillons. Cependant, un tel nombre est encore petit devant la taille de l'espace chimique théorique : le nombre de composés facilement synthétisables est estimé à une fourchette entre 10^{14} et 10^{200} composés [Petit-Zeman, 2003], [Parker and Schreyer, 2004].

Le criblage d'une grande collection de composés peut avoir un coût financier important, il est consommateur de temps et la quantité de matériel biologique requise peut être tout simplement irréaliste. Les biologistes doivent donc souvent revoir leurs ambitions à la baisse et sélectionner un nombre limité de molécules à tester. La conception de chimiothèques pertinentes, souvent appelées "core-libraries" car elles sont supposées refléter avec précision la diversité d'une grande collection de molécules [Dubois *et al.*, 2008], est donc un problème central pour le criblage. La classification automatique de molécules est une approche qui peut générer des sous-ensembles homogènes basés sur

une mesure de similarité et qui permet une définition rationnelle pour une chimiothèque centrale [Willett, 1998].

De plus, après un premier criblage, il a été montré qu'il pouvait être utile de sélectionner des molécules similaires aux touches qui sont ensuite testées à travers d'autres étapes de validation [Engels, 2000]. Le deuxième intérêt à la recherche d'analogues structuraux est qu'une touche présente rarement toutes les propriétés chimiques et biologiques optimales et que l'identification de molécules similaires peut permettre d'améliorer la sensibilité et la spécificité des molécules. Les scientifiques sont donc demandeurs d'outils de comparaison des molécules du point de vue de leur structure et d'analyse de la diversité des chimiothèques.

Dans ce contexte, on peut distinguer trois objectifs principaux :
- rechercher si une molécule se trouve dans une chimiothèque ;
- rechercher si une touche partage une ou plusieurs sous-structures communes avec les composés d'une chimiothèque ;
- rechercher les molécules d'une chimiothèque qui contiennent une structure particulière (par exemple, un châssis moléculaire ou une fonction biologique d'intérêt).

Dans le premier cas, on peut utiliser une mesure de similarité "symétrique" (ou absolue) qui évalue la similarité de deux composés en les considérant de la même façon. Dans les deux autres cas, on se servira plutôt d'une mesure de similarité "asymétrique" (ou relative) qui quantifie la similarité d'une molécule m_1 relativement à une molécule m_2, ce qui revient à évaluer un *degré d'inclusion* entre les deux molécules. Une telle mesure permet en outre de comparer des molécules ayant des tailles différentes, en terme de nombre d'atomes. Par exemple, si deux molécules m_1 et m_2, de taille très différente, partagent la même sous-structure commune, la valeur d'une mesure de similarité classique refléterait la différence de taille entre les deux molécules tandis qu'une mesure de similarité relative (de la plus petite molécule relativement à la plus grande) aurait une valeur proche de 1 et refléterait plus la présence de la sous-structure commune.

Le travail présenté dans ce mémoire se place dans le contexte de la comparaison de molécules mais plus généralement dans la comparaison d'objets structurés. Plus précisément, le problème traité consiste à comparer deux objets structurés A et B en :
- recherchant des sous-parties de l'objet A qui se retrouvent dans l'objet B ; autrement dit, des sous-structures communes aux deux objets ;
- évaluant la taille de ces sous-parties communes par rapport à l'objet A pour définir un degré d'inclusion entre les objets A et B.

Conceptuellement, ces problèmes de recherche sont souvent simples mais difficiles d'un point de vue calculatoire car le temps d'exécution peut augmenter de façon exponentielle avec la taille du problème. L'objectif de ce travail est alors de proposer des méthodes heuristiques pour réduire la complexité du problème.

Ce mémoire de thèse est organisé en deux parties.

Partie I : Degré d'inclusion entre objets structurés

Dans cette première partie, nous proposons un algorithme d'évaluation du degré d'inclusion entre deux objets structurés. Cette partie comporte 3 chapitres :

- Le chapitre 1 décrit la terminologie qui sera utilisée dans ce travail pour représenter et comparer les objets structurés en théorie des graphes et en logique du premier ordre ;
- Le chapitre 2 traite des opérations de comparaison classiques que sont l'égalité, l'inclusion, la similarité et le degré d'inclusion. Les notions utilisées en logique du premier ordre et dans les graphes pour caractériser les parties communes à deux objets structurés sont abordées. Quelques algorithmes de comparaison de deux clauses sont ensuite décrits ;
- Le chapitre 3 présente l'algorithme que nous proposons pour évaluer le degré d'inclusion entre deux objets structurés représentés sous forme de clauses logiques ;
- Dans le chapitre 4, les performances de l'algorithme sont testées sur des problèmes aléatoires complexes dans le cadre de la transition de phase en Programmation Logique Inductive.

Partie II : Application à la chimie

Cette partie comporte 2 chapitres consacrés à la mise en application de l'algorithme à des composés chimiques.

- Le chapitre 5 est un état de l'art des mesures de similarité moléculaire couramment utilisées pour rechercher des analogues structuraux ou pour prédire la bioactivité des molécules avec des méthodes d'apprentissage artificiel ;
- Dans le chapitre 6, l'algorithme du calcul d'un degré d'inclusion est utilisé dans un problème de classification automatique de molécules. Dans ce contexte, une étude a été menée pour comparer notre méthode avec d'autres mesures de similarité moléculaire, dont des fonctions noyau.

Première partie

Degré d'inclusion entre objets structurés

Chapitre 1

Terminologie

La représentation de données complexes ou la formalisation de certains problèmes nécessite d'utiliser des langages suffisamment expressifs pour intégrer leur complexité. Les langages de représentation sont définis par des symboles capables de représenter l'information à traiter ainsi qu'une série d'opérations sur ces symboles qui modélisent un raisonnement comme c'est le cas en Représentation des Connaissances (RC) ou encore dans les systèmes de logique. La diversité et la variété du monde ne permettent pas de le représenter ni de l'exploiter par un formalisme unique.

Parmi les langages de représentation existants, on trouve des langages simples comme les langages propositionnels où les objets sont décrits par une liste de caractéristiques (ou descripteurs). Prenons l'exemple d'un véhicule automobile. Un langage propositionnel simple pour représenter cet objet serait composé de l'ensemble E des symboles correspondant au nom des pièces qui le composent : E = {vitres, optiques, sièges, moteur, roues, etc...}. Lorsque les objets sont décrits par des langages de ce type, des algorithmes très efficaces peuvent être utilisés pour analyser les données de manière automatique.

Cependant, ces langages ne sont pas suffisamment expressifs pour représenter correctement des objets complexes sans perte d'information. Dans le cas du véhicule par exemple, on aimerait pouvoir représenter la façon dont les pièces de l'ensemble E sont reliées les unes aux autres, c'est-à-dire d'exprimer les relations entre ces éléments. Cette deuxième représentation est plus complexe que les langages propositionnels mais en même temps plus riche car elle capture ce que l'on appelle la *structure* de l'objet. Un certain nombre de ces langages sont issus de la représentation des connaissances, des logiques de description [Baader *et al.*, 2003] ou des travaux en sciences cognitives comme les graphes conceptuels [Sowa, 1984]. De nombreuses disciplines s'intéressent à la représentation et au traitement des données structurées : la reconnaissance des formes, l'analyse d'images, la théorie des graphes, le webmining, les logiques de description, la

Programmation Logique Inductive (PLI) [Muggleton and Raedt, 1994] ou les Problème de Satisfaction de Contraintes (PSC) [Tsang, 1993].

Nous nous intéressons ici à la représentation d'un objet structuré d'un point de vue syntaxique, c'est-à-dire à l'ensemble des symboles d'un langage donné qui permettent de décrire les caractéristiques structurales de l'objet. Dans ce chapitre, nous commençons par décrire, de façon informelle, la notion d'*objet structuré*. Nous présentons ensuite la syntaxe de deux langages adaptés à la représentation d'objets structurés que nous allons utiliser tout au long de ce mémoire :

- la théorie des graphes qui permet également de modéliser certains problèmes d'optimisation ;
- la logique des prédicats qui est utilisée dans une variété de domaines comme la linguistique, la reconnaissance des formes, les sciences cognitives ou l'apprentissage artificiel.

1.1 Les données structurées

La notion de structure est très générale et se retrouve dans des domaines aussi divers que la psychologie, la philosophie, les mathématiques et, de manière générale, dans les sciences. Une objet structuré est la représentation d'une information dans laquelle des parties élémentaires, appelées *entités*, sont connectées les unes aux autres par des *relations*. Plus généralement, un objet structuré O est défini par l'ensemble des entités $E = \{e_1, \ldots, e_n\}$ dont il est composé et l'ensemble des relations $R = \{r_1, \ldots, r_m\}$ qui relient les entités entre elles.

Pour définir une donnée structurée, comme dans tout processus de représentation de la réalité, à l'aide d'un langage de représentation, il faut définir la granularité de la représentation, c'est-à-dire le niveau de détail avec lequel l'information est décrite. Dans l'exemple de l'automobile, est-ce que le moteur correspond à une partie élémentaire du véhicule ou alors est-il vu comme un objet complexe composé de parties élémentaires ? Il faut également définir avec précision la correspondance entre les informations réelles (*i.e.* les entités et les relations) et leur représentation : c'est la notion d'interprétation.

1.1.1 Les entités

Une entité représente une partie élémentaire dans un objet structuré (une pièce d'une voiture ou une variable mathématique par exemple) et existe dans l'objet indépendamment d'autres entités. Elle peut ainsi être décrite et manipulée sans qu'il soit nécessaire de connaître les autres entités de l'objet.

Si le langage le permet, les propriétés de chaque entité peuvent être précisées à l'aide d'un ensemble de descripteurs (ou attributs) typés. Les attributs numériques correspondent à des valeurs entières ou réelles et permettent de décrire des données quantitatives. Les attributs symboliques sont composés d'un ensemble énumérable de valeurs symboliques. Lorsqu'il y a une notion d'ordre, on parle de type ordinal comme par exemple {petit, moyen, grand}.

1.1.2 Les relations

Une relation met en correspondance un certain nombre d'entités d'un même objet. Elle peut, par exemple, correspondre à des notions comme l'égalité et l'inclusion de deux entités.

Plus formellement, étant donné l'ensemble E des entités d'un objet O et l'ensemble R des relations dans l'objet O, la relation r d'arité n est définie dans $E^n \rightarrow R$. Elle s'applique à n entités appelées ses *arguments* ; on notera $r(e_1, \ldots, e_n)$. Suivant son arité, une relation porte un nom différent : on dira qu'elle est unaire pour $n = 1$, binaire pour $n = 2$, ternaire pour $n = 3$, etc... Tout comme une entité, une relation peut-être décrite par un ensemble d'attributs, de type numérique ou symbolique.

Une relation r est dite *réflexive* si une entité e_1 peut être en relation avec elle-même : $r(e_1, e_1)$. La relation binaire r de $E^2 \rightarrow R$ est *symétrique* si et seulement si lorsqu'une entité e_1 dans E est en relation avec une autre entité e_2 dans E, l'entité e_2 est elle aussi en relation avec e_1, c'est-à-dire $\forall(e_1, e_2) \in E^2$, $r(e_1, e_2) \Rightarrow r(e_2, e_1)$. L'ordre (*i.e.* la position) des arguments dans une relation symétrique n'est pas important, ce qui n'est pas vrai pour les relations non symétriques. Par exemple, dans la relation "est_le_père_de", si la proposition est_le_père_de(Jean, Louis) est vraie, sa réciproque est fausse ; dans ce cas, les arguments ne peuvent pas être permutés. En étendant cette notion aux relations n-aires, nous dirons qu'une relation est commutative si la position des arguments peut être modifiée.

1.2 Le formalisme des graphes

Les graphes sont des objets mathématiques qui permettent de représenter des objets structurés. Ce formalisme est intéressant car les objets sont représentés de manière graphique rendant ainsi leur conceptualisation plus aisée. Par ailleurs, la *théorie des graphes* (qui prend ses origines au $XVIII^e$ siècle) s'est considérablement développée au XX^e siècle notamment grâce aux travaux de Claude Berge [Berge, 1958], [Berge, 1969] et propose aujourd'hui des algorithmes pour résoudre de nombreux types de problèmes modélisés sous forme de graphes.

1.2.1 Définitions

Définition (Graphe simple)
Un graphe simple $G = (S, A)$ est défini par le couple (S, A) où :

- S est un ensemble d'entités $S = \{s_1, \ldots, s_n\}$, appelées les sommets (ou nœuds) du graphe ;
- A est un sous-ensemble de S^2 dont les éléments (s_i, s_j) sont les arêtes du graphe.

L'arête (s_i, s_j) est incidente aux sommets s_i et s_j, qui sont dits *adjacent*s. Le *degré* d'un nœud est défini comme le nombre d'arêtes qui lui sont incidentes.

Définition (Graphe orienté)
Un graphe orienté est un graphe $G = (S, A)$ où A est un ensemble d'arcs de la forme (s_i, s_j), tel que l'arc part de s_i et arrive en s_j. On dit que s_i est l'origine de l'arc et s_j son extrémité, que l'arc est sortant en s_i et incident en s_j. Le degré entrant $d-$ d'un sommet est le nombre d'arcs qui arrivent à ce sommet et le degré sortant $d+$, le nombre d'arcs qui en partent.

Il est possible d'affecter aux nœuds et aux arêtes (ou arcs) d'un graphe un ensemble d'étiquettes (valeurs numériques ou symboliques). Le graphe est alors dit *étiqueté*.

Définition (Graphe étiqueté)
Étant donné un ensemble fini d'étiquettes L_S sur les sommets et un ensemble fini d'étiquettes L_A sur les arêtes (ou arcs), un graphe étiqueté est défini par :

- S est l'ensemble des sommets ;
- $et_S \subseteq S \times L_S$ est l'ensemble des couples (s_i, l_s) tels que le sommet s_i a pour étiquette l_s ;
- $et_A \subseteq S \times S \times L_A$ est l'ensemble des triplets (s_i, s_j, l_a) tels que l'arête (ou l'arc) (s_i, s_j) a pour étiquette l_a.

La figure 1.1 montre comment la structure 2D d'une molécule se représente sous la forme d'un graphe étiqueté non orienté.

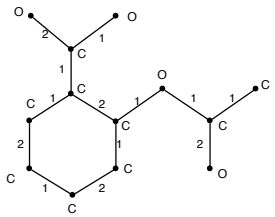


FIGURE 1.1 – Représentation d'une molécule (ici, l'aspirine) par son graphe moléculaire : un graphe étiqueté non orienté dont les sommets correspondent aux atomes de la molécule (étiquetés par le type d'atome) et les arêtes correspondent aux liaisons (étiquetées par la valence de la liaison).

Suivant les applications, il est possible d'ajouter d'autres descripteurs, parmi les quelques 3300 que l'on recense aujourd'hui [Todeschini and Consonni, 2000], comme par exemple :

- la charge électronique, les propriétés pharmacophoriques, l'appartenance à un cycle aromatique ou à un groupe fonctionnel, le nombre d'électrons libres ou encore l'état d'hybridation pour les atomes ;
- l'appartenance à un cycle ou à un groupe fonctionnel, le fait d'être en rotation libre pour les liaisons.

Une molécule peut également être décrite de manière globale par ses propriétés physico-chimiques comme la masse et la surface moléculaires, le coefficient de lypophilie.

La correspondance entre un graphe simple et une représentation par des entités et des relations est immédiate : les sommets (resp. les arêtes) du graphe correspondent aux entités (resp. aux relations) de l'objet structuré. Les arêtes représentent les relations (binaires) symétriques entre les entités. Si le graphe est orienté, les arcs entre les sommets représentent des relations binaires non symétriques.

Un *chemin* de longueur k est une séquence finie d'arêtes consécutives $(a_1, \ldots, a_k)$ dans un graphe avec un sommet de départ et un sommet d'arrivée. La longueur du chemin est le nombre d'arêtes traversées. Une *chaîne* est une séquence finie de sommets $(s_1, \ldots, s_k)$ tels que, pour tout i, il existe une arête entre les sommets s_i et s_{i+1} (les sommets sont adjacents deux à deux). Un *cycle* est une chaîne $(s_1, \ldots, s_k)$ dont le premier et le dernier sommet sont identiques et tous les autres sommets distincts. Un graphe *acyclique* est un graphe qui ne contient pas de cycle. Le *voisinage* d'un sommet dans un graphe est l'ensemble des sommets qui lui sont adjacents. Un graphe est *connexe* s'il est possible, à partir de n'importe quel sommet, de rejoindre tous les autres en parcourant les arêtes (figure 1.2).

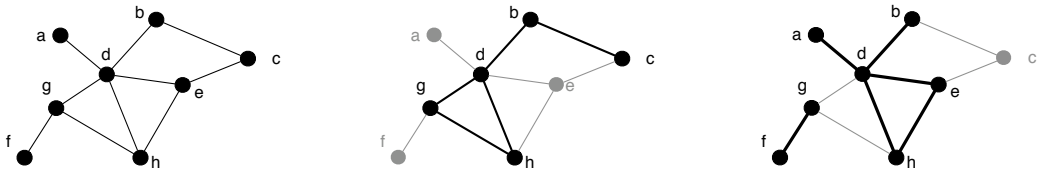
FIGURE 1.2 – Le graphe G est connexe et le graphe H n'est pas connexe.

Définition (Sous-graphe induit)
Pour un sous-ensemble de sommets S' inclus dans S, le sous-graphe de $G = (S, A(S))$

induit par S' est le graphe $G' = (S', A(S'))$ dont l'ensemble des sommets est S' et l'ensemble des arêtes $A(S')$ est formé par toutes les arêtes de G ayant leurs deux extrémités dans S'. Autrement dit, le graphe G' est obtenu en enlevant un ou plusieurs sommets au graphe G, ainsi que toutes les arêtes incidentes à ces sommets (figure 1.3(b)).

Définition (Graphe partiel)
Le graphe partiel de $G = (S, A)$ est le graphe $G' = (S, A')$ dont l'ensemble des sommets est S et l'ensemble des arêtes A' qui est un sous-ensemble d'arêtes inclus dans A. Autrement dit, le graphe G' est obtenu en enlevant une ou plusieurs arêtes au graphe G (figure 1.3(c)).

Un sous-graphe de G est entièrement défini (induit) par l'ensemble de ses sommets S, et un graphe partiel de G par l'ensemble de ses arêtes A.

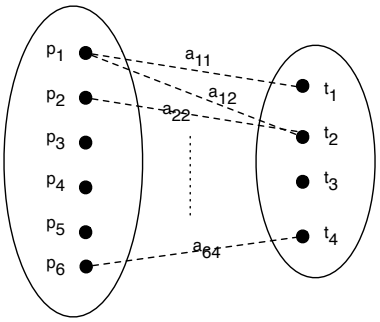


(a) Graphe G (b) Sous-graphe induit de G (c) Graphe partiel de G

FIGURE 1.3 – Soit $G = (S, A)$ avec $S = \{a, b, c, d, e, f, g, h\}$. Le sous-graphe induit du graphe G est défini par l'ensemble de sommets S = { b, c, d, g, h }. Le graphe partiel du graphe G est défini par l'ensemble d'arêtes A = { (a, d), (b, d), (d, e), (e, h), (h, d), (f, g) }.

Définition (Graphe bipartite)
Un *graphe bipartite* est un graphe $G = (U, V, A)$ dont l'ensemble des sommets S peut être divisé en deux sous-ensembles disjoints U et V tels que chaque arête dans A relie un sommet dans U à un sommet dans V et qu'aucune paire de sommets dans un même ensemble ne soient adjacents.

Les graphes bipartites sont très utiles pour modéliser des problèmes d'appariement comme l'affectation de tâche. Etant donné un ensemble P de personnes et un ensemble T de tâches à réaliser, le problème consiste à affecter une tâche par personne sachant que toutes les personnes ne sont pas aptes à réaliser toutes les tâches. Ce problème peut être modélisé par le graphe bipartite $G = (P, T, A)$. Si une personne p_i est apte à réaliser une tâche t_j, alors il y a une arête entre p_i et t_j dans le graphe. Les graphes bipartites dont les arêtes sont étiquetées (valuées) sont également intéressants pour modéliser les problèmes d'optimisation.



$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \\ a_{61} & a_{62} & a_{63} & a_{64} \end{pmatrix} \end{array}$$

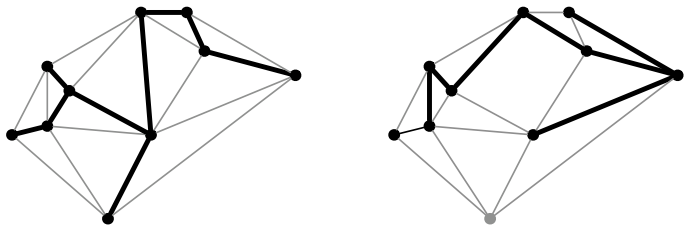
FIGURE 1.4 – Exemple de graphe bipartite. L'aptitude a_{ij} d'une personne p_i à effectuer une tâche t_j est quantifiée dans la matrice à droite..

Un problème d'optimisation peut également se représenter sous forme d'un graphe bipartite où chaque sommet p_i est relié à chaque sommet t_j. Dans ce cas, la valeur d'aptitude pourra être égale à 0 pour indiquer que la personne p_i n'est pas apte à réaliser la tâche t_j. L'avantage de cette représentation est que la matrice qui contient les valeurs d'aptitude est pleine, comme c'est le cas sur l'exemple de la figure 1.4.

Définition (Arbre Couvrant)
Un arbre couvrant G' (ou arbre maximal) d'un graphe connexe G est un sous-ensemble T de A tel que $G' = (S,T)$ est un graphe connexe et acyclique et tout sommet $s \in S$ apparaît dans une arête de T.

Définition (Arbre Partiellement Couvrant (APC))
Un arbre partiellement couvrant G' d'un graphe connexe G est un sous-ensemble d'arêtes T de A tel que $G' = (S,T)$ est un graphe connexe et acyclique et un sous-ensemble de sommets $s \in S$ apparaît dans une arête de T. Autrement dit, c'est un arbre qui ne contient pas tous les sommets du graphe G.

Un même graphe peut comporter différents arbres couvrants et APC. La figure 1.5 montre un exemple d'arbre couvrant et un exemple d'APC.



(a) Arbre couvrant

(b) Arbre Partiellement Couvrant

FIGURE 1.5 – Exemple d'arbres couvrant et partiellement couvrant.

La notion de graphe peut être généralisée en considérant que p arcs (ou arêtes) distincts, de même sens ou non, relient un sommet s_i à un sommet s_j ; on parle alors de multigraphe ou de p-graphe (illustré sur la figure 1.6).

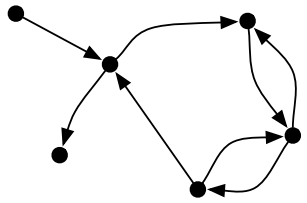


FIGURE 1.6 – Exemple de multi-graphe orienté.

Les hypergraphes [Berge, 1987] généralisent la notion de graphe dans le sens où les arêtes ne relient plus 1 ou 2 sommets, mais un nombre n de sommets (n compris entre 1 et le nombre de sommets de l'hypergraphe) ; les arêtes (ou arcs) représentent alors des relations n-aires. Ce type de graphe se dessine difficilement. Certains concepts (comme la connexité) s'exportent mal des graphes aux hypergraphes, et encore moins aux hypergraphes orientés pour lesquels aucune des diverses définitions possibles ne s'est encore imposée.

1.3 La logique des prédicats

La logique a été l'un des premiers formalismes proposés pour représenter le monde ; c'est principalement sous l'impulsion d'Aristote qu'elle prit son envol. On a assisté durant le XX^e siècle au développement d'une approche mathématique et informatique de la logique. La Logique du Premier Ordre (LPO) (ou calcul des prédicats) a été proposée par Frege au début du XX^e siècle pour formaliser le langage des mathématiques.

Elle trouve maintenant de nombreuses applications en ingénierie, en intelligence artificielle, en linguistique, en psychologie cognitive ou en philosophie analytique. Outre le formalisme de représentation des objets du monde, les langages logiques sont accompagnés d'opérations qui modélisent les raisonnements comme la déduction, l'induction ou la généralisation. Ces mécanismes sont utilisés par exemple en PLI [Muggleton, 1992].

Comme pour les graphes, nous précisons ici les éléments syntaxiques de ce formalisme qui sont nécessaires pour représenter des objets structurés.

1.3.1 Définitions

Nous considérons ici des expressions de la logique des prédicats basées sur un alphabet composé de :

- un ensemble de C symboles appelés *constantes*, éventuellement vide ;
- un ensemble V de *variables* [1]. Les variables prennent leurs valeurs dans un ensemble de constantes appelé *domaine* ;
- un ensemble P de symboles de prédicats. Un *prédicat* est une fonction d'un ensemble d'arguments (son domaine de définition) sur l'ensemble image {faux, vrai} ;

Chaque symbole de prédicat ou de fonction a une arité (un entier strictement positif) qui détermine le nombre d'arguments auquel il est appliqué.

- les symboles $\forall$ (quel que soit) et $\exists$ (il existe), appelés quantificateurs ;
- les symboles $\neg$ (négation), $\wedge$ (et, conjonction), $\vee$ (ou, disjonction) et $\rightarrow$ (implication), qui sont les connecteurs logiques du calcul des propositions.

Les règles de formation des expressions logiques sont les suivantes.

Définition (Terme)
Un terme est défini récursivement :

- une variable est un terme ;
- une constante est un terme ;
- si p est un symbole de prédicat d'arité n et $t_1, \ldots, t_n$ sont des termes, alors $p(t_1, \ldots, t_n)$ est un terme.

Définition (Littéral)
Un littéral est un symbole de prédicat appliqué à des termes, éventuellement précédé du symbole $\neg$. S'il n'est pas précédé de ce symbole, il est *positif*. C'est une expression de la forme $p(t_1, \ldots, t_n)$ ou $\neg p(t_1, \ldots, t_n)$, où p est un symbole de prédicat et les t_i sont des termes.

Exemple
Le littéral *enfant(x, y, z)* est défini de la manière suivante. Il est formé du symbole de prédicat *enfant* qui possède trois arguments : les deux premiers désignent deux personnes (la première variable désigne l'enfant de la personne représentée par la variable y) et le dernier indique l'âge de cette personne. Ce littéral signifie que la personne désignée par x est l'enfant de y et a z ans. Si $x =$ Henri, $y =$ Martin et $z = 5$, alors le littéral *enfant(Henri, Martin, 50)* indique que Henri est l'enfant de Martin et a 5 ans.

1. Par convention, nous notons les noms des variables (resp. les noms des constantes) en lettres minuscules (resp. en lettres majuscules).

Nous définissons la *signature* d'un littéral de la forme $p(v_1, \ldots, v_m, C_1, \ldots, C_n)$ qui contient m variables et n constantes, comme la donnée de son arité et de la signification de chacun de ses arguments ainsi que leur position dans le littéral (*i.e.* indice de la variable ou de la constante). Deux littéraux ont la même signature s'ils possèdent le même nombre de variables et de constantes et si les variables et les constantes situées à la même position ont le même type et la même sémantique.

Définition (Clause)
Une clause est une disjonction finie de littéraux dont toutes les variables sont quantifiées universellement, c'est-à-dire commandées par $\forall$. On simplifie en général l'écriture d'une clause en supprimant les quantificateurs $\forall$: toute variable doit donc être interprétée comme étant universellement quantifiée.

Définition (Clause de Horn)
Une clause de Horn est une clause qui a soit zéro soit un seul littéral positif.

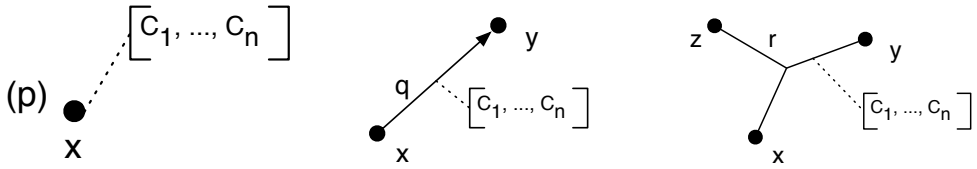
Définition (Clause définie)
Une clause définie est une clause de Horn qui a exactement un littéral positif. Quand on a supprimé les quantificateurs universels, une clause définie s'écrit donc : $A \vee \neg B_1 \vee \ldots \vee \neg B_m$. Cette notation peut se transformer en : $A \leftarrow B_1, \ldots, B_m$.

En logique du premier ordre, un objet structuré peut être représenté par une clause de Horn définie : les variables correspondent alors aux entités et les symboles de prédicats aux relations. Les constantes qui apparaissent dans les littéraux définissent des caractéristiques (ou attributs). Une clause Datalog [2] [Ceri *et al.*, 1989] est une conjonction de littéraux positifs dont les arguments sont limités aux variables et aux constantes.

Nous définissons maintenant la sémantique des littéraux que nous allons utiliser dans la suite de ce document.

Un littéral lit_v de la forme $p(x, C_1, \ldots, C_n)$ est construit sur un symbole de prédicat p qui prend en argument une variable x et n constantes. Dans notre interprétation, un tel littéral définit une entité représentée par la variable x dont les propriétés sont les n constantes. En termes de graphes, si $n = 0$, alors ce littéral définit un sommet ; si $n \geq 1$, il définit un sommet étiqueté (les n constantes représentant les étiquettes du sommet) comme l'illustre la figure 1.7(a).

2. Le langage Datalog est un sous-ensemble syntaxique de Prolog.



(a) (b) (c)

FIGURE 1.7 – Représentation graphique de différents types de littéraux. (a) Le littéral $p(x, C_1, \ldots, C_n)$ définit la variable x qui est décrite par n constantes. (b) Le littéral $q(x, y, C_1, \ldots, C_n)$ définit une relation binaire entre les variables x et y qui est décrite par n constantes. (c) Le littéral $r(x, y, z, C_1, \ldots, C_n)$ définit une relation ternaire entre les variables x, y et z qui est décrite par n constantes.

Un littéral lit_r de la forme $p(x_1, \ldots, x_m, C_1, \ldots, C_n)$ avec $m > 1$ définit une relation n-aire de type p entre les m variables $x_1, \ldots, x_m$; nous appelons ce type de littéral un *littéral relationnel*. Cette relation est caractérisée par n attributs représentés par les n constantes typées. On distingue les cas suivants :

- si $m = 2$, la relation est binaire et ce littéral correspond à l'arête (x_1, x_2) dans un graphe où les deux variables x_1 et x_2 sont les sommets. Si la relation n'est pas commutative, le littéral représente l'arc (x_1, x_2) où x_1 est l'origine de l'arc et x_2 son extrémité. Cette arête (ou arc) est étiqueté par n attributs pour $n > 0$. Un exemple de ce type de littéral est illustré sur la figure 1.7(b) ;
- lorsque $m \geq 3$, on se trouve dans le cas des hypergraphes. Les m constantes sont les étiquettes associées à l'hyperarête (ou hyperarc). Un exemple de ce type de littéral est illustré sur la figure 1.7(c).

Le langage des clauses Datalog est syntaxiquement équivalent à celui des graphes (figure 1.8) [Rensink, 2004]. Ce sont tous deux des exemples de langages de type entité-relation.

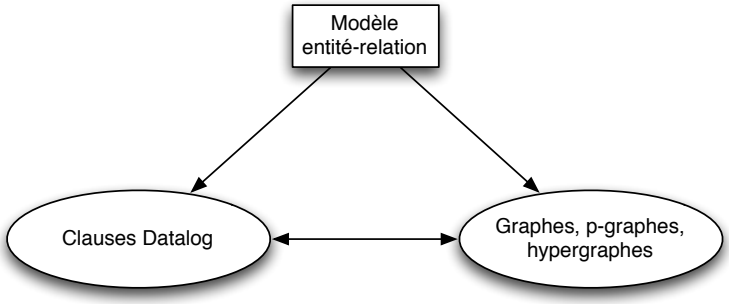


FIGURE 1.8 – Les clauses Datalog et les graphes sont des formalismes équivalents pour représenter des objets structurés selon le modèle entité-relation.

Exemple.
Pour représenter la structure de la molécule d'aspirine, illustrée sur la figure 1.1, nous nous donnons le langage formé du :

- symbole de prédicat "atome" qui a deux arguments : une variable représentant un atome et une constante nominale qui indique le type d'atome ;
- symbole de prédicat "liaison" qui a trois arguments : deux variables représentant chacune un des atomes impliqués dans la liaison et une constante de type entier indiquant la valence de la liaison ;
- un ensemble de constantes typées : deux constantes entières (1 et 2) qui quantifient la valence des liaisons, et deux constantes nominales (C et O) qui définissent le type d'atome (carbone et oxygène) ;
- un ensemble de variables $V = \{a_1, a_2, \ldots, a_{13}\}$ représentant les atomes de la molécule.

Pour décrire une molécule, il ne peut exister deux littéraux différents, construits sur le symbole de prédicat atome, et portant sur la même variable. Autrement dit, chaque atome est unique. De la même façon, on ne peut définir deux liaisons différentes portant sur la même paire d'atomes.

A l'aide de ce langage, la molécule d'aspirine (notée *M*) s'écrit alors :
M $\leftarrow$ atome(a_1, O), atome(a_2, O), atome(a_3, C), atome(a_4, C), atome(a_5, C), atome(a_6, C), atome(a_7, C), atome(a_8, C), atome(a_9, C), atome(a_{10}, O), atome(a_{11}, C), atome(a_{12}, C), atome(a_{13}, O), liaison(a_1, a_3, 2), liaison(a_2, a_3, 1), liaison(a_3, a_4, 1), liaison(a_4, a_5, 1), liaison(a_5, a_6, 2), liaison(a_6, a_7, 1), liaison(a_7, a_8, 2), liaison(a_8, a_9, 1), liaison(a_4, a_9, 2), liaison(a_9, a_{10}, 1), liaison(a_{10}, a_{11}, 1), liaison(a_{11}, a_{12}, 1), liaison(a_{11}, a_{13}, 2).

Chapitre 2

Comparer des objets structurés

Les méthodes d'apprentissage artificiel reposent sur la recherche des ressemblances et des dissemblances entre objets. C'est une tâche complexe, surtout dans le cas d'objets structurés, mais qui est fondamentale dans ce genre d'approche. Quels que soient les algorithmes, les opérations de comparaison utilisées sont toutes caractérisées par la nature de l'opération (égalité ou inclusion par exemple) et le type de résultat qu'elles renvoient (valeur booléenne ou continue).

Parmi les opérations classiques à valeur booléenne (que nous appelons des *tests*), on trouve le *test d'égalité* et le *test d'inclusion*. Le test d'égalité indique si deux objets A et B sont égaux tandis que le test d'inclusion indique si un objet A est inclus dans un objet B. Pour avoir une idée plus précise de la proximité de ces deux objets, on pourra utiliser des opérations qui renvoient une valeur continue comme une mesure de *similarité* qui quantifie la ressemblance entre les deux objets et un *degré d'inclusion* qui évalue l'importance de l'inclusion de l'objet A dans l'objet B.

Dans ce chapitre, nous nous intéressons aux différentes manières de comparer deux clauses Datalog en faisant également un parallèle avec la théorie des graphes. La section 2.1 présente un cadre général pour identifier les parties communes à deux objets structurés, qui sont à la base des opérateurs de comparaison et repose sur la notion d'appariement. Les appariements sont caractérisés dans la section 2.2. Dans la section 2.3, nous présentons quelques approches en logique du premier ordre pour tester l'inclusion de clauses Datalog. Nous nous intéressons dans la section 2.4 à l'évaluation d'une mesure de similarité entre clauses. Enfin, dans la section 2.5, nous présentons le problème de l'évaluation d'un degré d'inclusion entre clauses.

2.1 Identifier l'intersection d'objets structurés

2.1.1 Approche générale

Pour comparer deux objets, il est courant et assez intuitif de faire le rapport entre la quantité de caractéristiques communes à ces deux objets et l'ensemble de leurs caractéristiques [Lin, 1998]. Par exemple, dans son travail séminal, [Tversky, 1977] s'intéresse à la manière dont les êtres humains perçoivent la notion de similarité et propose un cadre formel pour en faire l'étude. Il définit la similarité entre deux objets A et B décrits dans un langage propositionnel, c'est-à-dire représentés par un ensemble de caractéristiques, comme une fonction S de la forme suivante :

$$S(A,B) = \frac{f(E_A \cap E_B)}{f(E_A \cap E_B) + \alpha.f(E_A \setminus E_B) + \beta.f(E_B \setminus E_A)} \text{ avec } \alpha,\beta \geq 0 \qquad (2.1)$$

où E_A (resp. E_B) est l'ensemble des caractéristiques de A (resp. de E_B) et f est une fonction croissante monotone [3]. La différence $(E_A \setminus E_B)$ correspond aux caractéristiques présentes dans E_A mais absentes de E_B et $(E_B \setminus E_A)$ correspond aux caractéristiques présentes dans E_B mais absentes de E_A. Les paramètres α et β pondèrent les différences entre les ensembles de caractéristiques. Si ces paramètres sont égaux à 1, alors l'équation 2.1 se simplifie de la façon suivante :

$$S(A,B) = \frac{f(E_A \cap E_B)}{f(E_A \cup E_B)} \qquad (2.2)$$

A partir de la formule de Tversky, il est possible de définir une fonction d'inclusion, à valeur réelle dans [0, 1], qui évalue la quantité de caractéristiques de l'objet A (appelé *objet source*) que l'on retrouve dans l'objet B (appelé *objet de référence*). Cette fonction Di se déduit de l'équation 2.1 en fixant la valeur des paramètres $\alpha = 1$ et $\beta = 0$ et s'écrit :

$$Di(A,B) = \frac{f(E_A \cap E_B)}{f(E_A)} \qquad (2.3)$$

Si les fonctions S et Di sont définies pour renvoyer une valeur booléenne, elles correspondent alors respectivement à un test d'égalité et un test d'inclusion.

La définition des quatre opérateurs de comparaison que sont l'égalité, la similarité, l'inclusion et le degré d'inclusion est donc basée sur deux étapes : i) l'identification de

3. C'est-à-dire que, pour tout couple d'éléments (x_1, x_2) tel que $x_1 \leq x_2$, on a $f(x_1) \leq f(x_2)$.

l'intersection des deux objets comparés, ii) la normalisation de cette intersection. La différence entre la similarité et l'égalité d'une part et l'inclusion et le degré d'inclusion d'autre part se situe au niveau du facteur de normalisation :

- dans le cas de la similarité et de l'égalité, qui sont des grandeurs *symétriques*, ce facteur tient indifféremment compte des deux objets (objet source et objet de référence) ;
- en ce qui concerne le test d'inclusion et le degré d'inclusion, qui sont des grandeurs *asymétriques*,, le facteur de normalisation n'est relatif qu'à l'objet source.

Dans le cas de la similarité (équation 2.2) ou de l'inclusion (équation 2.3), le facteur de normalisation ne dépend d'aucune comparaison entre les objets et peut donc être évalué facilement. Par contre, la recherche de l'intersection des objets est plus difficile car c'est l'étape où les structures des objets sont effectivement comparées.

2.1.2 Appariements

Pour identifier l'intersection de deux objets A et B, les caractéristiques de l'objet A sont comparées à celles de l'objet B dans le but de trouver le plus grand nombre de caractéristiques communes aux deux objets. Celles-ci sont ensuite *appariées* (*i.e.* mises en correspondance) et l'intersection des objets est représentée par l'ensemble des caractéristiques appariées.

Définition (Appariement)
Etant donné deux ensembles de caractéristiques A et B, l'appariement d'une caractéristique $a \in A$ avec une caractéristique $b \in B$ est une relation $\pi : A \rightarrow B$ qui met en correspondance a avec b par un opérateur de comparaison. On dit que l'élément b est l'image de a par la relation π que l'on écrit $b = \pi(a)$.

Lorsque tous les appariements ont été trouvés, on note $App(A,B)$, la liste des appariements $\pi(a_i, b_j)$ entre les éléments a_i de A et b_j de B.

Exemple
Pour trouver l'intersection des deux ensembles d'entiers A et B, représentés sur la figure 2.1, on recherche, pour chaque entier de A, un entier dans B qui lui soit égal ; si on le trouve, les deux éléments égaux sont appariés. L'intersection de ces deux ensembles est alors le plus grand sous-ensemble de A (ou de B) contenant les éléments mis en correspondance. Dans le cas de données non structurées, cette intersection est triviale puisqu'elle repose sur l'égalité entre les éléments qui expriment toute la sémantique des objets.

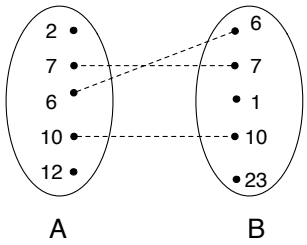


FIGURE 2.1 – Intersection de deux ensembles d'entiers naturels. Les lignes représentent les correspondances (ici, basées sur l'égalité) entre les éléments de l'ensemble A et ceux de l'ensemble B. L'intersection est le sous-ensemble $\{6, 7, 10\}$.

En logique du premier ordre, l'appariement des variables de deux clauses h et ex s'apparente à la notion de *substitution* de variables.

Définition (Substitution)
Une substitution est une application qui remplace des variables d'une clause h par des termes d'une clause ex. Une substitution θ est une liste de couples x_i/t_j où x_i est une variable de h et t_j un terme de ex. Le résultat de l'application de la substitution θ à la clause h, noté $h\theta$, est obtenu en remplaçant chaque occurrence des variables x_i par le terme t_j correspondant.

Dans la suite de ce document, pour avoir un langage commun entre la logique du premier ordre et les graphes, nous parlerons d'appariement au lieu de substitution qui reste un terme propre à la logique.

Exemple
Soit la clause h définie par $h \leftarrow p(x,y), q(y,z), r(z,x)$. L'application de la substitution $\theta = \{y/B; z/C\}$ à la clause h est la clause $h\theta \leftarrow p(x,B), q(B,C), r(C,x)$. Les variables qui ne sont pas substituées restent dans la nouvelle clause.

Cependant, le problème de l'appariement est plus complexe pour les objets structurés comme des clauses logiques puisqu'elles ne sont pas décrites par un seul ensemble d'éléments mais par deux : un ensemble de variables V et un ensemble de littéraux L. La difficulté vient du fait que l'intersection de ces clauses ne se réduit pas à l'intersection de leurs variables car il faut, en même temps, tenir compte de leurs littéraux (et vice-versa), comme l'illustre la figure 2.2.

Dans cet exemple, les variables sont représentées par les mêmes nombres entiers que ceux de l'exemple précédent (figure 2.1) mais ici, les littéraux empêchent de faire les mêmes appariements que précédemment. Par exemple, la variable 6 dans l'objet A est située à la position 1 du littéral p(6, 10) (c'est l'origine de l'arc vers le sommet 10) alors que c'est l'inverse dans l'objet B : les deux variables, bien qu'étant égales, ne peuvent pas être appariées car cela serait incohérent avec les littéraux des deux clauses.

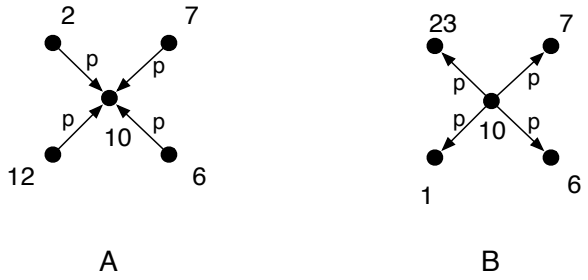


FIGURE 2.2 – Mise en correspondance de variables dans des clauses Datalog, représentées sous forme de graphe orienté.

2.1.3 Sous-Structure Commune

L'intersection entre deux objets structurés est également un objet structuré : l'intersection de deux clauses est une clause, l'intersection de deux graphes est un graphe. De manière générale, nous parlerons de Sous-Structure Commune (SSC) entre deux objets structurés pour désigner leur intersection. Dans le cas de la LPO, nous définissons la SSC entre deux clauses h et ex de la façon suivante. Pour une substitution θ donnée, la SSC entre h et ex est la sous-expression de $h\theta$ dans laquelle apparaissent les littéraux qui ne contiennent que des variables substituées.

Exemple
Soit les formules suivantes : $h \leftarrow p(x,y), q(x,y), p(y,z)$ et $ex \leftarrow q(A,B), p(B,C), r(B,C)$ et la substitution $\theta = \{x/A, y/B, z/C\}$. L'application de θ à h donne : $h\theta = p(A,B), q(A,B), p(B,C)$. En appliquant la définition de la SSC, les littéraux qui apparaissent dans la SSC sont $q(A,B)$ et $p(B,C)$. Le littéral $p(A,B)$ n'est pas retenu car il n'existe pas dans ex.

En logique du premier ordre, les SSC telles que nous les avons définies sont de nature asymétrique, c'est-à-dire que pour une substitution θ et deux clauses h et ex données telles que, $h\theta \subseteq ex$ et $ex\theta \subseteq h$, on a en général $h\theta \neq ex\theta$. Pour une construction symétrique, il faut se rapprocher de la notion d'unification de deux clauses h et ex qui consiste à trouver (quand elles existent) deux substitutions θ_1 et θ_2 telles que $h\theta_1 = ex\theta_2$. S'il existe une substitution θ telle que $h\theta = ex\theta$, alors θ est un unificateur pour les clauses h et ex.

La notion d'unificateur en logique est très similaire à la notion de SSC entre deux graphes A et B qui correspond à un sous-graphe partiel induit de A, isomorphe à un sous-graphe de B. Dans l'exemple de la figure 2.3, l'appariement entre les graphes G et G' est défini par $\pi = \{s_1/s'_1, s_2/s'_4, s_3/s'_3, s_4/s'_2\}$ et la SSC correspond aux sommets et aux arcs colorés en noir. L'arc (s_2, s_4) est conservé dans la SSC car elle apparaît également

entre les sommets des images s'_2 et s'_4. Par contre, comme il n'existe pas dans G' d'arc entre $s'_4 = \pi(s_2)$ et $s'_3 = \pi(s_3)$, l'arc (s_2, s_3) est exclu de la SSC.

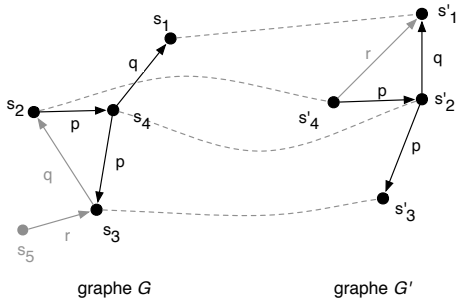


FIGURE 2.3 – La SSC entre deux graphes G et G' correspond à un sous-graphe partiel induit par l'appariement π des sommets de G avec ceux de G'. Les lignes en pointillés représentent les appariements entre les sommets des graphes G et G'.

Par extension de langage, on dira qu'un appariement *App* est connexe s'il engendre une SSC connexe.

2.2 Caractérisation des appariements

Selon les domaines d'application, la relation d'appariement π peut adopter des comportements différents. Par exemple, on ne manipulera pas de la même façon des données abstraites comme des variables logiques ou mathématiques et des données représentant des objets du monde réel.

Nous décrivons dans cette section quelques caractéristiques de la relation π qui permettent de s'adapter à des situations différentes : l'appariement des variables ou des littéraux, les appariements exacts ou inexacts, les appariements total et partiel, les appariements unique et multiple.

2.2.1 Appariement des variables ou des littéraux

Une clause Datalog est entièrement décrite par l'ensemble de ses variables et l'ensemble de ses littéraux. De la sorte, pour comparer deux clauses, l'appariement peut être dirigé par les variables (ce sont les variables qui sont appariées) ou par les littéraux (ce sont les littéraux qui sont appariés).

Lorsqu'un appariement est dirigé par les littéraux, seuls ceux qui ont la même signature peuvent être appariés, c'est-à-dire des littéraux construits sur le même symbole de

prédicat, ayant le même nombre de variables et de constantes et tels que les constantes situées à la même position soient de même type.

Cette dualité dans l'opération d'appariement permet d'utiliser l'une ou l'autre des approches pour tirer profit d'algorithmes plus adaptés à la recherche d'un appariement dirigé par les variables ou par les littéraux. Dans la suite de ce document, nous considérons que les appariements sont dirigés par les variables. Lorsque ça ne sera pas le cas, nous le préciserons explicitement.

2.2.2 Appariement exact ou inexact

Pour apparier les variables d'une clause h et d'une clause ex, on recherche, pour chaque variable dans h, s'il existe une variable dans ex avec laquelle elle puisse s'apparier. Pour savoir si une variable peut être appariée avec une autre, on utilise une fonction de comparaison f_c dont la nature définit deux types d'appariements : un *appariement exact* si la fonction f_c est le test d'égalité et un *appariement inexact* dans les autres cas.

Nous décrivons ci-dessous une approche générique pour comparer deux clauses avec un appariement exact. Comme la comparaison de deux clauses revient à comparer deux à deux les littéraux de même signature, nous distinguons le cas des littéraux ne contenant qu'une seule variable et celui des littéraux relationnels qui contiennent plusieurs variables.

2.2.2.1 Littéraux à une variable

Soit deux littéraux lit_v et lit'_v définis respectivement par $lit_v = p(v, C_1, \ldots, C_n)$ et $lit'_v = p(v', C'_1, \ldots, C'_n)$ qui définissent les propriétés des variables v et v' (représentées par les n constantes typées). La *fonction de comparaison* $fc_{Litvar} : L \times L' \rightarrow \{vrai, faux\}$, qui teste l'égalité des littéraux lit_v et lit'_v, est définie comme suit.

Si les littéraux n'ont pas de constantes (n=0), alors les variables v et v' sont vues comme identiques et par conséquent, les littéraux lit_v et lit'_v également.

Les littéraux qui possèdent n constantes (n>0) sont identiques si les constantes sont égales deux à deux (*i.e.* celles situées à la même position). Pour tester l'égalité des constantes, nous devons définir une fonction $fc_{const} : T \times T \rightarrow \{vrai, faux\}$ qui indique si les constantes C_i et C'_i de type T, situées à la position i, sont égales.

La fonction de comparaison des littéraux à une variable est donc définie comme suit :

$$fc_{Litvar}(lit_v, lit'_v) = f\Big(fc_{const}(C_i, C'_i)\Big) \tag{2.4}$$

Dans le cas d'un appariement exact, la fonction fc_{Litvar} peut être définie à l'aide de l'opérateur $\wedge$ (ET logique) entre les $fc_{const}(C_i, C'_i)$:

$$fc_{Litvar}(lit_v, lit'_v) = \bigwedge_{i=1}^{n} [C_i = C'_i] \tag{2.5}$$

2.2.2.2 Littéraux relationnels

Soit deux littéraux relationnels de la forme $lit_r = p(v_1, \ldots, v_m, C_1, \ldots, C_n)$ et $lit'_r = p(v'_1, \ldots, v'_m, C'_1, \ldots, C'_n)$. Avant de pouvoir les comparer, nous devons d'abord définir les deux fonctions suivantes :

- la fonction $fc_{var} : V \times V' \rightarrow \{vrai, faux\}$ qui teste l'égalité de chaque paire de variables v_i et v'_i situées à la position i ;
- la fonction $fc_{const} : T \times T \rightarrow \{vrai, faux\}$ (la même que précédemment) qui teste l'égalité de chaque paire de constantes C_j et C'_j à la position j.

La comparaison des deux littéraux lit_r et lit'_r est effectuée par la fonction $fc_{litR} : L_R \times L'_R \rightarrow \{vrai, faux\}$ qui dépend de la comparaison des variables et des constantes :

$$fc_{litR}(lit_r, lit'_r) = f_{i,j}\Big(fc_{var}(v_i, v'_i), fc_{const}(C_j, C'_j)\Big) \tag{2.6}$$

Comme précédemment, dans le cas d'un appariement exact où les comparaisons sont basées sur l'égalité, les fonctions fc_{Litvar} et fc_{litR} peuvent être définies à partir de l'opérateur $\wedge$ (ET logique) :

$$fc_{Litvar}(lit, lit') = \bigwedge_{i=1}^{m} [v_i = v'_i] \wedge \bigwedge_{j=1}^{n} [C_j = C'_j] \tag{2.7}$$

où $[C_i = C'_i]$ (resp. $[C_i, C'_i]$) renvoie la réponse au test d'égalité entre les deux arguments.

Si les constantes sont des valeurs réelles (par exemple, des mesures de grandeurs physiques) qui manquent de précision, il est difficile que deux valeurs soient strictement égales. Avec un appariement exact, le risque est alors qu'aucune variable ne soit appariée. Pour assouplir cette contrainte forte, d'autres fonctions de comparaisons comme une mesure de similarité entre les variables ou une mesure d'inclusion peuvent être utilisées à la place de l'opérateur d'égalité. On parle alors d'appariement inexact.

Dans ce cas, la manière de comparer les littéraux est identique aux appariements exacts à la différence que la réponse des fonctions de comparaison est maintenant une

valeur continue. Il convient alors de définir une valeur seuil pour décider si deux variables peuvent être appariées. Par exemple, dans le cas d'une mesure de similarité où la valeur 1 indique deux variables similaires, on pourra décider que deux variables sont appariables si leur similarité est supérieure à cette valeur seuil.

2.2.3 Appariement total ou partiel

L'appariement $App(h, ex)$ entre deux clauses h et ex est *total* si toutes les variables de h sont appariées et si tous les littéraux contenant des variables appariées apparaissent dans la clause ex. Autrement dit, un appariement est total si la clause h est identique à la SSC entre les clauses h et ex (voir section 2.1.3). Dans le cas contraire, on dit que l'appariement est *partiel*. La figure 2.4 montre des exemples d'appariements total et partiel.

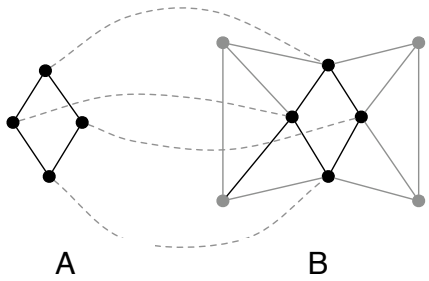


FIGURE 2.4 – Exemple d'appariement exact dirigé par les sommets entre deux graphes. Les traits en pointillés représentent les appariements entre les sommets. L'appariement $App(A, B)$ est total car tous les sommets de A sont appariés et que les arêtes entre les sommets appariés se retrouvent dans B. L'appariement $App(B, A)$ est partiel car certains sommets de B ne sont pas appariés.

Ces définitions font référence à des notions fondamentales en logique du premier ordre ainsi qu'en théorie des graphes où l'on distingue plusieurs cas de figure selon que les appariements $App(h, ex)$ et $App(ex, h)$ sont totaux ou partiels. Les tests classiques dans ces deux domaines sont identifiés dans le tableau 2.1.

$App(h, ex)$	Théorie des graphes	Logique du premier ordre	Type de test
total	isomorphisme de graphe		égalité
total	isomorphisme de sous-graphe	θ-subsomption	inclusion totale
partiel	isomorphisme de sous-graphe partiel		inclusion partielle, plus grande SSC

TABLEAU 2.1 – Correspondance en logique du premier ordre et en théorie des graphes des appariements total et partiel.

Ces notions sont définies ci-dessous.

Définition (θ-subsomption) [Plotkin, 1970]
Une clause h θ-subsume une clause ex (noté $h \leq ex$) si et seulement si il existe une substitution θ telle que $h\theta \subseteq ex$. Dans ce cas, tous les littéraux de $h\theta$ apparaissent également dans ex.

Le test de θ-subsomption permet de vérifier si l'appariement est total ou non et peut être vu comme un test d'inclusion en logique du premier ordre [4]. Si deux clauses h et ex se subsument l'une l'autre ($h\theta_1 \subseteq ex$ et $ex\theta_2 \subseteq h$), alors elles sont identiques.

Exemple
Etant donné la clause h définie par $h \leftarrow p(x,y), q(y,z), r(z,x)$ et les deux clauses ex_1 et ex_2 respectivement définies par : $ex_1 \leftarrow p(A,B), p(C,D), r(A,D), r(C,A), q(B,C)$ et $ex_2 \leftarrow p(A,B), p(C,D), r(A,D), q(B,C)$.
L'application de la substitution $\theta = \{x/A; y/B; z/C\}$ à la clause h donne
$h\theta \leftarrow p(A,B), q(B,C), r(C,A)$ qui vérifie $h\theta \subseteq ex_1$: h θ-subsume ex_1. Par contre, h ne θ-subsume pas ex_2 car le littéral $r(C,A)$ de $h\theta$ n'apparaît pas dans ex_2.
Les définitions relatives aux graphes sont données ci-dessous.

Définition (Isomorphisme de graphe)
Deux graphes $G = (S,A)$ et $G' = (S',A')$ sont isomorphes s'il existe une bijection $f : S \leftrightarrow S'$ telle que : pour tout couple de sommets $s_i \in S$ et $s_j \in S$, l'arête $(s_i, s_j) \in A$ si et seulement si l'arête $\big(f(s_i), f(s_j)\big) \in A'$. Autrement dit, en renommant correctement les sommets de G', on obtient $G' = G$.

4. Il introduit une notion syntaxique de généralité : si une clause h θ-subsume une clause ex alors c'est que h est au moins aussi générale que ex. La relation de subsomption est une relation d'ordre partiel entre les clauses : c'est un élément fondamental de la Programmation Logique et la Programmation Logique Inductive.

Deux graphes isomorphes peuvent être dessinés de la même façon en déplaçant simplement les sommets. Un exemple est illustré sur la figure 2.5. Il existe des algorithmes polynomiaux pour des cas spécifiques (arbres, graphes planaires) mais dans le cas général, la classe de complexité n'est pas tranchée.

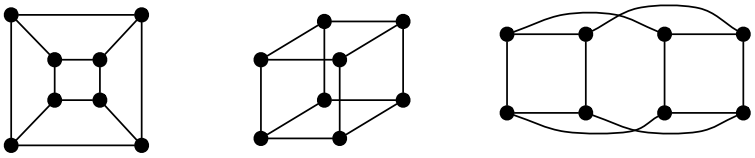


FIGURE 2.5 – Graphes isomorphes.

Le test d'*isomorphisme de sous-graphe* indique si un graphe G est totalement inclus dans un graphe G'.

Définition (Isomorphisme de sous-graphe)
Etant donné deux graphes $G = (S, A)$ et $G' = (S', A')$ tels que $|S| \leq |S'|$. Le graphe G est inclus dans le graphe G' s'il existe une fonction injective $f : S \rightarrow S'$ telle que $\forall s_i \in S$ et $s_j \in S, (s_i, s_j) \in A \Rightarrow \Big(f(s_i), f(s_j)\Big) \in A'$. Le graphe G est alors isomorphe à un sous-graphe de G' et il correspond à la SSC entre les deux graphes.

Ce problème est connu pour être NP-complet. Un exemple d'isomorphisme de sous-graphe est illustré sur la figure 2.6.

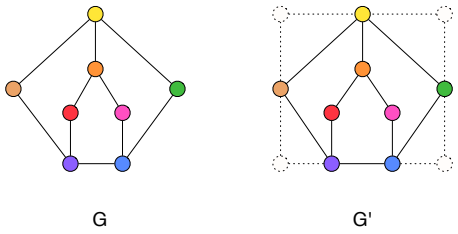


FIGURE 2.6 – Isomorphisme de sous-graphe : le graphe G est inclus dans le graphe G'.

L'identification du *plus grand sous-graphe commun* entre deux graphes donnés est relatif au test d'isomorphisme de sous-graphe partiel. Il s'agit de trouver les deux plus grands sous-graphes $G_1 \subseteq G$ et $G'_1 \subseteq G'$ tels que les graphes G_1 et G'_1 soient isomorphes.

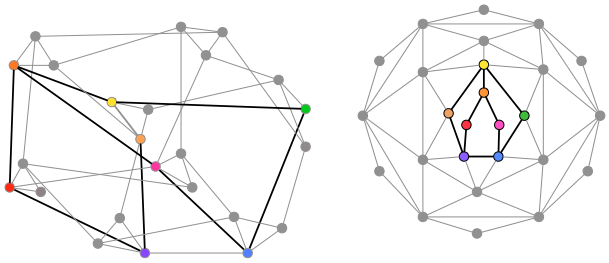


FIGURE 2.7 – Isomorphisme de sous-graphe partiel : le graphe à gauche est partiellement inclus dans le graphe à droite.

Ces notions sont particulièrement utiles car elles permettent de tester l'égalité ou l'inclusion entre deux objets structurés.

2.2.4 Appariement unique ou multiple

Dans les exemples précédents, nous avons considéré la relation d'appariement π comme la mise en correspondance d'une variable avec une seule autre variable. Un tel appariement, dit unique (ou de type 1-1), est donc une relation $\pi_{1-1} : V \rightarrow V'$ où une variable v de V ne peut être appariée qu'à une seule variable v' de V'. Il y a dans ce cas $min(|V|, |V'|)!$ appariements possibles. En logique du premier ordre, il correspond à l'hypothèse d'*identité d'objet* [Ferilli *et al.*, 2002] qui est plus pertinente dès lors que les variables s'identifient à des objets du monde réel. C'est également le type d'appariement que l'on trouve classiquement dans les graphes.

La taille des clauses (en termes de nombre de variables) détermine la possibilité ou non de trouver un appariement total : un appariement total n'est théoriquement possible que si $|V| \leq |V'|$. En théorie des graphes, les algorithmes d'isomorphisme de graphe utilisent d'ailleurs cette propriété pour détecter au plus tôt une réponse négative (*i.e.* les graphes ne sont pas isomorphes) avant de rechercher effectivement un isomorphisme.

On distingue trois autres types d'appariement appelés *appariements multiples*.

Un appariement de type 1-n est une relation $\pi_{1-n} : V \rightarrow V'^n$ où une variable d'une clause h peut être appariée avec n ($n \geq 2$) variables distinctes d'une clause ex. Dans les graphes, certains auteurs comme [Sorlin *et al.*, 2003], [Sorlin, 2006] autorisent ces appariements multiples comme dans le cas de la comparaison d'images représentées par des graphes : une partie d'une image A peut se retrouver en plusieurs endroits dans une image B.

Un appariement de type n-1 est une relation $\pi_{n-1} : V^n \rightarrow V'$ où n variables distinctes ($n \geq 2$) dans h peuvent être appariées avec une seule et même variable dans ex.

Enfin, un appariement de type n-n est une relation $\pi_{n-n} : V^n \rightarrow V'^n$ où n variables distinctes ($n \geq 2$) dans V peuvent être appariées avec n variables dans V'. C'est le cas pour certaines opérations en logique comme la généralisation la moins générale (Least General Generalization ou LGG) où il peut y avoir des substitutions multiples pour les deux clauses.

Pour ces appariements multiples, le nombre d'appariements possibles est égal à $|V'|^{|V|}$.

2.3 Test d'inclusion entre clauses

En logique du premier ordre, la θ-subsomption entre deux clauses h et ex peut être vue comme un test d'inclusion entre les clauses dont la réponse est booléenne : soit h subsume ex, soit elle ne la subsume pas.

Les algorithmes de θ-subsomption proposent des solutions pour réduire la complexité de ce problème qui est NP-complet. Nous en présentons quelques-uns dans cette section.

2.3.1 Approche par satisfaction de contraintes

[Prosser, 1996] a montré que le test de θ-subsomption de la logique du premier ordre est équivalent à un Problème de Satisfaction de Contraintes (PSC). Dans ce contexte, [Maloberti, 2005] utilise cette équivalence pour transformer un problème de θ-subsomption en un Problème de Satisfaction de Contraintes (PSC) et ainsi déléguer la recherche de la θ-subsomption à des procédures efficaces.

Définition (Problème de Satisfaction de Contraintes)
Un Problème de Satisfaction de Contraintes [Tsang, 1993] est défini par un triplet $< V, D, C >$ où :

- V est un ensemble de n variables $V = \{x_1, \ldots, x_n\}$;
- D est un ensemble de domaines (chaque domaine D_i associe un ensemble de valeurs admissibles pour la variable x_i) ;
- un ensemble de m contraintes spécifiant les relations entre les variables (les valeurs simultanément admissibles des variables). Chaque contrainte C_i implique un sous-ensemble des variables et spécifie les combinaisons possibles de valeurs pour ce sous-ensemble.

Un état du problème est défini par un appariement des variables avec les constantes. Un appariement qui ne viole aucune contrainte est dit cohérent. Un appariement total

est un appariement dans lequel chaque variable est associée à une valeur. Une solution au PSC est un appariement complet et cohérent. L'exemple ci-dessous illustre la transformation d'un problème de θ-subsomption en PSC.

Exemple
Soit les clauses $h \leftarrow p(x_0, x_1), q(x_0, x_2, x_3)$ et $ex \leftarrow p(a_0, a_1), p(a_1, a_2), q(a_0, a_2, a_3), q(a_0, a_1, a_3)$. Ce problème de θ-subsomption peut se réécrire en un PSC dont :
- l'ensemble des variables V est équivalent à l'ensemble des variables $\{x_0, x_1, x_2, x_3\}$;
- chaque domaine D_i est équivalent à l'ensemble des termes dans ex : $D_i = \{a_0, a_1, a_2, a_3\}$;
- chaque littéral lit_i dans h est transformé en une contrainte $Cont_i$.

La contrainte $Cont_1$, qui porte sur x_0 et x_1, est associée aux valeurs $\{\langle a_0, a_1\rangle, \langle a_1, a_2\rangle\}$ et la contrainte $Cont_2$, qui porte sur x_0, x_2 et x_3, est associée aux valeurs $\{\langle a_0, a_2, a_3\rangle, \langle a_0, a_1, a_3\rangle\}$.

Le programme Django [Maloberti and Sebag, 2004] intègre des procédures PSC standards comme la cohérence d'arc, le chaînage avant ainsi que des procédures et des structures de données spécifiques à la θ-subsomption. Comme il n'existe pas de procédure générale, efficace sur n'importe quel problème, les auteurs proposent un système appelé méta-Django qui sélectionne les meilleures procédures implémentées dans Django pour résoudre le problème posé. Cet algorithme a été testé sur des problèmes complexes (voir le chapitre 4 sur la transition de phase) et se révèle aujourd'hui, le test de θ-subsomption le plus rapide, basé sur des satisfactions de contraintes.

2.3.2 Réduction du nombre de candidats à l'appariement

La complexité de la recherche d'un appariement multiple entre deux clauses est due à la taille de l'espace de recherche. En effet, pour chaque variable d'une clause h, il faut en théorie tester l'appariement avec les m variables de la clause ex pour identifier celle qui semble la plus judicieuse pour conduire à la subsomption.

Pour diminuer la complexité du problème, on peut réduire, pour chaque variable (ou littéral) à apparier dans la clause h, le nombre de variables (ou littéraux) candidats à l'appariement dans la clause ex, c'est-à-dire ceux qui sont appariables avec les variables (ou littéraux) de la clause h. L'idée est de ne sélectionner et tester que les variables ou les littéraux similaires (voir la section 2.2.2 sur les appariements exacts).

[Wysotzki *et al.*, 1981] a proposé une telle approche pour le problème d'isomorphisme de graphe. Cette idée a été reprise par [Scheffer *et al.*, 1996] dans le cadre du test de

subsomption en cherchant à réduire le nombre de littéraux candidats ; il se base pour cela sur le *graphe dual* d'une clause et le *contexte* d'un littéral.

Définition (Graphe dual)
Le graphe $G_d = (S, A)$ est le graphe dual (Scheffer l'appelle le graphe de littéraux) d'une clause h où chaque sommet dans S est un littéral lit_i de h et l'ensemble A composé des arêtes (lit_i, lit_j) si et seulement si il y a une variable à la fois dans lit_i et dans lit_j.

Un exemple de graphes duaux pour les clauses $h = p(x_1, x_2), p(x_2, x_3), p(x_1, x_3)$ et $ex = p(x'_1, x'_2), p(x'_4, x'_1), p(x'_1, x'_3), p(x'_2, x'_3), p(x'_3, x'_4)$ est illustré sur la figure 2.8.

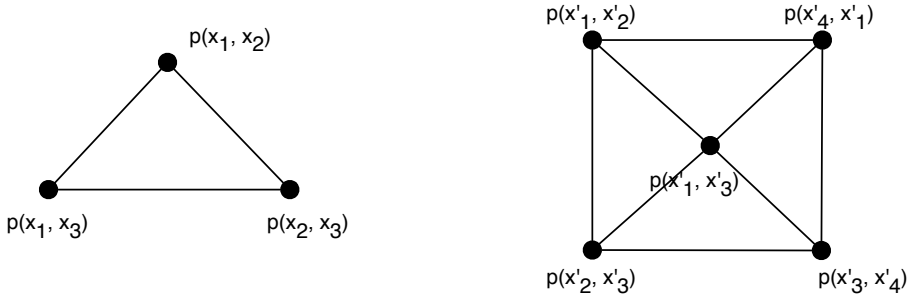


FIGURE 2.8 – Exemples de graphes duaux (graphes de littéraux).

Définition (Contexte d'un littéral)
Le contexte de profondeur d d'un littéral $lit \in h$, noté $con_{lit}(lit, h, d)$, est l'ensemble de tous les chemins de longueur d ayant pour origine le littéral lit dans le graphe dual de h. La taille du contexte croît exponentiellement avec d mais reste bornée par la taille de la clause h.

Dans l'algorithme LITCON, [Scheffer *et al.*, 1996] utilise le contexte des littéraux pour réduire le nombre de candidats à l'appariement de la manière suivante : un littéral lit dans h ne peut être apparié avec un littéral lit' dans ex que si le contexte de lit est inclus dans le contexte de lit' : $con_{lit}(lit, h, d) \subseteq con_{lit}(lit', h, d)$.

Reprenons l'exemple des graphes duaux de la figure 2.8 où l'on cherche à apparier le littéral $p(x_1, x_2)$ appartenant à la clause h. Sans réduction, il faudrait tester l'appariement avec les 5 littéraux de la clause ex. On construit le contexte du littéral $con_{lit}(p(x_1, x_2), h, 1) = \{p(x_1, x_3), p(x_2, x_3)\}$ que l'on compare au contexte des littéraux de ex :

$con_{lit}(p(x'_1, x'_2), ex, 1) = \{p(x'_4, x'_1), p(x'_1, x'_3), p(x'_2, x'_3)\}$
$con_{lit}(p(x'_1, x'_3), ex, 1) = \{p(x'_1, x'_2), p(x'_4, x'_1), p(x'_3, x'_4), p(x'_2, x'_3)\}$
$con_{lit}(p(x'_2, x'_3), ex, 1) = \{p(x'_1, x'_2), p(x'_1, x'_3), p(x'_3, x'_4)\}$

$con_{lit}(p(x'_4, x'_1), ex, 1) = \{p(x'_1, x'_2), p(x'_1, x'_3), p(x'_3, x'_4)\}$
$con_{lit}(p(x'_3, x'_4), ex, 1) = \{p(x'_2, x'_3), p(x'_1, x'_3), p(x'_4, x'_1)\}$

Le seul contexte de littéral dans *ex* qui contienne le contexte $con_{lit}(p(x_1, x_2), h, 1)$ est $con_{lit}(p(x'_1, x'_2), ex, 1)$. Ainsi, il n'y a plus qu'un seul littéral candidat pour l'appariement : le littéral $p(x'_1, x'_2)$.

Dans de nombreux cas, lorsque le graphe de littéral est développé à une profondeur d, les contextes sont suffisamment grands pour que les appariements soient uniques. Les tests d'appariement peuvent alors être réalisés de façon déterministe, c'est-à-dire que la liste des candidats à l'appariement ne contient qu'un seul littéral. La contrepartie est que la construction du graphe dual est exponentielle en d.

Cependant, cet algorithme souffre du passage à l'échelle pour de grandes valeurs de d [Karabaev *et al.*, 2006]. Une approche similaire, nommée OBJCON, est proposée par [Hölldobler and Skvortsova, 2006] où les appariements sont dirigés par les variables et non plus par les littéraux ; la réduction du nombre de candidats étant basée sur le contexte lié aux variables. D'autres approches utilisent la notion de contexte comme [Zampelli *et al.*, 2007].

Cette approche est intéressante car elle permet d'identifier rapidement les appariements qui conduisent à des échecs. Cependant, le concept de chemin dans un graphe ne concerne que les relations binaires (des graphes ou des littéraux à deux variables) et la notion de contexte, telle qu'elle est définie ici, semble être difficilement généralisable à des relations n-aires.

2.4 Similarité entre clauses logiques

Nous nous intéressons maintenant à l'évaluation de la similarité entre de deux clauses logiques qui renvoie une réponse à valeur continue. Pour répondre à cette problématique, [Landwehr *et al.*, 2006] propositionnalise les structures qui sont alors représentées par un ensemble de descripteurs pour définir une mesure de similarité entre ces ensembles de descripteurs ; ce type d'approche est souvent accompagnée d'une perte d'information structurale.

Un autre type d'approche, que nous allons détailler plus profondément, consiste à comparer directement les structures ; c'est ce que propose [Bisson, 1992] en recherchant le meilleur appariement entre les littéraux de deux clauses. Le principe de la méthode est identique aux approches présentées dans la section précédente à savoir la réduction du nombre de candidats à l'appariement. Ici, l'algorithme cherche à apparier les littéraux

contenant les variables les plus similaires en se basant sur l'hypothèse suivante. Deux variables sont d'autant plus similaires qu'elles partagent des propriétés communes et qu'elles apparaissent dans des contextes similaires, c'est-à-dire qu'elles apparaissent, à la même position, dans des littéraux basés sur les mêmes symboles de prédicats et que leurs variables voisines sont elles-mêmes similaires entre elles.

La méthode se déroule en deux étapes :
- la construction d'une matrice SIM de similarité locale entre chaque paire de variables (v_i, v'_m) appartenant respectivement à h et ex ;
- la recherche d'un appariement optimal entre les variables des deux clauses, en se guidant sur les valeurs contenues dans la matrice SIM.

La méthode est décrite ici en ne considérant que des littéraux commutatifs mais elle s'adapte naturellement aux littéraux non commutatifs.

2.4.1 Evaluation des similarités locales

Le problème est ici d'évaluer la matrice SIM dont les éléments correspondent à la similarité entre les variables v_i dans V (l'ensemble des variables de h) et v'_m dans V' (l'ensemble des variables de ex). Avant de détailler l'algorithme, nous introduisons quelques fonctions nécessaires à la comparaison des littéraux dans les clauses.

Nous nous plaçons ici dans le cas d'appariements inexacts où les fonctions de comparaison renvoient une mesure de similarité. Comme précédemment, nous distinguons les littéraux contenant une seule variable et les littéraux relationnels contenant deux variables.

2.4.1.1 Comparaison de littéraux à une variable

Chaque variable est caractérisée par différentes propriétés, représentées par les constantes typées, dans les littéraux de la forme : $p(v, C_1, \ldots, C_n)$ et $p(v', C'_1, \ldots, C'_n)$ (voir section 1.3). Nous considérons qu'il existe une fonction générique $sim_{const} : T \times T \rightarrow [0,1]$ qui définit la similarité entre les constantes C_i et C'_i de type T.

La similarité entre deux variables $v \in V$ et $v' \in V'$, apparaissant respectivement dans les littéraux lit_v et lit'_v, d'un point de vue local, est évaluée par la fonction $\mathrm{Sim}_{var} : L \times L' \rightarrow [0,1]$, définie comme la moyenne des valeurs de sim_{const} pour toutes les constantes :

$$\mathrm{Sim}_{var}(lit_v, lit'_v) = \frac{1}{n} \sum_{k=1}^{n} sim_{const}(C_k, C'_k) \tag{2.8}$$

2.4.1.2 Comparaison de littéraux relationnels

Avant de définir la mesure de similarité entre deux littéraux relationnels, il est nécessaire de définir les deux fonctions suivantes :

- la fonction Link_Of : V → L renvoie, pour chaque variable v_i d'une clause, l'ensemble $\{lit_{r1}, ..., lit_{rp}\}$ des littéraux dans lesquels elle apparaît ;
- la fonction Voisins : $V \times L \rightarrow V$ est définie pour des littéraux binaires (c'est-à-dire contenant deux variables). Elle renvoie, pour une variable v_i et un littéral lit_{rk} donnés, la référence sur une variable voisine v_j (v_i et v_j apparaissent dans le même littéral lit_{rk}).

Soit les deux littéraux relationnels $lit_r = p(v_1, v_2, C_1, \ldots, C_n)$ et $lit_r{}' = p(v'_1, v'_2, C'_1, \ldots, C'_n)$. De la même façon que pour les variables, nous considérons qu'il existe une fonction générique Sim_{lit} : $\text{SIM} \times L \times L' \times V \times V' \rightarrow [0,1]$ qui évalue la similarité de deux littéraux du point de vue de deux variables données, en se basant sur les valeurs de la matrice SIM qui contient la valeur de similarité entre chaque paire de variables. Cette fonction s'écrit :

$$\text{Sim}_{lit}\,(\text{SIM}, lit_r, lit'_r, v_i, v'_m) = \frac{1}{2}\left(1 + \text{SIM}\Big[Voisins(lit_r, v_i), Voisins(lit_r{}', v'_m)\Big].\frac{1}{n}\sum_{k=1}^{n} sim_{const}(C_k, C'_k)\right) \quad (2.9)$$

La fonction Sim_{lit} prend en compte la ressemblance entre les propriétés des deux littéraux. Le terme SIM donne une définition récursive de la similarité locale entre variables et permet ainsi de prendre en compte la structure des clauses. La figure 2.9 résume l'ensemble des informations qui sont prises en compte lorsque l'on compare deux littéraux.

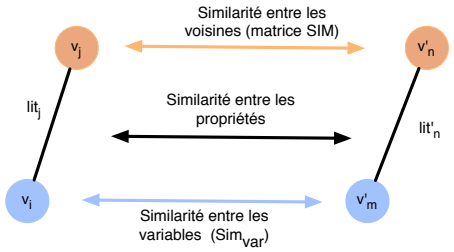


FIGURE 2.9 – Similarité entre littéraux relationnels du point de vue des variables v_i et v'_m.

2.4.1.3 Calcul de la matrice de similarité

Nous pouvons à présent décrire le calcul de la matrice SIM. Dans le cas de prédicats binaires, l'algorithme Similarite_Locale est le suivant.

Procédure Similarite_Locale
Input : clause h, clause ex, entier max_{Iter}
B_1 **pour** k=*1* **to** max_{Iter} **faire**
B_2 **pour chaque** *couple de variables* (v_i, v'_m) *respectivement dans h et ex* **faire**
L ← Link_Of (v_i) ;
L' ← Link_Of (v'_m) ;
1 Nsim ← Meilleur_Appariement_Litteraux $\left(\text{SIM}, L, L', v_i, v'_m\right)$;
2 $\text{SIM}[v_i, v'_m] \leftarrow \left[\text{Sim}_{var}(v_i, v'_m) + Nsim/Max(|L|, |L'|)\right]/2$;
finprch
finpour
Retourner SIM;

Algorithme 1 : Calcul de la matrice SIM.

La boucle B_2 effectue le calcul de similarité locale entre les variables v_i et v'_m. Le problème central est de trouver la manière d'apparier, de façon optimale, les littéraux de l'ensemble L avec ceux de l'ensemble L'. Comme dans l'exemple de la figure 2.10, tous les littéraux sont construits sur le symbole de prédicat p et ne contiennent que deux variables. De plus, il n'y a qu'un seul littéral pour une paire de variables donnée, ce qui simplifie grandement le processus d'appariement.

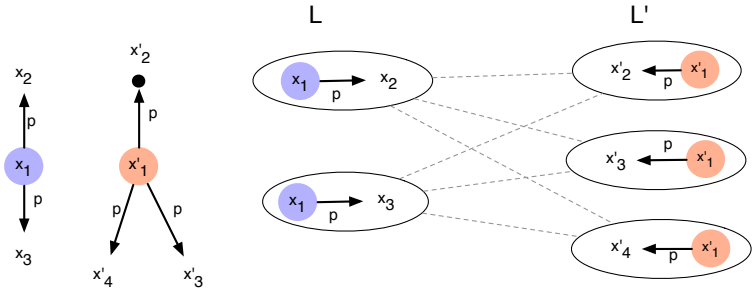


FIGURE 2.10 – La fonction Meilleur_Appariement_Litteraux recherche le meilleur appariement entre les littéraux des ensembles L=Link_Of(x_1) et L'=Link_Of(x'_1).

Ce problème d'appariement classique est géré par la procédure Meilleur_Appariement_Litteraux. En effet, considérons les littéraux contenus dans les listes L et L' comme les sommets d'un graphe bipartite dont on connaît, pour chaque couple (lit_{rj}, lit'_{rn}), la valeur de ressemblance grâce à la fonction Sim_{lit}. Trouver le meilleur appariement à effectuer entre ces éléments équivaut à maximiser la somme des Sim_{lit} et donc à résoudre un *problème de couplage de poids maximum dans un graphe bipartite* (assignment problem). L'algorithme hongrois [Kuhn, 1955] est utilisé pour cette tâche.

La méthode proposée par [Bisson, 1992] est cependant plus générale et est utilisable lorsque les prédicats sont des relations n-aires et qu'il peut y avoir plusieurs littéraux contenant les mêmes variables. Dans ce cas, le meilleur appariement est recherché entre littéraux de même signature. La fonction Meilleur_Appariement_Litteraux recherche le meilleur appariement pour chaque sous-ensemble $L(s_i)$ de littéraux de même signature s_i ce qui conduit à l'évaluation d'une variable Nsim_{s_i}. La variable Nsim est la somme maximale de tous les $L(s_i)$.

Finalement, le calcul de la similarité locale $\text{SIM}[v_i, v'_m]$ entre deux variables correspond à la moyenne entre leur ressemblance (Sim_{var}) et la ressemblance normalisée de leur voisinage ($\text{Nsim}/\max(|L|, |L'|)$), ligne 2 de l'algorithme.

Puisque la similarité de deux variables v_i et v'_m est fonction de celle de ses voisines et réciproquement, les éléments de SIM peuvent s'écrire sous forme d'un système d'équations. La résolution de ce système s'effectue à l'aide d'une méthode itérative (Jacobi), la boucle B_1 permettant de calculer les valeurs successives de la matrice. Ces itérations ont pour effet de propager les informations de similarités locales : le nombre d'itérations caractérise la profondeur de diffusion des informations, c'est-à-dire la taille du voisinage pris en compte pour comparer deux variables.

On retrouve cette idée générale dans de nombreux travaux et domaines d'applications, comme par exemple :

- dans [Fröhlich *et al.*, 2005a], [Rupp *et al.*, 2007] pour la comparaison de graphes moléculaires (voir section 5.4.2) ;
- dans [Scheffer *et al.*, 1996], [Hölldobler and Skvortsova, 2006] (voir section 2.3.2) pour la comparaison de clauses logiques ;
- dans [Singh *et al.*, 2008] pour la comparaison de réseaux d'interaction protéine-protéine.

2.4.2 Appariement et similarité globale

Une fois que les valeurs de similarité locales ont été calculées dans la matrice SIM, l'algorithme cherche le meilleur appariement entre les variables de h et les variables de ex, c'est-à-dire celui qui maximise la similarité globale entre les clauses, notée SIM (h, ex).

Cette fonction de similarité est proportionnelle à la somme des littéraux communs (c'est-à-dire la somme des Lsim) des variables appariées, le tout divisé par la taille de h et la taille de ex en termes de nombre de littéraux :

$$SIM(h,ex) = \frac{\sum_{v_i \in V_{apparies}} \text{SIM}[v_i, \pi(v_i)]}{max\left(\sum_{v_k \in V} \left|L(v_k)\right|, \sum_{v'_p \in V'} \left|L'(v'_p)\right|\right)} \tag{2.10}$$

où :

- V (resp. $V_{apparies}$) est l'ensemble des variables (resp. appariées) de l'hypothèse h ;
- $\pi(v_i)$ la variable dans ex qui a été appariée avec v_i dans h par l'appariement π.

La recherche du meilleur appariement est effectuée au moyen d'un algorithme glouton : les variables sont appariées suivant l'ordre décroissant des valeurs de similarité dans la matrice SIM. A chaque itération, on recherche la plus grande valeur dans la matrice SIM dans les lignes qui correspondent aux variables de h non encore appariées et les variables correspondantes sont appariées.

L'utilisation d'une matrice de similarité entre les variables permet de réduire le nombre de candidats à l'appariement et guide la recherche du meilleur appariement. On retrouve la même approche dans [Sorlin and Solnon, 2004].

2.5 Degré d'inclusion entre clauses

En logique du premier ordre, les opérations de comparaisons entre clauses sont des tests booléens mais il n'y a pas pléthore d'approches pour évaluer un degré d'inclusion entre clauses. Nous avons vu que la comparaison entre deux clauses nécessite de trouver la plus grande substitution entre les clauses h et ex puis de comparer la taille de cette intersection par rapport à la taille de la clause h. Cette intersection correspond à une généralisation entre les deux clauses.

L'une des méthodes de généralisation les plus étudiées en PLI est la lgg (Least General Generalization), définie par [Plotkin, 1970] qui permet de construire la généralisation la plus spécifique qui θ-subsume les deux clauses.

Il est possible de déterminer le degré d'inclusion entre les clauses h et ex en faisant le rapport entre la taille de la lgg et la taille de la clause h. En effet, si h est très peu semblable à ex, la généralisation devrait être petite comparativement à h. Cette idée ne fonctionne pas pour deux raisons : le problème des littéraux redondants (c'est le principe même de la lgg) et même après réduction, on ne peut accepter que les appariements de type n-1 et non ceux de type n-n pour ne pas se retrouver avec une conjonction de clauses mélangeant différentes "interprétations" et qui font que $|lgg| > |h|$. L'idée de comparer les tailles de la lgg et de h est intéressante mais pas du tout dans le cas de la lgg.

Une autre approche pourrait consister à rechercher la plus grande substitution possible en énumérant directement l'ensemble des substitutions. Cette approche pourrait, par exemple, s'inspirer du travail effectué dans LOGAN-H [Arias *et al.*, 2005], un système d'apprentissage de type "bottom-up", qui a la particularité de pouvoir générer toutes les substitutions possibles entre une hypothèse h et un exemple ex. Toutefois, le nombre de solutions peut être potentiellement très grand puisqu'il correspond à la taille du treillis de généralisation. Dès lors, sans critère de sélection, il semble difficile de garantir qu'une telle recherche puisse fournir un résultat en temps raisonnable dans toutes les situations.

Finalement, peu d'approches s'attaquent au problème de l'évaluation d'un degré d'inclusion à valeur continue entre clauses. Les méthodes décrites dans ce chapitre se rapprochent de la problématique posée sans toutefois y répondre entièrement.

Chapitre 3

Evaluation d'un degré d'inclusion entre clauses

Les algorithmes d'appariement présentés dans le chapitre précédent ne répondent pas exactement à la problématique de la mesure d'un degré d'inclusion entre deux clauses. En effet, à l'exception de [Bisson, 1992] qui peut quantifier la ressemblance, ces approches testent l'inclusion totale d'une clause *h* dans une clause *ex* et renvoient une réponse booléenne. D'autre part, les algorithmes qui manipulent des graphes sont souvent limités aux graphes simples non orientés et ne semblent pas facilement modifiables pour traiter des structures plus générales comme les multi-graphes ou les hypergraphes.

Le point commun à ces algorithmes est qu'ils cherchent à réduire, d'une manière ou d'une autre, la complexité du problème en limitant le nombre de candidats à l'appariement, c'est-à-dire le nombre de variables dans la clause *h* qui peuvent être appariées à une variable donnée dans la clause *ex*.

Dans le même esprit, nous abordons le problème de la couverture partielle en nous basant sur le travail de [Bisson, 1995] qui présente l'avantage d'être générique et de s'appliquer à des relations *n*-aires (clauses logiques quelconques et hypergraphes). Enfin, il apparaît assez simple de modifier l'algorithme pour qu'il évalue un degré d'inclusion, à valeur dans l'intervalle[0, 1], entre deux clauses.

La méthode repose sur deux étapes : l'évaluation d'un degré d'inclusion entre les variables de deux clauses et l'appariement global entre ces variables. Dans la section 3.1, nous décrivons une approche simple pour évaluer un degré d'inclusion entre deux clauses Datalog. Les paramètres du calcul des degrés d'inclusion entre variables sont détaillés dans la section 3.2, ainsi que les modifications apportées pour corriger certains points gênants de l'algorithme. Dans la section 3.3, plusieurs stratégies d'appariement global sont proposées. Une étude de la complexité des différents algorithmes proposés est présentée dans la section 3.5.

3.1 Première approche de l'inclusion

Le degré d'inclusion entre deux clauses h et ex, noté $Di(h, ex)$, peut être évalué en comparant la taille de la SSC induite par l'appariement *App* de leurs variables à la taille de la clause source (voir équation 2.3 page 16). Il prend ses valeurs dans l'intervalle $[0, 1]$. Les propriétés du degré d'inclusion dérivent directement de l'inclusion ensembliste :

- si $Di(h,ex) = 1$, c'est que l'appariement $App(h, ex)$ est total. Dans ce cas, si h et ex sont des formules logiques, alors h *App*-subsume ex ; si ce sont des graphes, h est un sous-graphe de ex ;
- si $App(h, ex) = \oslash$ (aucune SSC n'a été identifiée), alors $Di(h, ex) = Di(ex, h) = 0$;
- si $h \subseteq ex$ et $ex \subseteq h$, alors $Di(h, ex) = Di(ex, h) = 1$: les objets sont identiques.

Pour passer de la notion de similarité à celle d'inclusion, il est nécessaire d'introduire la notion de *clause source* et de *clause de référence* dans les équations et les algorithmes. Ainsi, chaque fois qu'une mesure de similarité est calculée, il convient de la transformer en degré d'inclusion, ce qui est réalisé par la redéfinition des dénominateurs dans les équations de la section 2.4 du chapitre 2.

3.1.1 Degré d'inclusion local entre variables

Soit deux littéraux lit_v et lit'_v de la forme $p(v_i, C_1, \ldots, C_n)$ et $p(v'_m, C'_1, \ldots, C'_n)$ ne contenant qu'une seule variable et où les constantes C_k (resp. C'_k) correspondent aux caractéristiques de la variable v_i (resp. v'_m). La fonction Di_{var} définit le degré d'inclusion entre les littéraux lit_v et lit'_v comme la moyenne des degrés d'inclusion des caractéristiques de leur variable respective. Elle est dérivée de la fonction Sim_{var} (équation 2.8) dans laquelle la fonction sim_{const} est remplacée par di_{const} :

$$Di_{var}(lit_v, lit'_v) = \begin{cases} \frac{1}{n} \sum_{k=1}^{n} di_{const}(C_k, C'_k) & \text{si } n \neq 0 \\ 1 & \text{sinon} \end{cases} \tag{3.1}$$

où n est le nombre de constantes dans les littéraux lit_v et lit'_v. La fonction $di_{const} : T \times T \rightarrow [0, 1]$ quantifie l'inclusion entre les constantes de type T qui apparaissent dans les littéraux. Lorsque les constantes C_k et C'_k représentent des ensembles (*e.g.* des intervalles, des ensembles de valeurs nominales), l'inclusion est identique à l'inclusion ensembliste : la taille de l'intersection $|C_k \cap C'_k|$ normalisée par $|C_k|$ pour se ramener à une valeur comprise entre 0 et 1. Dans le cas de constantes distinctes (entiers, réels, valeurs symboliques), une valeur d'inclusion ne peut être définie ; la valeur renvoyée par di_{const} est alors l'égalité entre les constantes.

Exemple
Soit les clauses $h \leftarrow p(x, 3, \{4, 7, 9\})$ et $ex \leftarrow p(y, 7, \{4, 9, 20\})$. Les premières constantes sont des entiers : le résultat de leur intersection est donné par l'égalité : $di_{const}(3, 7) = 0$. Les constantes au rang 3 sont des ensembles d'entiers, l'inclusion est donnée par $di_{const}(\{4, 7, 9\}, \{4, 9, 20\}) = 2/3$. Le degré d'inclusion entre les deux variables est donc égal à $Di_{var}(x, y) = (0 + 2/3)/2 = 0,33$.

Nous considérons à présent les littéraux contenant deux variables et m constantes qui définissent une relation entre ces variables : $lit_r = p(v_i, v_j, C_1, \ldots, C_m)$ dans la clause h et $lit'_r = p(v'_m, v'_n, C'_1, \ldots, C'_m)$ dans la clause ex.
Les fonctions Link_Of et Voisins ne sont pas modifiées. Nous rappelons que :

- la fonction Link_Of : $V \rightarrow L$ renvoie, pour une variable v_i, l'ensemble $\{lit_{r1}, ..., lit_{rp}\}$ des littéraux dans lesquels v_i apparaît ;
- la fonction Voisins : $V \times L \rightarrow V$ renvoie, pour une variable v_i et un littéral lit_{rk} donnés , la référence sur une variable voisine v_j (v_i et v_j apparaissent dans le même littéral lit_{rk}).

Le degré d'inclusion Di_{lit} entre les littéraux lit_r et lit'_r, du point de vue de deux variables v_i et v'_m situées à la même position, dérive de la fonction de similarité Sim_{lit} entre littéraux (équation 2.9), où sim_{const} est remplacée par di_{const}. Il s'écrit comme une fonction f du degré d'inclusion entre les variables voisines de v_i et v'_m et du degré d'inclusion entre leurs constantes [5] :

$$Di_{lit}(\text{INC}, lit_r, lit'_r, v_i, v'_m) = f\left(\text{INC}\Big[Voisins(lit_r, v_i), Voisins(lit_r{}', v'_m)\Big], \frac{1}{m}\sum_{k=1} di_{const}(C_k, C'_k)\right) \quad (3.2)$$

Le choix de la fonction f dépend du domaine d'application. On pourra par exemple choisir de faire la moyenne des deux arguments de la fonction f (équation 3.3) ou leur produit. L'intérêt de considérer la moyenne est que cela évite d'obtenir trop de valeurs nulles dans la matrice (élément absorbant du produit). En effet, ces valeurs correspondraient à des variables totalement différentes qui ne pourraient pas être comparées ce qui bloquerait la propagation des informations lors du calcul de la matrice.

$$Di_{lit}(\text{INC}, lit_r, lit'_r, v_i, v'_m) = \frac{1}{2}\left(\text{INC}\Big[Voisins(lit_r, v_i), Voisins(lit_r{}', v'_m)\Big] + \frac{1}{m}\sum_{k=1} di_{const}(C_k, C'_k)\right) \quad (3.3)$$

5. C'est-à-dire une fonction de la ressemblance entre les atomes voisins et la ressemblance entre les propriétés des relations entre les deux paires d'atomes (v_i, v_j) et (x'_m, v'_n).

La matrice appelée INC, qui remplace la matrice SIM, contient les degrés d'inclusion de chaque paire de variables (v_i, v'_m) dans les clauses *h* et *ex*.

Le calcul des éléments de la matrice INC est basé sur l'algorithme 1 (voir section 2.4.1.3) auquel une modification est apportée. La valeur Nsim du meilleur appariement entre les littéraux de deux variables v_i et v'$_m$, calculée par la fonction Meilleur_Appariement_Litteraux, doit refléter maintenant l'asymétrie de l'inclusion.

Cette procédure reste inchangée (car elle renvoie une valeur absolue et non relative) mais la valeur Nsim doit être normalisée par le cardinal $\left|L(v_i)\right|$ de l'ensemble *L* (nombre de littéraux dans lesquels la variable v_i apparaît) dans la clause *h* qui est la clause source. L'équation de mise à jour du degré d'inclusion entre deux variables v_i et v'_m (ligne 2 de l'algorithme 1) devient donc :

$$\mathrm{INC}[v_i, v'_m] = \frac{1}{2}\left(Di_{var}(v_i, v'_m) + \frac{Nsim}{\left|L(v_i)\right|}\right) \tag{3.4}$$

L'algorithme d'évaluation de la matrice INC, appelé Calcul_Matrice_Inclusion, est donc le suivant.

Procédure Calcul_Matrice_Inclusion
Input : clause *h*, clause *ex*, entier Iter
pour *k=1* **to** *Iter* **faire**
  pour chaque *paire de variables (v$_i$, v'$_m$) respectivement dans h et ex* **faire**
    L(v$_i$) = Link_Of (v$_i$) ;
    L'(v'$_m$) = Link_Of (v'$_m$) ;
    Nsim = Meilleur_Appariement_Litteraux (INC, L(v$_i$), L'(v'$_m$), v$_i$, v'$_m$);
    INC [v$_i$, v'$_m$] = (Di_{var} (v$_i$, v'$_m$) + Nsim / | L(v$_i$) |) / 2;
  finprch
finpour
Retourner INC;

Algorithme 2 : Calcul de la matrice INC.

Pour une variable donnée, la propagation des valeurs d'inclusion suit ici une loi de décroissance en $2^n/V$ où *n* est le nombre d'itérations et *V* le nombre de variables voisines. Donc, au fur et à mesure des itérations, les valeurs de INC sont supposées converger vers un point fixe (la preuve de la convergence et la valeur du point fixe restent toutefois à démontrer).

3.1.2 Inclusion globale entre les clauses

L'étape suivante consiste à apparier les variables de la clause h avec celles de la clause ex. Nous ne nous intéressons pas ici à l'algorithme d'appariement qui sera discuté dans la section 3.3. Etant donné un appariement π entre les variables des clauses h et ex, le degré d'inclusion est évalué par la fonction $I_\pi(h, ex)$ [6] qui résulte d'une modification de l'équation 2.10. Comme précédemment, la différence se trouve dans le dénominateur qui ne fait référence ici qu'à la clause h :

$$I_\pi(h, ex) = \frac{\sum_{v_i \in V_{apparies}} \text{INC}\Big[v_i, \pi(v_i)\Big].\Big|L(v_i)\Big|}{\sum_{v_k \in V} \Big|L(v_k)\Big|} \tag{3.5}$$

où V (resp. $V_{apparies}$) est l'ensemble des variables (resp. appariées) de la clause h.

Comme la fonction I_π est évaluée à partir de la matrice INC,

3.1.3 Exemple

L'exemple ci-dessous illustre le fonctionnement de cet algorithme. Soit les clauses h et ex définies par $h \leftarrow r(x_1, x_2), p(x_2, x_3)$ et $ex \leftarrow p(a_2, a_1), r(a_1, a_2), q(a_2, a_3), r(a_3, a_4)$, représentées sur la figure 3.1.

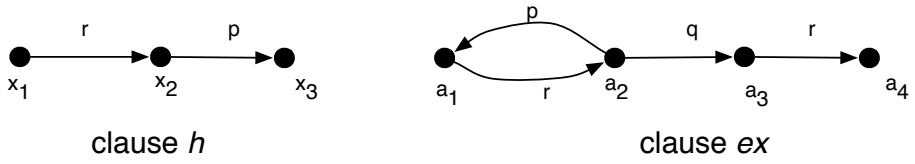


FIGURE 3.1 – Représentation graphique des clauses h et ex.

Les valeurs de la matrice $\text{INC}^{(k)}$, à l'itération k, pour les variables appariables sont définies par les équations ci-dessous. Le coefficient multiplicateur à gauche de chacune correspond à la normalisation par le nombre de littéraux dans $L(x_i)$:

$$\begin{aligned}
\text{INC}^{(k)}[x_1, a_1] &= 1/1 \times \Big[(1 + \text{INC}^{(k-1)}[x_2, a_2])/2\Big] \\
\text{INC}^{(k)}[x_1, a_3] &= 1/1 \times \Big[(1 + \text{INC}^{(k-1)}[x_2, a_4])/2\Big] \\
\text{INC}^{(k)}[x_2, a_2] &= 1/2 \times \Big[(1 + \text{INC}^{(k-1)}[x_1, a_1])/2 + (1 + \text{INC}_{k-1}[x_3, a_1])/2\Big] \\
\text{INC}^{(k)}[x_2, a_4] &= 1/2 \times \Big[(1 + \text{INC}^{(k-1)}[x_1, a_3])/2\Big] \\
\text{INC}^{(k)}[x_3, a_1] &= 1/1 \times \Big[(1 + \text{INC}^{(k-1)}[x_2, a_2])/2\Big]
\end{aligned}$$

6. Le nom de la fonction vient de Inclusion partielle où π tient pour "partielle".

Dans cet exemple, les éléments de la matrice INC ont été initialisés à 1 (nous discutons de l'initialisation de la matrice dans la section 3.2.1). La matrice $\text{INC}^{(k)}$ pour les 3 premières itérations est la suivante :

	k=0	k=1	k=2	k=3
$\text{INC}^{(k)}[x_1, a_1]$	1	1	1	**1**
$\text{INC}^{(k)}[x_1, a_3]$	1	1	0,75	0,75
$\text{INC}^{(k)}[x_2, a_2]$	1	1	1	**1**
$\text{INC}^{(k)}[x_2, a_4]$	1	0,5	0,5	0,5
$\text{INC}^{(k)}[x_3, a_1]$	1	1	1	**1**

TABLEAU 3.1 – Itérations dans le calcul de la matrice INC.

Au terme de la troisième itération, les valeurs des degrés d'inclusion sont stabilisées et conduisent à l'appariement optimal suivant : $App = \{x_1/a_1; x_2/a_2; x_3/a_1\}$. La valeur d'inclusion globale I_π est donc :

$$\begin{aligned} I_\pi(h, ex) &= \frac{1}{4}\Big(\text{INC}[x_1, a_1] + 2.\text{INC}[x_2, a_2] + \text{INC}[x_3, a_1]\Big) \\ &= 1 \end{aligned}$$

3.2 Paramètres du calcul de la matrice d'inclusion

Nous allons à présent discuter des paramètres suivants concernant l'algorithme : l'initialisation de la matrice INC, le critère d'arrêt des itérations pour le calcul de cette matrice ainsi que la possibilité de pondérer les littéraux ou les variables dans le calcul des degrés d'inclusion. Nous nous interrogeons également sur la manière de modifier certains comportements gênants.

3.2.1 Initialisation de la matrice

Comme pour le calcul de la matrice SIM dans la section 2.4.1, les éléments de la matrice INC correspondent aux variables d'un système d'équations non linéaire. La résolution de ce système est réalisée de manière itérative par la méthode de Jacobi qui nécessite d'initialiser les éléments de INC.

Se pose alors la question des valeurs d'initialisation, l'idée étant de trouver une valeur qui soit la plus proche possible de la valeur finale pour réduire le nombre d'itérations nécessaire à la stabilisation du système. Malheureusement, les valeurs finales ne sont pas connues à l'avance. La seule indication que l'on ait, c'est que les éléments

INC$[i, j]$ représentent des degrés d'inclusion et doivent donc prendre leur valeur dans l'intervalle [0, 1].

Il semble logique d'initialiser chaque élément INC$[v_i, v'_j]$ avec la valeur d'inclusion $Di_{var}(v_i, v'_j)$. Si les variables sont caractérisées par des attributs (les littéraux définissant les variables ont n constantes), on peut déjà fournir au système une information importante et on peut s'attendre à ce que les valeurs de la matrice convergent plus rapidement. C'est effectivement ce que l'on observe expérimentalement. Par contre, lorsque les variables ne sont décrites par aucun attribut, la valeur de $Di_{var}(v_i, v'_j)$ est égale à 1 et la matrice est donc initialisée avec la valeur 1.

3.2.2 Arrêt des itérations

Le nombre d'itérations nécessaire pour que les éléments de la matrice se stabilisent n'est pas connu à l'avance car il dépend :

- du nombre et de la longueur des chemins entre les variables (*i.e.* séquences de littéraux) : plus ils sont grands et nombreux, plus le nombre d'itérations sera important ;
- du fait que les degrés d'inclusion entre les variables et entre les relations sont très "contrastées" ou non, c'est-à-dire qu'il y ait des valeurs différentes ce qui conduit à des choix d'appariements rapides ; la convergence est alors rapide. Dans le cas contraire (si toutes les valeurs sont égales à une même constante), seule l'information topologique fait la différence et il faut qu'elle se "diffuse" dans la structure ce qui nécessite plus d'itérations pour que le système converge.

Plusieurs critères d'arrêt peuvent être définis pour déterminer à quel moment l'algorithme peut être arrêté. Une première solution consiste à mesurer la variation des valeurs de $\text{INC}_{i,j}$ d'une itération sur l'autre. Pour cela, on définit la matrice $\text{DIFF}^{(k)}$ qui évalue la différence entre la matrice $\text{INC}^{(k)}$ à l'itération k et la matrice $\text{INC}^{(k-1)}$ à l'itération précédente k-1 :

$$\text{DIFF}^{(k)}_{i,j} = \text{INC}^{(k)}_{i,j} - \text{INC}^{(k-1)}_{i,j} \tag{3.6}$$

Dans ce cas, l'algorithme est arrêté lorsque la plus grande valeur $\text{DIFF}_{i,j}$ est inférieure à une valeur seuil ϵ. La diculté réside dans le choix du seuil ϵ qui dépend de la complexité des clauses traitées et du domaine dapplication. Si le calcul de la matrice est arrêté trop tôt, le risque est que les valeurs ne reètent pas les degrés dinclusion réels ; on choisira donc une valeur suffisamment petite (par exemple, $\epsilon = 10^{-4}$).

Dans le cas d'applications où l'ordre partiel entre les paires de variables, induit par les valeurs de la matrice, est important (en classification automatique par exemple), une

autre possibilité consiste à surveiller l'ordre des éléments d'une itération à l'autre et à détecter le moment où il n'y a plus d'inversion dans les valeurs des éléments de la matrice (*i.e.* lorsque l'on a un ordre stable). On peut évidemment combiner l'utilisation de la valeur seuil et le respect de l'ordre des éléments.

3.2.3 Privilégier les variables ou les littéraux

Nous avons vu qu'il existe deux sortes de littéraux : ceux qui n'ont qu'une variable en argument et qui définissent les propriétés de cette variable et ceux qui ont au moins deux variables en argument et qui définissent une relation entre ces variables. Le calcul des éléments de la matrice INC, défini dans l'équation 3.4, tient compte, à part égale, de ces deux types de littéraux : les littéraux à un seul argument avec ($Di_{var}(v_i, v'_m)$) et les littéraux qui définissent les relations entre variables (valeur Nsim renvoyée par la procédure Meilleur_Appariement_Litteraux).

Dans certains cas, il peut être intéressant de pondérer le degré d'inclusion entre variables ou celui entre littéraux, en privilégiant plutôt les attributs locaux ou la topologie des structures. Pour cela, nous introduisons le paramètre α de la manière suivante :

$$\text{INC}[v_i, v'_m] = (1-\alpha).Di_{var}(v_i, v'_m) + \alpha\frac{Nsim}{\left|L(v_i)\right|} \qquad \text{avec } 0 \leq \alpha < 1 \tag{3.7}$$

Plus le coefficient α est proche de 0, plus l'algorithme ne tient compte que des degrés d'inclusion intrinsèques aux variables (littéraux ne contenant qu'une seule variable). A l'inverse, lorsque α tend vers 1, il se focalise sur la topologie. Si $\alpha = 1$, alors le système ne peut pas converger.

Cette pondération a été implémentée dans le calcul de la matrice INC ; par défaut, la valeur de α est fixée à 0.5. Un autre point qui n'a pas été considéré consiste à pondérer directement les symboles de prédicats au lieu des littéraux, ce qui revient à pondérer l'étiquette . Cela nécessite cependant de considérer autrement le calcul de la matrice INC.

3.2.4 Appariement local des variables

Le calcul du degré d'inclusion entre deux variables nécessite de rechercher le meilleur *appariement local* entre leurs variables voisines ; cette tâche est effectuée par l'algorithme hongrois. Cet algorithme définit un graphe bipartite à partir des ensembles de variables voisines et recherche le couplage de poids maximal entre les deux ensembles de variables voisines.

A l'itération k du calcul de la matrice $\text{INC}^{(k)}$, les degrés d'inclusion utilisés par l'algorithme hongrois sont pris dans la matrice $\text{INC}^{(k-1)}$ de l'itération précédente. Pour un

même graphe bipartite (mêmes ensembles de variables voisines), il apparaît que, d'une itération à l'autre, les choix d'appariements locaux peuvent changer : localement, les appariements entre les variables voisines sont optimaux mais d'un point de vue global, ils ne reflètent pas la structure globale des clauses.

Considérons l'exemple de la figure 3.2 où, à l'itération k, il s'agit d'apparier, de manière unique, les variables voisines de x_1 (ensemble V=$\{x_2,x_3\}$) dans la clause h avec les variables voisines de x'_1 (ensemble V'=$\{x'_2, x'_3\}$) dans la clause ex.

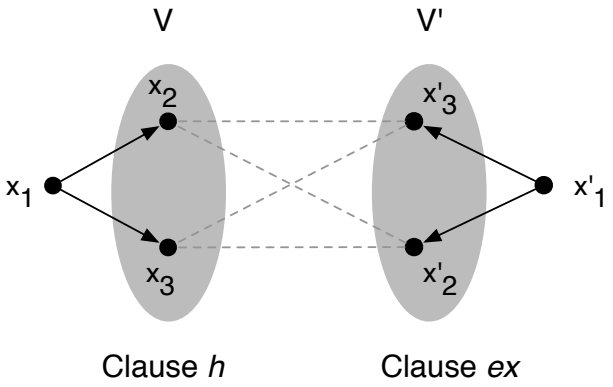


FIGURE 3.2 – Exemple de choix d'appariement local entre variables voisines.

Les matrices correspondantes $\text{INC}^{(k)}$ (à gauche) et $\text{INC}^{(k+1)}$ (à droite) sont les suivantes. Les degrés d'inclusion écrits en gras correspondent aux appariements effectués par l'algorithme hongrois qui maximise la somme des valeurs des appariements.

$$\begin{array}{c} \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cc} x'_2 & x'_3 \end{array} \\ \begin{pmatrix} 0,36 & \mathbf{0,75} \\ \mathbf{0,40} & 0,73 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cc} x'_2 & x'_3 \end{array} \\ \begin{pmatrix} \mathbf{0,35} & 0,68 \\ 0,30 & \mathbf{0,70} \end{pmatrix} \end{array}$$

On remarque qu'à l'itération k, la variable x_2 est appariée à la variable x'_3 et la variable x_3 est appariée à la variable x'_2. Si l'on regarde la matrice obtenue à l'itération $k+1$, on remarque que l'appariement local a changé : la variable x_2 est maintenant appariée à la variable x'_2 et la variable x_3 à x'_3. Pris séparément, chacun de ces deux appariements est tout à fait cohérent et optimal mais, d'une itération à l'autre, ils sont différents. Ainsi, le calcul de la matrice INC maximise le degré d'inclusion local entre deux variables v_i et v'_m à chaque itération, en dépit des choix effectués dans les itérations précédentes ; ce qui conduit finalement à une surestimation du degré d'inclusion réel.

Pour éviter ce comportement, il faudrait qu'à chaque fois qu'un appariement a été réalisé dans la phase globale, les valeurs de la matrice INC soient recalculées en ayant

fixé l'appariement qui vient d'être réalisé. Cette amélioration, non encore implémentée, est une des perspectives importantes de l'algorithme.

Le changement d'appariements locaux d'une itération à l'autre peut également apparaître lorsque la matrice contient des valeurs identiques, comme dans la matrice ci-dessous. Lors de l'appariement des variables voisines de x_1 et x'_1, il existe dans ce cas deux appariements optimaux : $App_1 = \{x_2/x'_2; x_3/x'_3\}$ et $App_2 = \{x_2/x'_3; x_3/x'_2\}$ ont la même valeur d'appariement.

$$\begin{array}{c} \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cc} x'_2 & x'_3 \end{array} \\ \begin{pmatrix} \mathbf{0.36} & \mathbf{0.75} \\ \mathbf{0.36} & \mathbf{0.75} \end{pmatrix} \end{array}$$

Ces changements d'appariements peuvent poser un problème dans la phase d'appariement global du fait que les choix effectués ici ne seront pas forcément cohérents avec ceux réalisés dans la phase constructive. De plus, lors du calcul du degré d'inclusion entre deux clauses données, on sait que la valeur finale sera faussement élevée.

3.2.5 Appariement des variables

Les valeurs contenues dans la matrice INC correspondent au degré d'inclusion de chaque paire de variables entre deux clauses données qui traduit l'étendue du recouvrement de la variable v_m ainsi que son environnement (défini par l'ensemble de ses variables voisines) dans l'environnement de la variable v'_m dans la clause *ex*. Une valeur proche de 1 signifie que le premier environnement est fortement inclus dans le deuxième ; à l'inverse, une valeur proche de 0 indique un environnement très peu inclus et donc différent.

Cette matrice contient donc des informations structurales sur les deux clauses et l'observation des valeurs permet d'identifier les paires de variables fortement incluses ou au contraire dissemblables. Il est donc important que ces valeurs soient cohérentes avec la structure qu'elles décrivent, ce qui n'est cependant pas toujours le cas. En effet, considérons par exemple les clauses *h* et *ex* illustrées sur la figure 3.3 qui contiennent chacune un cycle.

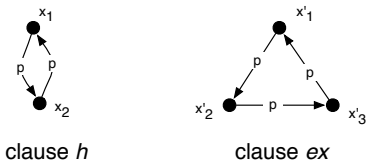


FIGURE 3.3 – Les clauses h et ex sont définies respectivement par $h \leftarrow p(x_1, x_2), p(x_2, x_1)$ et $ex \leftarrow p(x'_1, x'_2), p(x'_2, x'_3), p(x'_3, x'_1)$.

Du fait de la structure cyclique de ces clauses, le système d'équations correspondant au calcul du degré d'inclusion entre les variables (algorithme 2 et équation 3.4) est formé de la même équation, à une permutation des variables près. L'équation relative au calcul de la valeur $\text{INC}^{(k)}[x_1, x'_1]$ est la suivante :

$$\text{INC}^{(k)}[x_1, x'_1] = \frac{1}{2}\left[\frac{1}{2}\Big(1 + \text{INC}^{(k-1)}[x_2, x'_2]\Big) + \frac{1}{2}\Big(1 + \text{INC}^{(k-1)}[x_2, x'_3]\Big)\right]$$

A l'issue du calcul, il se trouve que tous les éléments de la matrice INC sont égaux à 1. Cela signifie que, quels que soient les appariements entre les variables effectués dans la seconde étape de la méthode, les degrés d'inclusion globaux $I_\pi(h, ex)$ et $I_\pi(ex, h)$ seront égaux à 1. En d'autres termes, les deux clauses sont vues comme identiques par l'algorithme ; ce qui n'est clairement pas le cas dans la réalité. Bien que la matrice soit cohérente avec l'algorithme, celui-ci ne fait pas la distinction entre ces deux cycles de taille différente. Un autre exemple, plus simple, est le suivant : $h \leftarrow p(x_1, x_1)$ et $ex \leftarrow p(y_1, y_2), p(y_2, y_1)$.

Pour comprendre le phénomène, considérons à présent les deux exemples illustrés sur la figure 3.4.

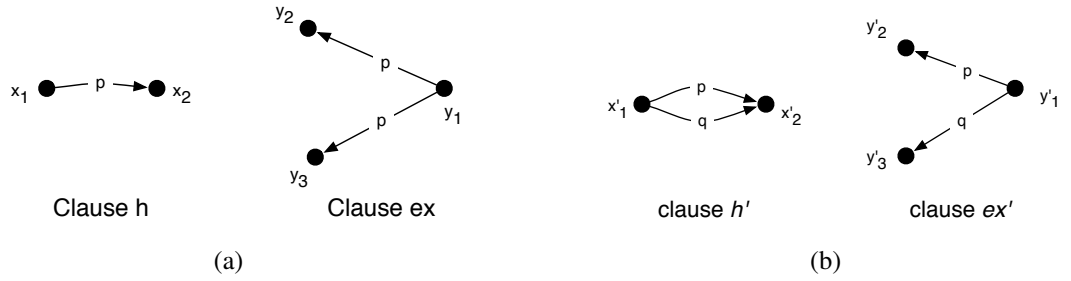


FIGURE 3.4 – (a)a clause h est définie par $p(x_1, x_2)$ et la clause ex est définie par $p(y_1, y_2), p(y_1, y_3)$. (b) La clause h' est définie par $p(x'_1, x'_2), q(x'_1, x'_2)$ et la clause ex' est définie par $ex \leftarrow p(y'_1, y'_2), q(y'_1, y'_3)$.

Dans l'exemple 3.4(a), lors du calcul de Nsim, l'algorithme cherche à apparier le littéral $p(x_1, x_2)$ avec l'un des littéraux : $p(y_1, y_2)$ ou $p(y_1, y_3)$. L'appariement retenu est celui qui maximise la valeur de Di_{lit} :

$$Nsim = \max\left(\frac{1}{2}\Big(1 + \text{INC}[x_2, y_2]\Big), \frac{1}{2}\Big(1 + \text{INC}[x_2, y_3]\Big)\right)$$

Dans ce cas, la variable x_2 sera alors appariée avec y_2 ou y_3.

Si l'on regarde à présent l'exemple de la figure 3.4(b), la recherche du meilleur appariement entre les littéraux implique de calculer le degré d'inclusion de deux sous-ensembles de littéraux (chacun d'eux rassemblant des littéraux de même signature) :

ceux construits sur le prédicat p ($L_p = \{p(x'_1, x'_2), p(y'_1, y'_2)\}$ et ceux construits sur le prédicat q ($L_q = \{q(x'_1, x'_2), q(y'_1, y'_3)\}$. La valeur Nsim retournée par la fonction Meilleur_Appariement_Litteraux est ici égale à :

$$Nsim = \frac{1}{2}\left[\frac{1}{2}\Big(1 + \text{INC}[x'_2, y'_2]\Big) + \frac{1}{2}\Big(1 + \text{INC}[x'_2, y'_3]\Big)\right]$$

On remarque que, dans ce cas, la variable x_2 est appariée à deux variables différentes contrairement à l'exemple précédent où elle n'est appariée qu'à une seule variable. Tout se passe comme si deux variables différentes mais portant le même nom coexistaient dans la clause h, ce qui modifie virtuellement la structure locale "vue" par l'algorithme qui prend alors en compte une structure qui n'existe pas en réalité. La clause $h' \leftarrow p(x_1, x_2), q(x_1, x_3)$ conduirait à la même équation.

Le problème vient de la présence dans la clause h de deux littéraux de signature différente mais contenant une même variable alors que dans la clause ex, les deux littéraux de même signature contiennent deux variables distinctes. Lorsque ce motif se répète à l'intérieur de cycles, des effets de bord apparaissent et la méthode n'est plus en mesure de distinguer des cycles de taille différente et les considère tous comme identiques, comme illustré sur la figure 3.3. De plus, en autorisant les appariements séparés, on peut être amené à surévaluer la similarité globale entre deux variables ce qui, à son tour, va induire une surévaluation de la ressemblance entre les deux structures. Le problème ne semble assurément pas limité aux seuls cycles.

Pour empêcher la prise en compte de ces variables virtuelles, nous modifions l'algorithme de sorte que les appariements ne soient plus dirigés par les littéraux mais par les variables. Cette stratégie est mise en oeuvre par la procédure Meilleur_Appariement_Variables qui construit l'ensemble V (resp. V') des variables voisines de v_i (resp. v'_m). Ces ensembles forment ainsi un graphe bipartite dont le poids des appariements est donné par la variable Nsim (renvoyée par la fonction Meilleur_Appariement_Litteraux). On se retrouve de nouveau dans un problème classique d'optimisation (voir figure 3.5). Pour se ramener à une valeur comprise entre 0 et 1, la somme maximale des Nsim est normalisée par le nombre de variables voisines de la variable v_i.

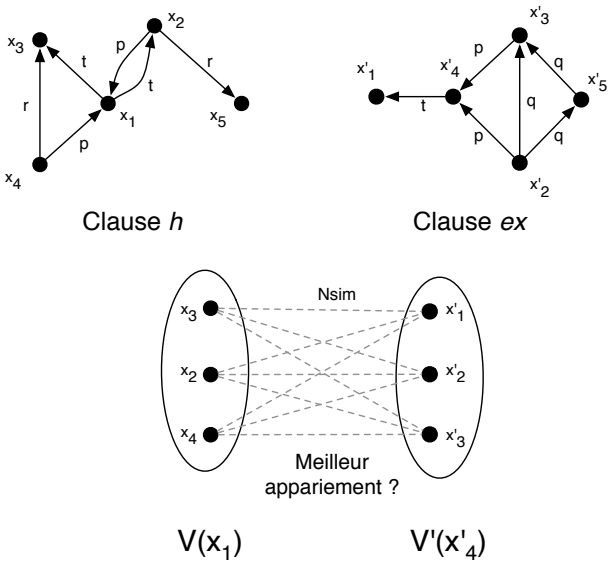


FIGURE 3.5 – Pour évaluer le degré d'inclusion entre les environnements des variables x_1 et x'_4, la procédure Meilleur_Appariement_Variables cherche à apparier leurs variables voisines.

L'algorithme de la procédure Meilleur_Appariement_Variables est le suivant.

Procédure Meilleur_Appariement_Variables
Input : variable v_i, variable v'_m, matrice INC
V ← Variables_Voisines (v_i) ;
V' ← Variables_Voisines (v'_m) ;
pour chaque *paire de variables voisines (v_j , v'_n) dans respectivement V et V'*
faire
  L ← Link_Of (v_j) ;
  L' ← Link_Of (v_n) ;
  Nsim ← Meilleur_Appariement_Litteraux $\left(\text{INC, L, L', } v_i, v'_m\right)$;
l_1 $M_{locale}[v_j , v_n]$ ← Nsim/ | L| ;
finprch
m ← MeilleurAppariement(V, V', M_{locale}) ;
Retourner m ;

Algorithme 3 : Meilleur appariement local entre variables.

Pour chaque paire de variables voisines (v_j, v'_n), la valeur de leur degré d'inclusion INC[v_j, v'_n] est stockée dans une matrice intermédiaire M_{locale} dont la taille est égale à |V|.|V'|. A l'aide de cette matrice locale, la procédure MeilleurAppariement recherche l'appariement entre les éléments de V et V' qui maximise la somme des poids. L'algorithme utilisé peut être l'algorithme hongrois si les appariements sont de type 1-1 mais pas si les appariements sont de type 1-n. Dans ce cas, on pourra utiliser un algorithme

glouton qui, pour chaque variable v_j de h, recherche la variable v'_n telle que la valeur de $\text{INC}[v_j, v'_n]$ soit maximale.

Voici l'algorithme Evaluation_matrice_INC qui intègre les modifications apportées au calcul de la matrice INC ; il remplace l'algorithme 2 du chapitre 2.

Procédure Evaluation_matrice_INC
Input : clause h, clause ex, INC, Iter
pour *k=1* **to** *Iter* **faire**
  pour chaque *paire de variables* (v_i, v'_m) *dans respectivement h et ex* **faire**
    m = Meilleur_Appariement_Variables $\Big(\text{INC}, v_i, v'_m\Big)$;
    $\text{INC}[v_i, v'_m] = (\text{Sim}_{var}(v_i, v'_m) + m/\Big|Variables\_Voisines(v_i)\Big|)/2$;
  finprch
finpour
Retourner INC ;

Algorithme 4 : Calcul de la matrice INC.

Nous reprenons l'exemple de la figure 3.3 avec ce nouvel algorithme. La nouvelle équation qui donne la valeur de $\text{INC}^{(k)}[x_1, a_1]$ est écrite ci-dessous. Les autres équations du système sont identiques, à une permutation des variables près.

$$\text{INC}^{(k)}[x_1, x'_1] = \frac{1}{2}\Big(Di_{var}(x_1, x'_1) + \frac{1}{1} max\Big(M_{locale}[x_2, x'_2], M_{locale}[x_2, x'_3]\Big)\Big) \tag{3.8}$$

avec

$$\begin{aligned} M_{locale}[x_2, x'_2] &= \frac{1}{2}\Big[\frac{1}{2}\Big(1 + \text{INC}^{(k-1)}[x_2, x'_2]\Big)\Big] \\ M_{locale}[x_2, x'_3] &= \frac{1}{2}\Big[\frac{1}{2}\Big(1 + \text{INC}^{(k-1)}[x_2, x'_3]\Big)\Big] \end{aligned}$$

où le coefficient multiplicateur à gauche de chacune des équations correspond à la normalisation par le nombre de littéraux dans $L(x_1)$. Cette fois, à l'issue du calcul de la matrice INC, toutes ses valeurs sont égales à 0,71. Le fait qu'elles ne soient plus égales à 1 signifie que l'algorithme différencie maintenant ces deux cycles.

Malheureusement, cette modification ne fonctionne que sur des cycles de petite taille et ne résout pas le problèmes des cycles de plus grande taille. Considérons l'exemple de la figure 3.6.

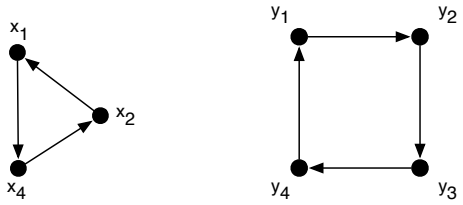


FIGURE 3.6 – Exemple de cycles contenant respectivement 3 et 4 variables.

Le calcul du degré d'inclusion entre les variables x_1 et y_1 s'écrit :

$$\mathrm{INC}^{(k)}[x_1, y_1] = \frac{1}{2}\Big(Di_{var}(x_1, y_1) + \frac{1}{2}\big(M_{locale}[x_2, y_4] + M_{locale}[x_4, y_2]\big)\Big)$$

avec

$$\begin{aligned} M_{locale}[x_2, y_4] &= \frac{1}{1}\Big[\frac{1}{2}\big(1 + \mathrm{INC}^{(k-1)}[x_2, y_4]\big)\Big] \\ M_{locale}[x_4, y_2] &= \frac{1}{1}\Big[\frac{1}{2}\big(1 + \mathrm{INC}^{(k-1)}[x_4, y_2]\big)\Big] \end{aligned}$$

La matrice INC résultante ne contient de nouveau que des valeurs égales à 1. Le tableau 3.2 indique pour des tailles de cycles différents, la valeur des éléments de INC.

	1	2	3	4	5	...	*n*
1	1	1	**0,71**	**0,71**	**0,71**	**...**	**0,71**
2	1	1	**0,71**	**0,71**	**0,71**	**...**	**0,71**
3	1	1	1	1	1	...	1
4	1	1	1	1	1	...	1
5	1	1	1	1	1	...	1
...	...	...	...	...	...	...	...
n	1	1	1	1	1	...	1

TABLEAU 3.2 – Valeurs de la matrice INC pour différentes tailles de cycles.

Les éléments situés sur la diagonale sont égaux à 1, ce qui est correct car les clauses sont identiques. On remarque que seuls les cycles de taille 1 et 2 sont correctement différenciés par l'algorithme proposé (valeurs dans INC inférieures à 1). Pour tous les autres cycles, la modification n'a aucun effet.

La différence se situe au niveau du facteur de normalisation |L| (ligne l_1 dans l'algorithme 3) pour normaliser le résultat du meilleur appariement lors de la construction de la matrice locale $\mathrm{M_{locale}}$. Dans les cycles de taille 1 et 2, l'ensemble de littéraux L

contient deux éléments (il y a deux arcs entre deux variables adjacentes) ; c'est ce qui permet de diminuer la valeur finale du degré d'inclusion.

En effet, dans la clause *h*, il y a deux littéraux alors que dans la clause *ex*, il n'y en a qu'un seul : l'algorithme distingue bien une différence locale. Par contre, dans les cycles de taille 3 et supérieure, ce facteur est toujours égal à 1 les variables dans *h* ou dans *ex* n'ayant qu'un seul littéral en commun avec chacune de leurs variables voisines, il n'y a, localement, aucune différence. Ce qui explique comment l'algorithme se trompe et pourquoi, au niveau global, deux cycles différents semblent identiques.

En pratique, ce problème sera pris en compte lors de la seconde étape de l'algorithme, celui de la mise en appariement global des variables.

3.3 Stratégies d'appariement global

Une fois que la matrice INC est calculée, le problème est maintenant de rechercher le meilleur appariement entre les variables des deux clauses, c'est-à-dire celui qui maximise le degré d'inclusion global I_π. Le problème essentiel est donc un problème d'optimisation que nous abordons ici par des méthodes heuristiques et stochastiques.

3.3.1 Algorithme hongrois

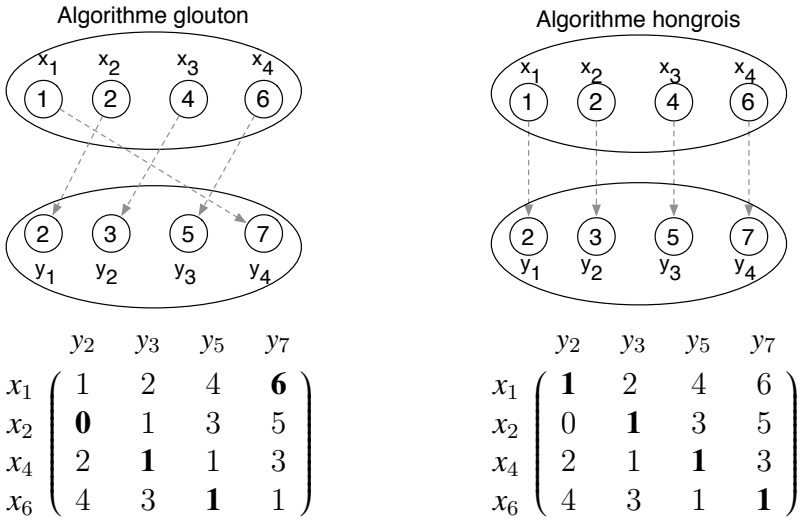
L'algorithme glouton, proposé initialement par [Bisson, 1995], n'est pas optimal dans le sens où il ne garantit pas de trouver le meilleur appariement global. En effet, pour apparier chaque variable, il fait un choix optimum local dans l'espoir d'obtenir un résultat optimum global mais ne le garantit pas.

Pour le vérifier, considérons l'exemple illustré sur la figure 3.7 dans lequel chaque ensemble est constitué de 4 variables auxquelles est associé un nombre entier. Le poids des appariements est défini par la différence entre les nombres associés aux variables. Le problème consiste alors à apparier les variables en minimisant la somme des poids entre les variables appariées.

La première variable appariée par l'algorithme glouton est la variable x_2 avec la variable y_2 car le poids entre ces variables ($M[x_2, y_2] = 0$) est le plus petit de toutes les valeurs de la matrice. Les variables x_3, x_4 et x_1 sont ensuite appariées dans cet ordre. La somme des poids est alors égale à 8.

Pour résoudre ce problème d'appariement, l'algorithme hongrois [Kuhn, 1955] recherche le couplage de poids maximal dans le dont les ensembles de sommets sont $\{x_1, x_2, x_3, x_4\}$ et $\{y_1, y_2, y_3, y_4\}$. L'appariement optimal est $App = \{x_1/y_1,\ x_2/y_2, x_3/y_3,$

$x_4/y_4\}$ et la somme des poids, qui est égale à 4, est plus petite que celle obtenue avec l'algorithme glouton. Cependant, l'algorithme hongrois ne peut être utilisé que pour des appariements uniques. C'est une contrainte limitative forte pour les applications en logique du premier ordre. La variante qui implémente cet algorithme pour l'appariement global est identifiée sous le nom de Ipi_Kuhn.



$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_4 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccc} y_2 & y_3 & y_5 & y_7 \end{array} \\ \begin{pmatrix} 1 & 2 & 4 & \mathbf{6} \\ \mathbf{0} & 1 & 3 & 5 \\ 2 & \mathbf{1} & 1 & 3 \\ 4 & 3 & \mathbf{1} & 1 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ x_1 \\ x_2 \\ x_4 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccc} y_2 & y_3 & y_5 & y_7 \end{array} \\ \begin{pmatrix} \mathbf{1} & 2 & 4 & 6 \\ 0 & \mathbf{1} & 3 & 5 \\ 2 & 1 & \mathbf{1} & 3 \\ 4 & 3 & 1 & \mathbf{1} \end{pmatrix} \end{array}$$

FIGURE 3.7 – Comparaison des algorithmes glouton (à gauche) et hongrois (à droite). En dessous, les matrices de poids correspondantes. Les éléments en gras correspondent aux variables appariées avec chacun des algorithmes.

En modélisant l'appariement des variables sous forme d'un graphe bipartite, ces deux algorithmes se focalisent sur les variables à apparier et tiennent uniquement compte des valeurs des degrés d'inclusion dans la matrice INC, ce qui peut conduire à des appariements qui ne correspondent pas à des sous-structures communes connexes.

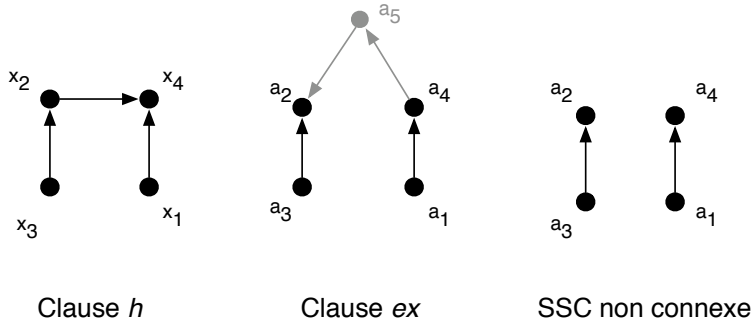


FIGURE 3.8 – L'utilisation de l'algorithme hongrois ou de l'algorithme glouton peut générer des appariements non connexes. Les variables en gras sont celles qui sont appariées.

Sur l'exemple de la figure 3.8, nous considérons l'appariement suivant : $App = \{x_1/a_1,\ x_2/a_2,\ x_3/a_3,\ x_4/a_4\}$. Les littéraux entre les paires de variables (x_2, x_3) et (x_1, x_4) se retrouvent dans la SSC car il existe les mêmes littéraux entre les images de ces variables, dans la clause *ex*. Par contre, le littéral dans *h* entre les variables x_2 et x_4 ne correspond à aucun littéral dans *ex* : il n'appartient donc pas à la SSC et comme c'est ce littéral qui "réunit" les deux parties de *h*, la SSC générée n'est pas connexe.

Ces cas peuvent être gênants dans certains domaines d'application comme la chimie où il est important de conserver certaines sous-structures intactes car elles ont un sens chimique.

3.3.2 Recherche locale par satisfaction de contraintes

Une alternative aux algorithmes présentés ci-dessus consiste à redéfinir le problème de l'appariement comme un PSC (voir section 2.3.1). Dans ce domaine, l'AMC [Minton *et al.*, 1993] est une méthode de recherche locale qui semble bien adaptée à la recherche d'un appariement maximal.

A l'initialisation du système, toutes les variables de la clause *h* sont appariées ; la façon la plus simple est de réaliser un appariement aléatoire. L'algorithme visite ensuite un état voisin en modifiant l'appariement d'une variable à la fois de façon à minimiser le nombre de conflits à l'intérieur du système. Un conflit apparaît si, étant donné deux littéraux $p(x,y)$ dans *h* et $p(x',y')$ dans *ex*, on a $x' = \pi(x)$ et $y' \neq \pi(y)$ (ou $x' \neq \pi(x)$ et $y' = \pi(y)$). En réduisant le nombre de conflits, on augmente le nombre de contraintes satisfaites (*i.e.* le nombre de littéraux ne contenant que des variables appariées) ce qui accentue les chances d'identifier un appariement connexe.

L'algorithme s'arrête lorsqu'une solution complète au PSC a été trouvée (dans ce cas, le nombre de conflits est nul) ou lorsque le nombre maximal d'itérations (variable max_{Iter} dont la valeur est fixée par l'utilisateur) a été atteint ; dans ce cas, l'état courant du système (le dernier appariement *App* trouvé) est retourné. L'algorithme AMC original, appelé AMC_non_Guidé (avec initalisation aléatoire du système), est le suivant.

```
Procédure AMC_non_Guidé
Input : entier max_Iter, clause h, clause ex
appCourant ← Initialise_Appariement () ;
pour i ← 1 to max_Iter faire
    x ← Variable_En_Conflit (h, ex, appCourant) ;
    a ← Valeur_Appariement_Minimale (h, ex, appCourant) ;
    Apparier la variable x avec la variable a;
    si NombreConflits(appCourant) = 0 alors
        Retourner appCourant;
    finsi
finpour
Retourner appCourant;
```

Algorithme 5 : AMC original (AMC_non_Guidé).

Les trois fonctions principales sont les suivantes.

La fonction ***Initialise_Appariement*** apparie toutes les variables de la clause *h* de façon aléatoire (voir figure 3.9a). Si l'appariement est de type 1-1, on veillera à ne pas apparier plusieurs fois une variable de *ex*.

La fonction ***Variable_En_Conflit*** construit la liste des variables en conflit dans *h* et renvoie l'une d'elles ; dans AMC_non_Guidé, le choix de cette variable est aléatoire. Dans l'exemple de la figure 3.9a, il y a un conflit sur les arêtes (x_2, x_3) et (x_1, x_4) : l'ensemble des variables en conflit est alors égal à $\{1, 2, 3, 4\}$. La fonction Variable_En_Conflit renvoie alors une de ces quatre variables.

La fonction ***Valeur_Appariement_Minimale*** établit la liste des variables de la clause *ex* avec lesquelles v_i (renvoyée par Variable_En_Conflit) est appariable [7] et calcule le nombre de conflits que les appariements pourraient générer. Elle renvoie alors la variable qui minimise le nombre de conflits dans le système. Dans l'exemple, suite à l'initialisation, x_2 est en conflit une fois, x_3 deux fois et x_4 une seule fois. Si l'appariement de la variable x_3 est modifié et passé sur la variable a_3 (figure 3.9b), alors le système n'a plus de conflit : l'appariement est total.

7. Dans le cas d'un appariement unique, cette liste ne contient que les variables non encore appariées.

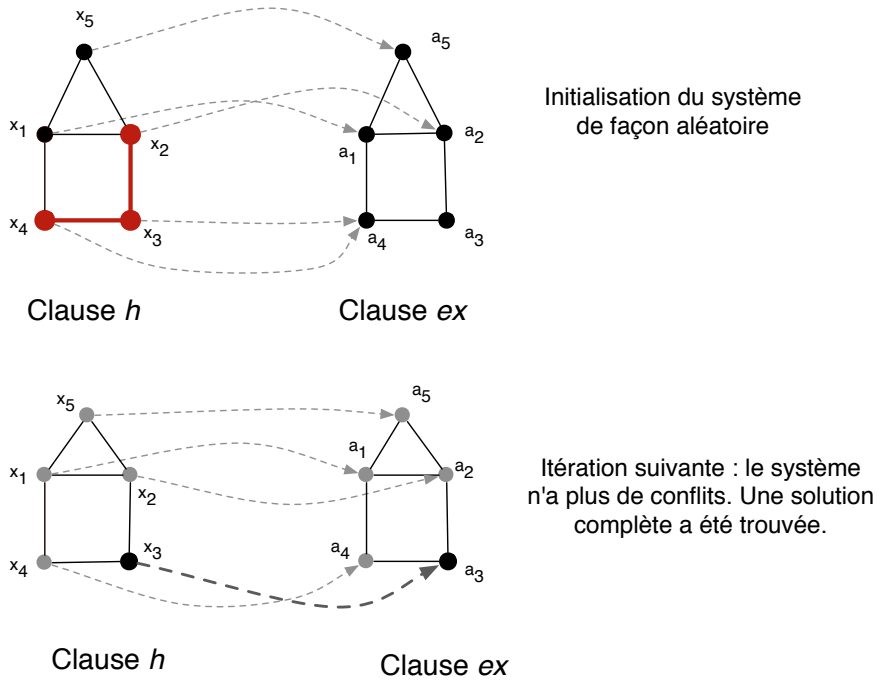


FIGURE 3.9 – En haut (figure 3.9a) : la fonction Initialise_Appariement apparie toutes les variables. En bas (figure 3.9b) : apparier la variable x_3 avec la variable a_3 fait disparaître les conflits.

Dans l'algorithme AMC_non_Guidé, la recherche d'une solution ne repose que sur la notion de conflit et consiste à identifier toutes les variables en conflit, sans aucune sélection. De ce point de vue, la modification des appariements reste locale à une paire de variables et ne réduit pas le nombre de variables candidates à l'appariement. Pour limiter cette combinatoire, nous utilisons la matrice INC et le degré d'inclusion I_π pour guider la recherche des appariements, en complément du nombre de conflits. Ce nouvel algorithme, appelée AMC_Guidé, est décrit sur la page suivante.

La procédure *Initialise_Appariement_Guide* utilise les valeurs de la matrice INC pour apparier, de façon gloutonne, les variables de h avec celles de ex, c'est-à-dire que chaque variable x_i est appariée à la variable x'_m telle que, quelle que soit x'_m, la valeur de INC $[x_i, x'_m]$ soit maximale (valeur maximale dans la ligne INC$[x_i,]$. La procédure *Valeur_Appariement_Minimale_Guide* renvoie une variable x'_m dans ex telle que l'appariement de x_i (renvoyée par Variable_En_Conflit_Guide) avec x'_m minimise le nombre de conflits et maximise la valeur de l'indice $I_\pi(h, ex,$ appCourant).

Si l'appariement est unique et que la clause h contient plus de variables que la clause ex, un appariement total ne pourra pas être trouvé. Dans ce cas, l'algorithme s'arrête au terme des max_{Iter} itérations et retourne le meilleur appariement trouvé.

```
Procédure AMC_Guidé
Entrées : matrice INC, entier max_Iter, clause h, clause ex, entier t_max
listeTaboue = ∅ //(file First In First Out (FIFO) de taille n) ;
compteur ← 0;
appCourant ← Initialise_Appariement_Guide () ;
appPrecedent ← appCourant ;
meilleurApp ← appCourant ;
pour i ← 1 to max_Iter faire
    si I_π (h, ex , appCourant) =1 alors
        Retourner appCourant;
    sinon
        si appCourant ∈ listeTaboue alors
            appCourant ← Initialise_Appariement_Guide () ;
        sinon
            SupprimerTête(listeTaboue);
            Ajouter(appCourant, listeTaboue) ;
            si I_π(h, ex , appCourant) = I_π(h, ex ,appPrecedent) alors
                compteur ← compteur +1;
            finsi
            si compteur = t_max alors
                compteur ← 0;
                appCourant ← Initialise_Appariement_Guide ();
            sinon
                si I_π (h, ex , appCourant) > I_π (h, ex , meilleurApp) alors
                    meilleurApp ← appCourant ;
                    appPrecedent ← appCourant ;
                finsi
            finsi
        finsi
    finsi
    x ← Variable_En_Conflit (h, ex, app) ;
    a ← Valeur_Appariement_Minimale_Guide (h, ex, app) ;
    Apparier la variable x avec la variable a ;
finpour
Retourner meilleurApp;
```

Algorithme 6 : AMC modifié (AMC_Guidé).

Lorsque l'algorithme est guidé par la fonction objectif I_π, il peut se retrouver piégé dans des optima locaux de la fonction : il visite alors souvent les mêmes états voisins et a tendance à atteindre toujours le même optimum local. L'introduction d'une *liste taboue* contenant les n états précédemment visités permet de détecter ce cas de figure (dans

notre cas, la liste taboue est implémentée par une file FIFO et sa longueur n est fixée de façon arbitraire à 10). Si l'algorithme arrive dans un état appartenant à la liste, le système est perturbé pour le forcer à atteindre un état non encore visité. Cette perturbation peut être réalisée par le changement aléatoire de l'appariement d'une variable ou un appel à la fonction Initialise_Appariement_Guide qui réinitialise alors l'appariement de toutes les variables. Nous utilisons ici cette deuxième solution car elle perturbe le système de façon plus importante que le changement d'une seule variable et augmente ainsi la probabilité de sortir d'un optimum local.

L'algorithme peut également se trouver piégé sur un plateau de la fonction objectif : pendant un nombre d'itérations t, la valeur de I_π reste constante quels que soient les états visités. La variable *compteur* comptabilise le nombre d'itérations pendant lesquelles la valeur de I_π reste constante : si la valeur de *compteur* dépasse une valeur seuil définie par l'utilisateur (dans notre cas, ce seuil est fixé à 10 itérations) alors le système est perturbé de la même façon que précédemment.

3.4 Recherche d'Arbres Partiellement Couvrants

Dans certains domaines, il est important d'identifier des appariements connexes, c'est-à-dire des appariements qui conduisent à des SSC connexes. Or, les algorithmes présentés dans la section précédente ne sont pas adaptés à ce type de recherche car ils ne peuvent pas garantir de trouver un appariement connexe. Nous proposons ici une autre stratégie d'appariement qui décompose deux clauses (ou graphes) en un ensemble de SSC connexes. La recherche des plus grandes SSC possibles permet d'identifier la SSCM.

3.4.1 Génération d'un appariement

Pour réduire la complexité de la recherche d'une SSC (qui, rappelons-le, correspond à un problème d'isomorphisme de sous-graphe partiel), le problème n'est pas traité directement. Nous allons utiliser le fait qu'une SSC est définie à partir d'un appariement (section 2.1.3) pour rechercher plutôt des *Arbre Partiellement Couvrant (APC)* (section 1.2) à partir desquels les SSC seront ensuite déduites.

L'algorithme repose sur la construction en parallèle de deux arbres isomorphes (un arbre pour chacune des clauses), comme illustré sur la figure 3.10. Contrairement aux algorithmes glouton et hongrois, on se guide ici directement sur la structure des objets en complément des informations apportées par la matrice INC.

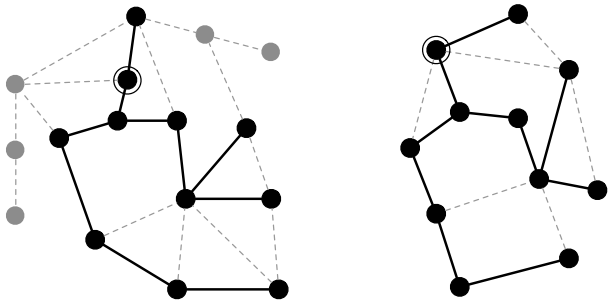


FIGURE 3.10 – Les deux arbres sont isomorphes. L'arbre de gauche est partiellement couvrant tandis que celui de droite est totalement couvrant (tous les sommets du graphe sont contenus dans l'arbre). Les sommets entourés d'un cercle sont les racines des arbres.

La construction des APC est réalisée au moyen d'un parcours classique de graphe. Deux approches sont possibles : le *parcours en largeur d'abord* ou le *parcours en profondeur d'abord*. L'algorithme de parcours en largeur découvre d'abord les variables situées à une distance k de la variable x_i avant de découvrir toute variable située à une distance $k+1$. Ce type de parcours construit une arborescence en largeur, qui ne contient initialement que sa propre racine (la variable origine v_o). Le parcours en profondeur d'abord explore le plus loin possible les chemins un par un : pour chaque variable, il prend la première variable voisine jusqu'à ce que la variable sélectionnée n'aie plus de voisins non visités, et revient alors à la variable */père*. La complexité de ces deux algorithmes est en $O(V + L)$ où V (resp. L) est l'ensemble des variables (resp. littéraux) de la clause.

Lors du calcul de la matrice INC, les informations concernant une paire de variables donnée sont construites en intégrant les informations des variables voisines : les informations sont propagées sur le voisinage de chaque variable. Le choix de l'algorithme s'est porté sur une exploration en largeur d'abord qui est semblable à la façon dont les informations sont propagées lors du calcul de la matrice.

La recherche d'un appariement entre les clauses h et ex est toujours dirigée par la clause h. L'étape d'initialisation consiste à choisir et apparier une première paire de variables ($x_i \in h$, $x'_m \in ex$). Dans leur clause respective, chacune de ces deux variables est la racine de l'arbre en construction. Cette paire de racines est appelé la *graine* de l'appariement ; le choix de la graine est discuté dans la section suivante.

Avant de décrire l'algorithme, nous devons définir la fonction V_Voisins : $V \rightarrow V^n$ qui, pour une variable donnée de V appelée *père*, renvoie l'ensemble de ses n variables voisines. Le comportement de cette fonction est différent selon qu'elle est appliquée à une variable de la clause h ou de la clause ex et que l'appariement est unique ou multiple.

Nous distinguons donc les deux fonctions suivantes :

- la fonction V_Voisins_H renvoie toujours les variables voisines d'une variable v dans la clause h, quel que soit le type d'appariement ;
- dans le cas d'un appariement de type 1-n, la fonction V_Voisins_EX renvoie toutes les variables voisines de la variable v' appartenant à la clause ex. Dans le cas d'un appariement unique, elle ne renvoie que les variables voisines non encore appariées.

La procédure Generer_Appariement génère un appariement connexe entre deux clauses en construisant les deux APC à partir d'une graine. Le parcours en largeur d'abord est implémenté de façon itérative par une file de type FIFO.

```
   Procédure Generer_Appariement
   Input : matrice INC, graine (x_i, x'_m)
   fifo ← ∅;
   appCourant ← ∅ // liste des paires de variables appariées;
   Ajouter la paire (x_i, x'_m) à appCourant;
   Ajouter V_Voisins_H (x_i) à la file fifo;
B1 tant que fifo ≠ ∅ faire
   |  vCourante ← Tête(fifo);
   |  Supprimer_Tête(fifo);
   |  V ← V_Voisins_H (vCourante);
   |  v ← variable dans ex appariée avec vCourante ;
   |  V' ← V_Voisins_EX (v);
   |  Appariement_Voisins (INC, V, V');
B2 |  pour chaque variable v ∈ V qui vient d'être appariée faire
   |  |  Ajouter v à la file fifo;
   |  finprch
   fintq
   Retourner appCourant ;
```

Algorithme 7 : Recherche parallèle d'APC.

A chaque étape du parcours, une paire de variables (x_i, x'_m) est appariée. On construit alors les ensembles V et V' contenant respectivement les variables voisines de x_i et x'_m qui forment un graphe bipartite pour lequel on recherche l'appariement qui maximise la somme des degrés d'inclusion.

Pour une variable v voisine de x_i dans h qui est appariée à x'_m dans ex, l'ensemble des variables candidates dans ex ne contient ainsi que les variables voisines de x'_m au lieu de contenir toutes les variables de la clause ex. Cette stratégie réduit de façon importante le nombre de variables candidates à l'appariement. Sur l'exemple de la figure 3.11, supposons que les variables de la graine (x_1, a_1) ont été appariées, ainsi que les variables

de la paire (x_2, a_2). L'étape suivante consiste ensuite à apparier les variables voisines de x_2 : l'ensemble $V = \{x_8, x_7, x_6\}$. En l'absence de stratégie de réduction du nombre de candidats, pour chacune de ces trois voisines, il faudrait considérer l'ensemble des 8 variables de *ex*, c'est-à-dire $V' = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$. Alors qu'en utilisant le mécanisme de sélection des candidats à l'appariement, l'ensemble V' est réduit à $V' = \{a_6, a_7, a_8\}$. Cette stratégie permet en outre de garantir que la SSC issue de l'appariement est connexe.

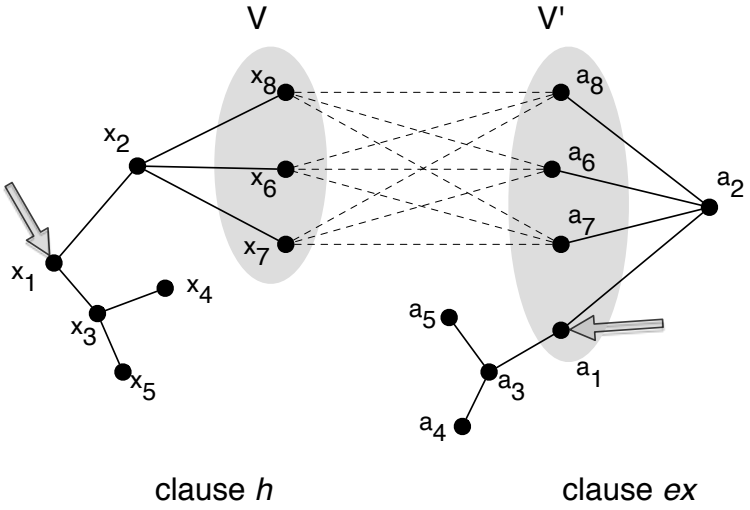


FIGURE 3.11 – Réduction du nombre de candidats à l'appariement.

Le nombre de variables candidates est davantage réduit du fait que l'on ne peut apparier que des variables qui apparaissent à la même position dans des littéraux de même signature. Sur l'exemple de la figure 3.12, chaque variable de l'ensemble V ne peut être appariée qu'à une seule variable de l'ensemble V'.

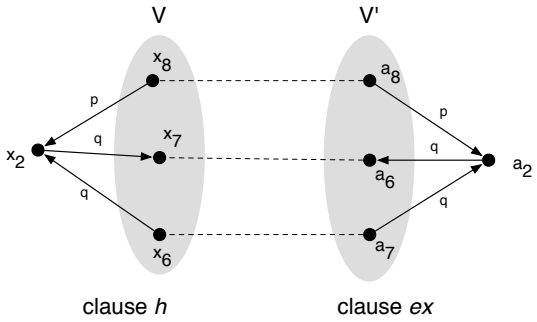


FIGURE 3.12 – Exemple de réduction du nombre de candidats à l'appariement lorsque les littéraux ne sont pas de même signature.

La recherche du meilleur appariement entre les variables de V et V' est ensuite réalisée par la procédure Appariement_Voisins. Lorsque ces ensembles contiennent des variables apparaissant dans des littéraux de signature différente, la procédure crée des sous-ensembles de V et V' ne contenant que des variables apparaissant dans des littéraux de même signature et résout chacun des sous-graphes bipartites générés.

Pour chaque (sous-)graphe bipartite, plusieurs stratégies d'appariement local peuvent être utilisées ici. Nous considérons dans ce travail les variantes suivantes :
- une variante stochastique : les appariements des variables de V et V' sont effectués de manière aléatoire, sans utiliser les informations contenues dans la matrice INC ;
- un algorithme glouton qui cherche, de manière itérative, le meilleur appariement à réaliser pour une variable à la fois ;
- l'algorithme hongrois qui recherche l'appariement optimal.

Chaque variable appariée est ensuite placée à la queue d'une file FIFO, l'algorithme sélectionne alors la prochaine variable à apparier à la tête de la file FIFO. Si cette variable a déjà été appariée auparavant (cela arrive fréquemment lorsque les clauses contiennent des cycles), elle est simplement ignorée : l'appariement d'une variable appariée n'est jamais modifié.

Arrêt de l'algorithme

L'algorithme 7 s'arrête dès lors qu'il n'y a plus de variables à apparier dans h (la file FIFO est vide) ; on renvoie alors l'appariement courant. Cette approche est dite "SansArrêt".

Une deuxième façon d'arrêter la construction d'un appariement est basée sur l'hypothèse suivante. Tant que les variables découvertes lors du parcours de la clause h sont appariées, cela signifie que l'algorithme explore des zones similaires dans les clauses h et ex. Par contre, dès qu'une variable extraite de la file FIFO ne peut pas être appariée (par manque de variable candidate dans la clause ex), cela peut signifier que l'on rencontre une zone différente entre les deux clauses et qu'il deviendra difficile de poursuivre l'appariement. On peut alors arrêter le parcours des clauses à ce moment.

Pour implémenter cette variante dite "AvecArrêt", il suffit d'introduire une variable booléenne *arret* renvoyée par la procédure Appariement_Voisins et qui indique si une variable n'a pu être appariée (VRAI) ou pas (FAUX). L'algorithme général de construction des appariements est alors le suivant.

```
   Procédure Generer_Appariement
   Input : matrice INC, graine (x_i, x'_m)
   arret ← FAUX ;
   fifo ← ∅;
   appCourant ← ∅ // liste des paires de variables appariées;
   Ajouter la paire (x_i, x'_m) à appariement;
   Ajouter V_Voisins_H (x_i) à la file fifo;
B1 tant que (fifo ≠ ∅) ET (arret ≠ VRAI) faire
      vCourante ← Tête(fifo);
      Supprimer_Tête(fifo);
      V ← V_Voisins_H (vCourante);
      v ← variable dans ex appariée avec vCourante ;
      V' ← V_Voisins_EX (v);
      arret ← Appariement_Voisins (INC, V, V');
      si ( arret = FAUX) alors
         Ajouter les variables de V à la file fifo;
      finsi
   fintq
   Retourner appCourant ;
```

Algorithme 8 : Recherche parallèle d'APC avec possibilité d'arrêt sur erreur d'appariement.

Cette recherche d'un APC à partir d'une variable racine rappelle la décomposition en sous-graphes à partir d'un noeud dans [El-Sonbaty and Ismail, 2000]. La différence est que, dans la méthode proposée ici, les deux clauses sont décomposées en parallèle, de façon à identifier des SSC tandis que l'auteur ne s'intéresse qu'à la décomposition d'un seul graphe.

3.4.2 Exemple

L'exemple de la figure 3.13 illustre le déroulement complet de l'algorithme 7 (sans arrêt) dans le cas de graphes simples non orientés. Les différentes étapes de l'exploration sont les suivantes :

1. La graine correspond à la paire de sommets $(1, a)$ qui sont appariés. Les voisins non appariés de 1 sont placés dans la file : $fifo = \{2, 5\}$;
2. On entre dans la boucle B_1 pour traiter tous les sommets restant à apparier. Le premier sommet de la file, le sommet 2, est sélectionné (et enlevé de la file) pour être apparié avec b. Les sommets voisins non appariés du sommet 2 (3 et 5) sont placés dans la file : $fifo = \{5, 3, 5\}$ (boucle B_2) ;
3. Le sommet 5 est apparié avec le sommet f. Le seul sommet voisin non apparié est le sommet 4 qui est ajouté à la file qui contient donc : $fifo = \{3, 5, 4\}$;

4. L'algorithme passe au sommet suivant : le sommet 3 est apparié avec le sommet c et on ajoute son seul voisin non apparié (le sommet 4) dans la file : $fifo = \{5, 4, 4\}$;
5. Le sommet 5 a déjà été apparié, il est ignoré et retiré de la file. La première occurrence du sommet 4 est retirée de la file : le sommet 4 est apparié avec le sommet e. Le sommet 4 n'a pas de voisins non appariés donc la file n'est pas modifiée : $fifo = \{4\}$;
6. Le sommet 4 a été apparié à l'étape précédente : la deuxième occurrence du sommet 4 dans la file est donc ignorée et retirée de la file qui est maintenant vide. L'algorithme s'arrête.

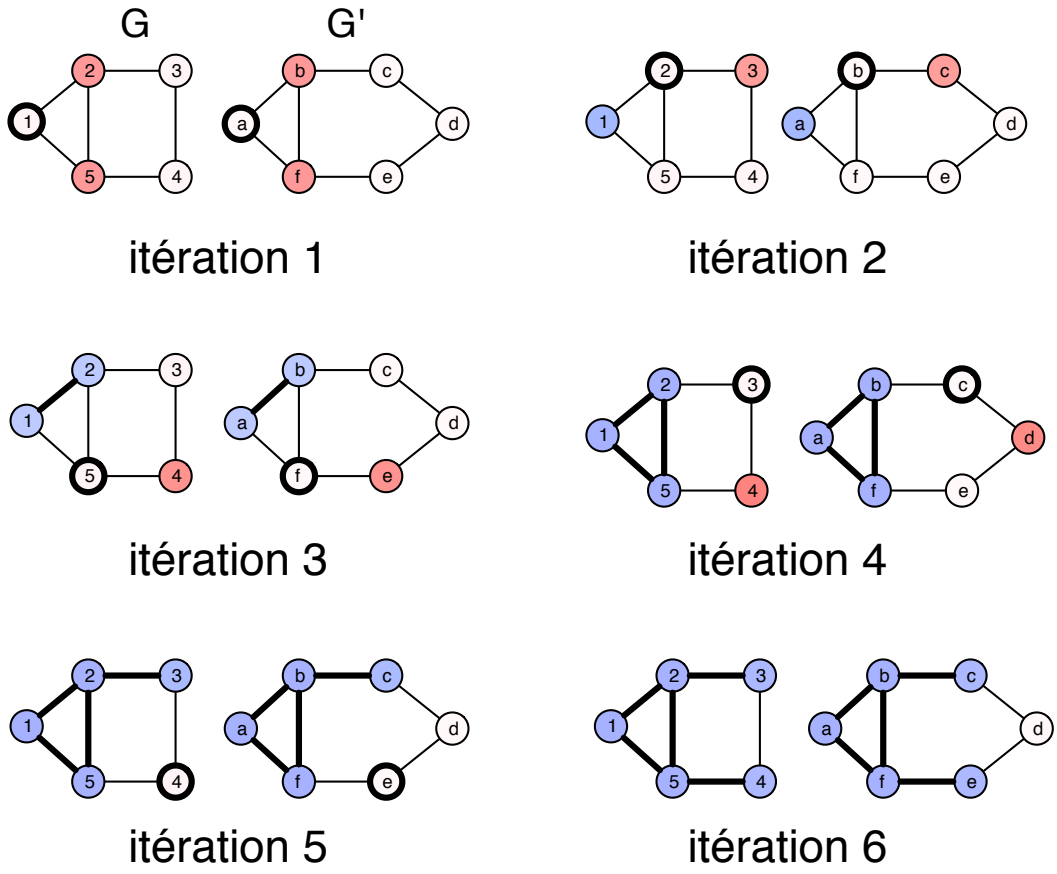


FIGURE 3.13 – Déroulement de l'algorithme de recherche d'APC. Les sommets entourés d'un cercle noir en gras correspondent aux sommets courants qui vont être appariés. Les sommets rouges forment l'ensemble des voisins des variables courantes. Les sommets bleus sont ceux qui ont été appariés lors des itérations précédentes.

A l'issue de l'itération 6, tous les sommets de G sont appariés. Toutes les arêtes de G appartiennent au sous-graphe identifié à l'exception de l'arête $(3, 4)$ car il n'existe

pas d'arête entre $c = \pi(3)$ et $e = \pi(4)$.

3.4.3 Génération des graines

Dans la section précédente, nous avons décrit la recherche d'un appariement *App* entre deux clauses *h* et *ex*, à partir d'une graine quelconque. Cependant, rien ne permet de prévoir si l'appariement généré à partir de cette graine est le plus grand appariement qu'il est possible de trouver entre ces deux clauses. En effet, dans une recherche locale de type constructive (comme le parcours de graphes), le choix du point de départ (*i.e.* la graine) est souvent problématique : l'algorithme peut par exemple démarrer la recherche avec une graine située sur un plateau de la fonction I_π ou dans un minimum local. On souhaite donc identifier plusieurs SSC.

Par exemple, dans le domaine de la chimie, l'algorithme SUPLIMIT [Bayada *et al.*, 1992] combine un parcours de graphe avec une stratégie de retour-arrière pour identifier les plus grandes SSC entre deux graphes moléculaires, à partir d'une graine donnée. Dans notre cas, la recherche d'APC n'a pas de mécanisme de retour-arrière. Aussi, pour augmenter les chances de trouver le plus grand appariement connexe, plusieurs graines sont utilisées dans l'idée de générer un ensemble d'appariements et de choisir ensuite le plus grand d'entre eux. Une première approche consiste à générer la liste exhaustive L_G des $|V|.|V'|$ graines possibles entre les clauses *h* et *ex*. Cependant, pour réduire les temps de calcul, le nombre de graines à tester peut être réduit de différentes façons.

Nous pouvons également utiliser les informations contenues dans la matrice INC pour sélectionner les graines à utiliser. La première stratégie consiste à ne considérer, dans la liste L_G, que le sous-ensemble des *n* graines ayant les plus fortes valeurs dans la matrice. En initialisant un appariement avec une variable fortement incluse dans l'autre (valeur de I_π proche de 1), il y a de plus grandes chances pour tomber à un endroit de la structure qui a de nombreux points communs et donc, on peut penser qu'on aboutira à un appariement de bonne qualité.

Cette stratégie peut être optimisée en considérant l'hypothèse suivante. Supposons qu'un appariement *App* a déjà été réalisé à partir de la graine (x_i, x'_m). Si une seconde graine dont les variables ont déjà été appariées dans *App* est sélectionnée, il y a de fortes chances pour que ce deuxième appariement soit similaire (voire identique) au premier ; ce cas est illustré sur la figure 3.14. Pour limiter cette redondance, à chaque fois qu'un appariement est réalisé, les paires de variables qui viennent d'être appariées sont supprimées de la liste L_G.

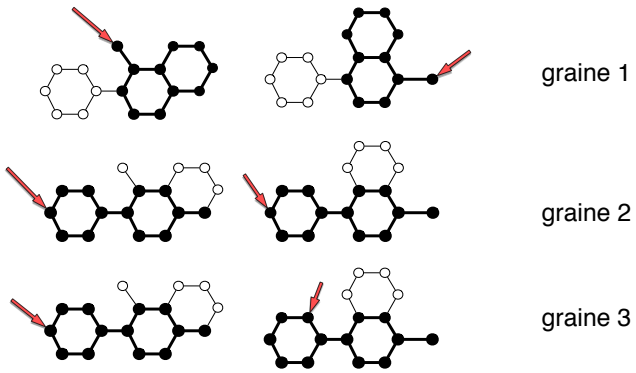


FIGURE 3.14 – Appariements générés par des graines différentes. Les graines 2 et 3 sont proches l'une de l'autre et génèrent les mêmes appariements. La graine 1, très différente des deux autres, génère un appariement différent.

L'algorithme général de recherche d'appariements partiels, appelé I_π, implémente la stratégie décrite ci-dessus. La fonction ConstruitListeTrieeGraines construit la liste L_G des graines et les ordonne dans le sens des valeurs décroissantes de INC. L'algorithme parcourt la liste des graines itérativement en sélectionnant systématiquement la première de la liste. Chaque appariement généré par la procédure Generer_Appariement est stocké dans la liste L_{App}. A chaque appel de la fonction Generer_Appariement, l'ensemble des appariements effectué est oté de la liste L_G (ligne l_1).

Procédure I_π
Input : clause h, clause ex, matrice INC
$L_G \leftarrow$ ConstruitListeTrieeGraines (h, ex, INC);
$L_{App} \leftarrow \emptyset$;
tant que $L_G \neq \emptyset$ **faire**
  Sélectionner la première graine g $\leftarrow (x_i, x'_m)$ dans L_G ;
  $appCourant \leftarrow$ Generer_Appariement (INC, g) ;
l_1 Supprimer dans L_G les paires de variables de $appCourant$;
  $L_{App} \leftarrow L_{App} \cup appCourant$;
fintq
Retourner L_{App} ;

Algorithme 9 : Recherche d'appariements partiels à partir de graines différentes.

Cet algorithme construit donc plusieurs appariements connexes entre deux clauses données. Les stratégies implémentées limitent le nombre d'appariements identiques mais ne garantissent pas l'absence de redondance. De la même façon, il est possible que des appariements issus de graines différentes soient partiellement recouvrants. Suivant les domaines d'application, on pourra conserver tous les appariements construits

(liste L_{App}) ou ne garder que le meilleur appariement (celui qui a la plus grande valeur de I_π).

3.5 Complexité

L'évaluation de la complexité en temps de cette approche se décompose en deux phases : le calcul de la matrice $\text{INC}^{(k)}$ et celui de l'appariement des variables. La complexité finale de l'algorithme I_π est donc la somme de la complexité de chacune des deux étapes de la méthode. La complexité de chacune d'elles dépend des algorithmes et des caractéristiques utilisées. Nous nous intéressons ici uniquement à des prédicats d'arité 2.

On pose les paramètres suivants :
- V : nombre de variables dans h et ex ;
- L : nombre de littéraux dans h et ex ;
- D : nombre de variables voisines pour une variable donnée dans h et ex (degré des sommets dans les graphes) ;
- X : nombre de littéraux entre deux variables données dans h et ex.

3.5.1 Calcul de la matrice INC

La phase de calcul de la matrice $\text{INC}^{(k)}$ se compose d'une boucle de K itérations pour la résolution du système d'équations qui comporte $K.V^2$ paires de variables (v_i, v'_j) à évaluer.

Pour évaluer les degrés d'inclusion entre chacune des D^2 paires de variables voisines, on compare les deux ensembles de littéraux ce qui nécessite X^2 comparaisons. Cependant, si les littéraux sont construits sur des symboles de prédicats différents, alors la comparaison peut être réalisée en temps constant du fait qu'on ne compare que des littéraux de même signature. La complexité à ce niveau est donc en $O(D^2)$.

On procède ensuite à la recherche du meilleur appariement entre les variables voisines. Il faut distinguer ici le cas des appariements uniques et multiples car les algorithmes utilisés sont différents.

Pour les appariements uniques où l'algorithme hongrois est utilisé, la complexité de cette étape est en $O(D^3)$. Lorsque le nombre de variables voisines à apparier est faible (inférieur à 5), la complexité peut être réduite en pré-calculant tous les couplages possibles entre les variables voisines puis en évaluant la somme des poids de chaque couplage. La recherche du couplage de poids maximal est alors en $O(1)$ car il suffit de rechercher la plus grande somme dans une liste de valeurs.

Pour les appariements multiples, l'utilisation de l'algorithme glouton nécessite, pour chaque variable voisine dans h, de parcourir l'ensemble des variables voisines dans la clause ex ; la complexité est alors en $O(D^3)$.

En fin de compte, pour cette étape, la complexité moyenne de l'algorithme générique est donc en :

- $O(K.V^2.D^3)$ pour les appariements uniques avec l'algorithme hongrois. Si le nombre de variables voisines est faible, cette complexité se simplifie en $O(K.V^2.D^2)$;
- $O(K.V^2.D^3)$ pour les appariements multiples avec l'algorithme glouton.

3.5.2 Appariement global

Nous regardons maintenant la complexité des différents algorithmes de la phase d'appariement global.

3.5.2.1 Algorithme hongrois

Pour apparier les variables de la clause h avec les variables de la clause ex , la complexité de l'algorithme hongrois est en $O(V^3)$. Comme précédemment, elle se réduit à $O(1)$ si l'on pré-calcule l'ensemble des couplages possibles, pour des petites valeurs de V. Cette simplification est très utilisée en chimie où, dans les petites molécules organiques, les atomes ont rarement plus de 4 atomes voisins.

3.5.2.2 Algorithme AMC_non_Guidé

L'algorithme AMC_non_Guidé est composé d'une boucle de max_{Iter} itérations. A chaque itération, deux fonctions sont appelées successivement pour rechercher la variable de h qui génère le maximum de conflits et la variable de ex qui minimise ces conflits. Chaque étape nécessite la comparaison deux à deux de l'ensemble des littéraux de h et ex ce qui donne une complexité en $O(L^2)$. La complexité totale de cette phase est donc en $O(max_{Iter}.L^2)$.

3.5.2.3 Algorithme de recherche d'Arbre Partiellement Couvrant

L'étape élémentaire de cet algorithme est la génération d'un appariement à partir d'une graine donnée. Dans le pire des cas, le parcours en largeur de la clause h visite toutes les variables de la clause. Seul le parcours de la clause h est important car c'est lui qui guide la recherche de l'APC.

Dans le cas d'appariements multiples, à chaque variable visitée, l'ensemble de ses variables voisines est apparié à l'aide d'un algorithme glouton, ce qui demande $O(D^3)$. La complexité est donc en $O(V.D^3)$. Si les appariements sont uniques, le nombre de tests est très inférieur car le nombre de variables restant à apparier, à chaque étape, décroît rapidement.

En utilisant l'ensemble des V^2 graines possibles pour générer des appariements différents, la complexité totale de la recherche d'un APC est en $O(V^3.D^3)$. Cette complexité diminue lorsque la stratégie qui consiste à limiter le nombre de graines est utilisée.

3.6 Conclusion

Pour évaluer le degré d'inclusion entre deux clauses, nous avons proposé une approche heuristique, inspirée d'une mesure de similarité proposée en logique du premier ordre. L'algorithme peut être vu comme un test de subsomption à valeur continue. Comme il est défini à l'aide de procédures et de fonctions génériques, qui peuvent être déterminées en fonction du domaine d'application, il s'applique naturellement à d'autres données structurées comme les graphes, les hypergraphes ou les Problème de Satisfaction de Contraintes (PSC).

Les différentes stratégies implémentées permettent d'identifier des SSC entre deux objets structurés, suivant les contraintes propres à chaque domaine d'application (type d'appariement par exemple). L'algorithme proposé étant basé sur des méthodes heuristiques et stochastiques, il est dit *incomplet* dans le sens où il tend à identifier la plus grande SSC entre deux objets mais ne garantit pas de la trouver.

Le tableau 3.3 donne un récapitulatif des différentes variantes [8] proposées dans ce chapitre.

8. Lorsqu'un paramètre n'est pas déterminé, il est désigné par le caractère '*'.

Groupe	Nom	Matrice INC	Appariement global						
			hongrois	AMC	APC				
					stochastique	glouton	hongrois	Arrêt	
AMC	AMC_Guidé	X		X					
	AMC_non_Guidé			X					
APC_*_AvecArret	APC_Stochast_AvecArret				X				oui
	APC_Glouton_AvecArret	X				X			oui
	APC_Kuhn_AvecArret	X					X		oui
APC_*_SansArret	APC_Stochast_SansArret				X				non
	APC_Glouton_SansArret	X				X			non
	APC_Kuhn_SansArret	X					X		non
	Ipi_Kuhn	X	X						

Tableau 3.3 – Récapitulatif des variantes de l'algorithme I_π. Les 'X' correspondent aux options effectives. NAindique une option non applicable pour la variante en question.

Chapitre 4

Evaluation de I_π dans la transition de phase

Les méthodes d'apprentissage relationnel reposent le plus souvent sur des algorithmes de type *générer et tester* dont l'un des composants principaux est le test de θ-subsomption (ou test de couverture). Dans ce domaine, [Botta *et al.*, 2003] et [Giordana *et al.*, 2000] ont mis en évidence un phénomène appelé Transition de Phase (TP), lié à l'utilisation de la θ-subsomption, mais que l'on retrouve également dans les Problème de Satisfaction de Contraintes (PSC) [Hogg *et al.*, 1996] et d'autres langages de représentation que la logique du premier ordre [Buhot and Gordon, 1998], [Pernot *et al.*, 2005]. Ces travaux montrent que, dans la région de la TP et les régions connexes où la probabilité de solution est faible, les algorithmes d'apprentissage procédant par généralisation ou spécialisation progressive peuvent être inopérant. Une raison de ces échecs est l'absence d'une fonction de coût adéquate, permettant de guider efficacement l'apprentissage dans des régions où la valeur de cette fonction reste constante.

Nous cherchons, dans ce chapitre, à étendre le test de couverture classique en PLI (*i.e.* la θ-subsomption) au moyen du degré d'inclusion I_π qui évalue la couverture partielle entre une hypothèse et un exemple. Cette approche permet de raffiner la réponse du test de θ-subsomption : non seulement elle indique si une hypothèse couvre ou rejette un exemple mais également elle estime, dans le second cas, l'importance de ce rejet.

Par ailleurs, l'efficacité des algorithmes incomplets comme I_π, pour trouver des solutions lorsqu'elles existent, peut être évaluée à l'aide de problèmes satisfiables [Achlioptas *et al.*, 2000], c'est-à-dire des problèmes qui contiennent une solution. C'est pourquoi nous étudions également le comportement et les limites de l'algorithme sur des problèmes satisfiables difficiles situés autour de la TP.

Ce chapitre est organisé de la façon suivante. Dans la section 4.1, nous présentons le phénomène de la TP dans le cadre du test de couverture et ses conséquences sur les

algorithmes d'apprentissage relationnel. Nous observons ensuite le comportement du degré d'inclusion autour de la transition de phase. Dans la section 4.2, nous testons les performances de l'algorithme et nous étudions la qualité des solutions identifiées sur des problèmes satisfiables. Nous terminons par la section 4.3 qui évoque les perspectives de cette étude.

4.1 La transition de phase en apprentissage relationnel

4.1.1 Introduction

Dans les années 1990, la communauté de la Satisfaction de Contraintes, motivée par l'étude des problèmes de complexité computationnelle, s'est intéressée à l'identification des problèmes réellement difficiles [Cheeseman *et al.*, 1991], [Smith, 1994].

Ces travaux ont conduit à étudier un phénomène appelé *transition de phase* [Hogg *et al.*, 1996] qui met en évidence trois catégories (ou régions) de problèmes. Une région OUI qui correspond aux problèmes sous-contraints où la probabilité de satisfiabilité est proche de 1 et la complexité moyenne est faible. Une région NON correspondant à des problèmes sur-contraints où la probabilité de satisfiabilité est proche de 0 et la complexité est faible également. Et une zone entre les régions OUI et NON, appelée la *Transition de Phase (TP)*, où la probabilité de satisfiabilité tombe brutalement de 1 à 0 et qui concentre en moyenne les PSC les plus difficiles.

Dans le domaine des PSC, les travaux autour de la TP se basent sur des analyses statistiques d'instances de problèmes générées aléatoirement [Hogg *et al.*, 1996]. Chaque problème peut être caractérisé par quatre nombres :

- v est le nombre de variables du PSC ;
- c est le nombre de valeurs dans le domaine de chaque variable ;
- p_1 est la probabilité qu'il existe une contrainte entre une paire de variables ;
- p_2 est la probabilité conditionnelle qu'une paire de valeurs soit incohérente pour une paire de variables liées par une contrainte.

Les paramètres p_1 et p_2 sont appelés respectivement la *densité de contraintes* et la *force des contraintes*.

Pour construire une instance aléatoire, il existe plusieurs manières de traiter les probabilités p_1 et p_2. Une possibilité est de sélectionner chacune des $v(v-1)/2$ arêtes possibles dans le graphe de contraintes [9] avec une probabilité p_1. Ensuite, pour chaque paire

9. Le graphe de contraintes d'un PSC est un graphe dans lequel les nœuds sont les variables du PSC et les arêtes correspondent aux contraintes entre deux variables.

de variables liées par une contrainte, l'ensemble des paires de valeurs incohérentes est généré en fixant la valeur FAUX à chaque paire de valeurs avec la probabilité p_2. Cette méthode, appelée modèle A par [Palmer, 1985], génère un ensemble d'instances dans lesquelles le nombre d'arêtes est en moyenne égal à $p_1 v(v-1)/2$ et pour chaque paire de variables contraintes, le nombre moyen de paires de valeurs incohérentes est $c^2 p_2$. Cependant, cette approche présente l'inconvénient de générer des instances avec une très grande variabilité dans le nombre de contraintes et le nombre de paires de valeurs incohérentes.

Le modèle B [Palmer, 1985], dérivé du modèle A, permet de générer des instances en spécifiant précisément (au lieu de la moyenne) le nombre de contraintes ainsi que le nombre de paires de valeurs incohérentes. C'est cette méthode qui est couramment utilisée dans les études sur la transition de phase.

4.1.2 Test de θ-subsomption

Dans ses travaux sur le test de *θ-subsomption* au niveau de la TP, [Botta *et al.*, 2003] se base sur le modèle B pour construire des instances de problèmes en logique du premier ordre.

Une instance d'un test de couverture est une paire (h, ex) de (hypothèse, exemple) caractérisée par le quadruplet (n, m, L, N) où :

- l'hypothèse h contient n variables et m littéraux ($m \geq n-1$) basés sur m symboles de prédicats différents ;
- l'exemple ex contient L constantes et N littéraux instanciés ($N < L^2$) pour chacun des m symboles de prédicats.

Un certain nombre d'hypothèses simplificatrices sont considérées. Tous les prédicats dans le langage sont binaires. L'hypothèse h est une conjonction de littéraux et contient seulement une occurrence de chacun des m symboles de prédicats, ce qui garantit que le PSC correspondant ne puisse être décomposé en plusieurs petits PSC indépendants. Enfin, toutes les variables de l'hypothèse prennent leur valeur dans le même domaine.

L'instance d'un test de couverture est construite de la manière suivante. L'hypothèse h est d'abord construite de façon à être connectée (première partie dans la définition de la clause) puis les $m-n+1$ littéraux manquants sont ajoutés :

$$h \leftarrow \bigwedge_{i=1}^{n-1} r_i(x_i, x_{i+1}) \wedge \bigwedge_{i=n}^{m} r_i(x_j, x_k) \tag{4.1}$$

avec $i \in \{1, \ldots, m\}$ et $j, k \in \{1, \ldots, n\}$. L'hypothèse contient alors exactement n variables et m littéraux, chacun d'eux étant construit sur un symbole de prédicat différent. Les

mêmes paires de variables peuvent apparaître dans plus d'un littéral. Dans l'exemple, pour chaque symbole de prédicat r_i, N paires de constantes (a_{ij_1}, a_{ij_2}) sont sélectionnées sans remplacement dans l'ensemble $A \times A$ des paires de constantes possibles (on a donc $N \leq L^2$) :

$$ex \leftarrow \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{N} r_i(a_{ij_1}, a_{ij_2}) \tag{4.2}$$

Du fait de l'équivalence de représentation entre les clauses Datalog et un PSC, les paramètres standards p_1 et p_2 peuvent être exprimés en termes de n, m, L et N :

$$p_1 = \frac{2m}{n(n-1)} \quad et \quad p_2 = 1 - \frac{N}{L^2}$$

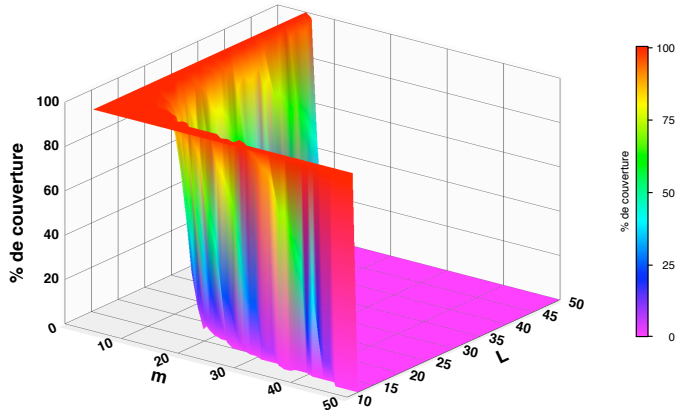
Cependant, dans le cas de l'apprentissage relationnel, on utilise plutôt les paramètres n, m, L et N car, contrairement à p_1 et p_2, ils donnent une interprétation naturelle de la complexité des clauses.

Exemple
Considérons la construction d'une instance de problème défini par le quadruplet $< n = 3$, $m = 5$, $N = 3$, $L = 3>$. La première étape consiste à construire la partie connectée de l'hypothèse qui contient 3 variables et 2 littéraux construits sur les symboles de prédicats r_1 et r_2, ce qui donne par exemple : $h \leftarrow r_1(x_1, x_2), r_2(x_2, x_3)$. A cette partie, il faut ajouter m-n+1 = 3 littéraux basés sur les symboles de prédicats suivants : r_3, r_4 et r_5 ; par exemple, les littéraux $r_3(x_3, x_3)$, $r_4(x_1, x_3)$ et $r_5(x_2, x_1)$. L'hypothèse finale est donc définie par : $h \leftarrow r_1(x_1, x_2), r_2(x_2, x_3), r_3(x_3, x_3), r_2(x_1, x_3), r_5(x_2, x_1)$.
Pour construire l'exemple *ex*, on construit N=3 littéraux pour chacun des m symboles de prédicats en sélectionnant, sans remplacement, $N = 3$ paires de constantes parmi les $L^2 = 9$ possibles.

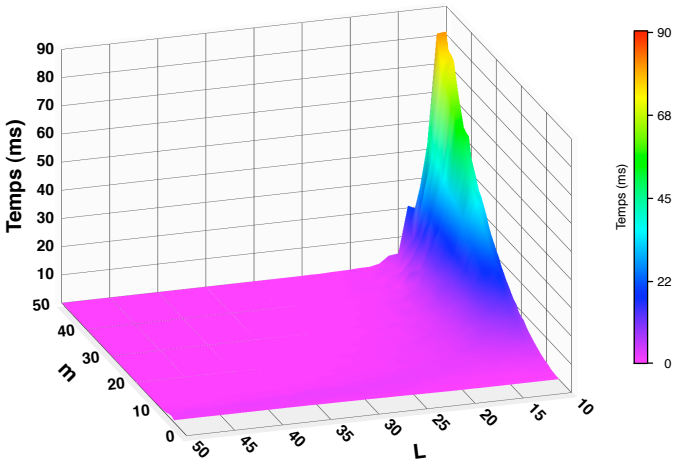
Une analyse systématique de la couverture de *ex* par h dans l'espace quadri-dimensionnel (n, N, m, L) est, en pratique, impossible. Pour conduire les expériences, [Botta *et al.*, 2000] sélectionne donc les variables m et L comme principaux paramètres d'ordre car elles sont directement liées à la complexité de l'hypothèse et de l'exemple, respectivement. Les variables n et N sont considérées comme des paramètres secondaires et sont fixées avec une valeur donnée.

Pour chaque tuple (n, N, m, L) avec $n = 10$ et $N = 100$, 100 instances de problèmes ont été générées et la probabilité de succès P_{sol} a été enregistrée. Les résultats illustrés sur la figure 4.1 ont été obtenus avec l'algorithme Django [Maloberti and Sebag, 2004]. Lorsque les valeurs de la probabilité de couverture P_{sol} sont affichées sur le plan (m, L) (figure 4.1(a)), on observe l'existence d'une transition de phase très raide. Dans

une large région, appelée région OUI, les hypothèses ont une probabilité proche de 1 de couvrir les exemples et dans une large région, appelée région NON, la probabilité qu'une hypothèse couvre un exemple est nulle pour un nombre infini d'exemples.



(a) Probabilité de couverture (exprimée en %)



(b) Coût computationnel

FIGURE 4.1 – (a) Probabilité P_{sol} qu'une hypothèse aléatoire h couvre un exemple aléatoire ex, moyenné sur 100 paires (h, ex), pour chaque point (m, L). Les courbes de niveau, projetées sur le plan (m, L), correspondent à la région de la TP avec une probabilité comprise entre 0.99 et 0.01. (b) Coût computationnel de ce test de couverture.

4.1.3 Impact sur l'apprentissage relationnel

De nombreux algorithmes en PLI sont basés sur le test de subsomption. Le principe de ces algorithmes est le suivant. A partir d'un ensemble d'exemples et de contre-exemples (des clauses respectivement étiquetées positives et négatives), on cherche à

construire une (ou plusieurs) clauses générales appelées hypothèses qui couvrent les exemples et ne couvrent pas (rejettent) les contre-exemples.

Depuis [Mitchell, 1982], l'apprentissage symbolique est défini comme une recherche dans l'espace des hypothèses qui est structuré de l'hypothèse la plus générale (l'hypothèse vide) à la plus spécifique grâce au test de θ-subsomption qui introduit une notion de généralité entre les clauses. Ainsi, les algorithmes de type *top-down* construisent une hypothèse, littéral par littéral, en commençant par l'hypothèse vide. A chaque itération, l'algorithme génère un ensemble de n hypothèses candidates $\{h'_1, \ldots, h'_n\}$ composées de l'hypothèse courante h auxquelles un littéral différent a été ajouté. Dans le plan (m, L), ce type d'approche construit donc une hypothèse en partant de la gauche (hypothèse vide) vers la droite, au fur et à mesure que le nombre de littéraux dans l'hypothèse courante (*i.e.* le paramètre m) est incrémenté à chaque itération. Le problème consiste alors à sélectionner, à chaque itération, la meilleure hypothèse candidate h'_i au sens d'une fonction d'évaluation.

Cependant, dans la plupart des cas, ces fonctions sont basées sur deux paramètres principaux qui sont le taux d'exemples couverts et le taux de contre-exemples rejetés [Fürnkranz and Flach, 2005]. Comme dans la région OUI, une hypothèse couvre n'importe quel exemple ou contre-exemple, les fonctions de coût renvoient une valeur constante, quelle que soit la taille de l'hypothèse. Ainsi, lors de la traversée de ces régions, les algorithmes ne disposent d'aucune valeur informative pour sélectionner une hypothèse parmi toutes les hypothèses candidates. L'exploration de l'espace de recherche dans ces régions est alors "aveugle". C'est ce qu'a montré [Botta *et al.*, 2003] en réalisant des expériences systématiques avec plusieurs algorithmes d'apprentissage (FOIL, G-Net, Smart+ et PROGOL) sur un ensemble de problèmes d'apprentissage aléatoires.

Plus récemment, [Alphonse and Osmani, 2008] ont repris les mêmes problèmes d'apprentissage avec l'algorithme PROPAL [Alphonse and Rouveirol, 2006] qui est un algorithme top-down dirigé par les données. C'est-à-dire qu'au lieu d'évaluer une fonction de coût qui ne tient compte que de la structure de l'espace des hypothèses, cette approche utilise directement les instances d'exemples et de contre-exemples pour guider l'apprentissage. Un exemple positif appelé graine est sélectionné et l'algorithme recherche des hypothèses plus générales que la graine dans l'espace des hypothèses en utilisant des contre-exemples pour élaguer les branches non pertinentes.

Dans cette expérience, les auteurs n'ont sélectionné que les contre-exemples qui diffèrent de la graine par un seul littéral (un littéral de la graine qui n'est pas dans le contre-exemple). En utilisant ces contre-exemples particuliers appelés *near-misses* par [Winston, 1975], l'algorithme a bien réussi à résoudre les problèmes d'apprentissages. Ces résultats montrent que la présence de la transition de phase n'impacte que

les algorithmes basés sur une fonction d'évaluation (qui ne tient compte que du nombre d'exemples couverts et de contre-exemples rejetés) pour guider la recherche et n'influence pas les stratégies dirigées par les données en utilisant les near-misses.

4.1.4 Comportement du degré d'inclusion

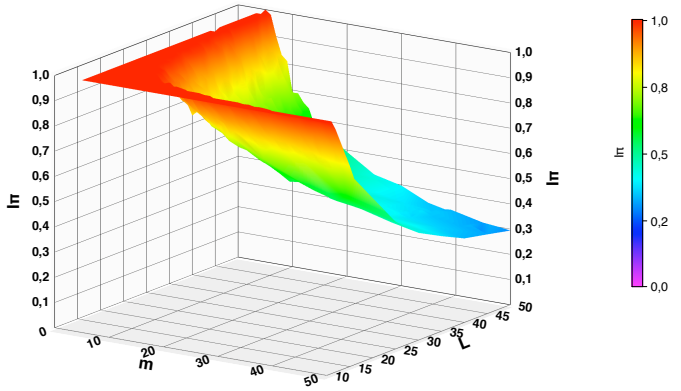
Pour évaluer le comportement d'un degré d'inclusion entre clauses dans le contexte de la TP, nous remplaçons le test de couverture classique qui donne une réponse booléenne par l'algorithme I_π qui renvoie une valeur réelle dans l'intervalle [0,1] [Wieczorek *et al.*, 2006b], [Wieczorek *et al.*, 2006a].

Nous reprenons le même cadre expérimental que précédemment en explorant les points du plan $< m, L >$ pour des valeurs de $m \in [10, 50]$, $L \in [10, 50]$ pour $n = 6$ et $N = 100$. Pour chaque paire (m, L), nous générons également 100 instances (h, ex) dont nous évaluons le degré d'inclusion $I_\pi(h, ex)$. Pour cette expérience, la recherche des solutions est réalisée par l'algorithme AMC_Guidé (section 3.3.2) en fixant le nombre d'itérations max_{Iter} à 100. Les résultats sont présentés sur la figure 4.2.

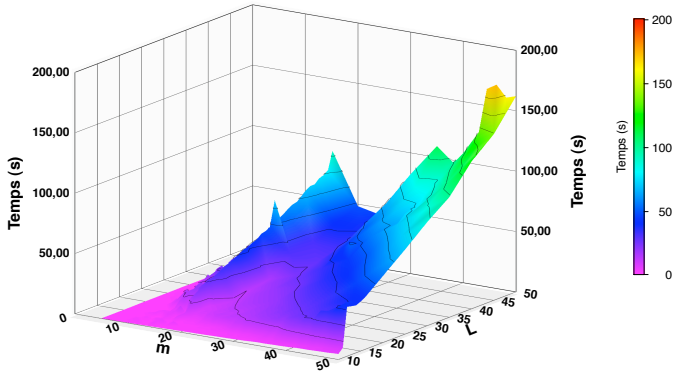
On observe sur la figure 4.2(a) un plateau qui correspond à un degré d'inclusion $I_\pi = 1$, ce qui est cohérent avec le fait que les problèmes sont situés dans la région OUI. En effet, dans cette région, les problèmes sont faciles et satisfiables : l'algorithme trouve une solution complète.

Dans la région NON, les instances n'ont pas de solution complète (aucune hypothèse ne couvre aucun exemple), c'est pourquoi le test de couverture classique renvoie une réponse toujours égale à FAUX. Il existe cependant des solutions partielles où, pour un problème donné, une sous-expression de l'hypothèse h couvre un exemple ex : ce sont ces solutions partielles que l'algorithme I_π identifie. La valeur du degré d'inclusion I_π étant proportionnelle à la taille de la solution partielle, dans cette région, elle est inférieure à 1.

On observe également une décroissance régulière de l'indice I_π : près de la TP, les solutions partielles sont grandes (proches d'une solution complète) et leur taille diminue au fur et à mesure que l'on s'en éloigne. Contrairement aux algorithmes complets pour lesquels la complexité reste faible dans la région NON, on observe ici que le temps de calcul augmente de façon exponentielle avec la taille de l'hypothèse et de l'exemple. On remarque également que, pour de petites hypothèses ($m \approx 15$) et des exemples de grande taille ($L > 30$), la complexité est élevée.



(a) Indice I_π



(b) Coût computationnel

FIGURE 4.2 – Evaluation du degré d'inclusion I_π autour de la transition de phase. (a) : moyenne des degrés d'inclusion I_π dans le plan $< m, L >$, calculés avec l'algorithme AMC_Guidé. (b) : moyenne des temps de calcul exprimés en secondes, sur l'ensemble des 100 instances.

Ce degré d'inclusion fait donc apparaître un gradient dans la région NON au lieu du plateau mis en évidence par le test de subsomption. Dans le cadre d'une méthode d'apprentissage, si ce gradient est correctement orienté, il pourrait permettre d'effectuer le bon choix de littéraux à ajouter en spécialisation ou à retirer en généralisation en les sélectionnant non seulement sur le nombre d'exemples couverts et de contre-exemples rejetés mais également sur la proximité (ou la dissemblance) structurelle de chaque hypothèse h_j par rapport aux exemples (ou aux contre-exemples).

Dans les PSC, cette problématique est déjà abordée sous le nom de satisfaction de contraintes partielle [Freuder and Wallace, 1992] ; par contre, il n'y a, à notre connaissance, que peu d'approches similaires en PLI.

4.2 Etude de l'algorithme I_π dans la transition de phase

La méthode que nous proposons est basée sur une heuristique de recherche locale et ne garantit pas de trouver une solution optimale au problème (*i.e.* une solution complète dans la région OUI ou la plus grande solution partielle dans la région NON). Dans l'expérience précédente, nous avons vu que l'algorithme I_π identifiait des solutions partielles (caractérisées par $0 < I_\pi < 1$) dans la région NON mais il n'existe pas de moyen de savoir si, pour chaque instance, c'est la plus grande solution qui a été identifiée (du fait que c'est une heuristique et que la limite de temps - le paramètre max_{Iter} - n'est pas infinie).

L'efficacité de ce type d'algorithmes incomplets peut être évaluée à l'aide d'instances difficiles satisfiables [Achlioptas *et al.*, 2000], c'est-à-dire des instances dont on sait qu'elles contiennent une solution complète (*i.e.* une SSC connue comme existante entre une hypothèse h et un exemple ex). Dans ce cas, une méthode complète retrouverait systématiquement cette solution et les meilleurs algorithmes incomplets sont ceux qui retrouvent la plus grande solution partielle (dans notre cas, la plus grande valeur de l'indice I_π).

Nous allons donc réitérer les expérimentations avec, cette fois, des instances de problèmes satisfiables.

4.2.1 Construction des problèmes satisfiables

Un problème satisfiable est caractérisé par le fait qu'il contient une solution : dans le cas des clauses Datalog, cela signifie qu'un exemple ex contient un sous-ensemble de littéraux (*i.e.* la solution cachée) qui est subsumé par une hypothèse h. Pour construire de telles instances difficiles, les générateurs comme le modèle B ne peuvent pas être utilisés car, au niveau de la TP (là où les instances les plus difficiles sont concentrées), ces modèles produisent généralement un mélange d'instances satisfiables et non satisfiables et dans la région NON, les problèmes ne sont en moyenne pas satisfiables.

Dans ce contexte, [Xu and Li, 2006], [Xu *et al.*, 2005] proposent une stratégie pour générer des instances satisfiables dont la difficulté est similaire aux instances du modèle B. L'approche consiste à construire d'abord une clause connectée aléatoire qui correspond à la solution que l'on souhaite cacher puis à ajouter à cette clause un certain nombre de littéraux supplémentaires. En nous basant sur cette stratégie, nous construisons une instance satisfiable de la manière suivante :

1. l'hypothèse h connectée est construite avec n variables et m littéraux ;
2. l'exemple ex est initialisé avec l'hypothèse ;
3. L–n variables supplémentaires sont adjointes à l'exemple ex ;

4. pour chacun des *m* symboles de prédicats, *N*–*m* littéraux aléatoires sont ajoutés à l'exemple *ex*.

Exemple
Les étapes de la construction du problème satisfiable, défini par $< n = 3, m = 3, L = 4, N = 4 >$ sont les suivantes :

1. $h \leftarrow r_1(x,y), r_2(y,z), r_3(x,z)$;
2. $ex \leftarrow r_1(x,y), r_2(y,z), r_3(x,z)$;
3. la variable *t* est ajoutée à l'exemple ;
4. $ex \leftarrow \boldsymbol{r_1(x,y), r_2(y,z), r_3(x,z)}, r_1(x,x), r_2(t,z), r_3(x,t)$ où les littéraux en gras correspondent à la solution cachée.

4.2.2 Evaluation des variantes de I_π

Dans cette étude, nous comparons plusieurs variantes de l'algorithme I_π. Deux variantes de la recherche d'un APC sont utilisées : l'algorithme APC_Stochast_* où l'appariement local entre les ensembles de voisins est réalisé de manière aléatoire (sans tenir des informations contenues dans la matrice INC) et l'algorithme APC_Glouton_* où cet appariement est pris en charge par un algorithme glouton. Les appariements entre variables étant de type n-1, il n'est pas possible d'utiliser l'algorithme hongrois.

Bien que, par construction, l'hypothèse soit connectée (*i.e.* correspondant à un graphe connexe), deux variantes de l'AMC sont également testées (nous rappelons que cet algorithme ne garantit pas de trouver un appariement connexe). La première variante est l'algorithme AMC_Guidé qui apparie toutes les variables à l'étape d'initialisation puis modifie l'appariement d'une variable voisine en cherchant à minimiser le nombre de conflits tout en maximisant la valeur de I_π. La deuxième variante est l'algorithme AMC_non_Guidé est similaire à AMC_Guidé à la différence qu'il n'utilise pas la matrice INC pour guider la recherche d'une solution : son objectif est uniquement la minimisation du nombre de conflits.

Les valeurs de l'indice I_π le long de la droite $m = L$ ont été calculées ($m, L \in [10, 30]$). Les résultats pour n=6 et N=100 sont illustrés sur la figure 4.3. Sur cette ligne, la transition de phase se trouve autour de $m = L \approx 17 \pm 1$. On constate que la méthode APC_Stochast_* ne trouve la solution que pour $m = L = 10$ et échoue dès que la taille de l'hypothèse (et donc de la solution) et de l'exemple augmentent. La variante AMC_non_Guidé (100 et 500 itérations) ne trouve la solution que dans une fraction des problèmes (qui dépend du nombre d'itérations max_{Iter}) non seulement dans la région de la transition de phase, mais également au-delà (en fait, $I_\pi < 1$ dès que $L = m > 15$).

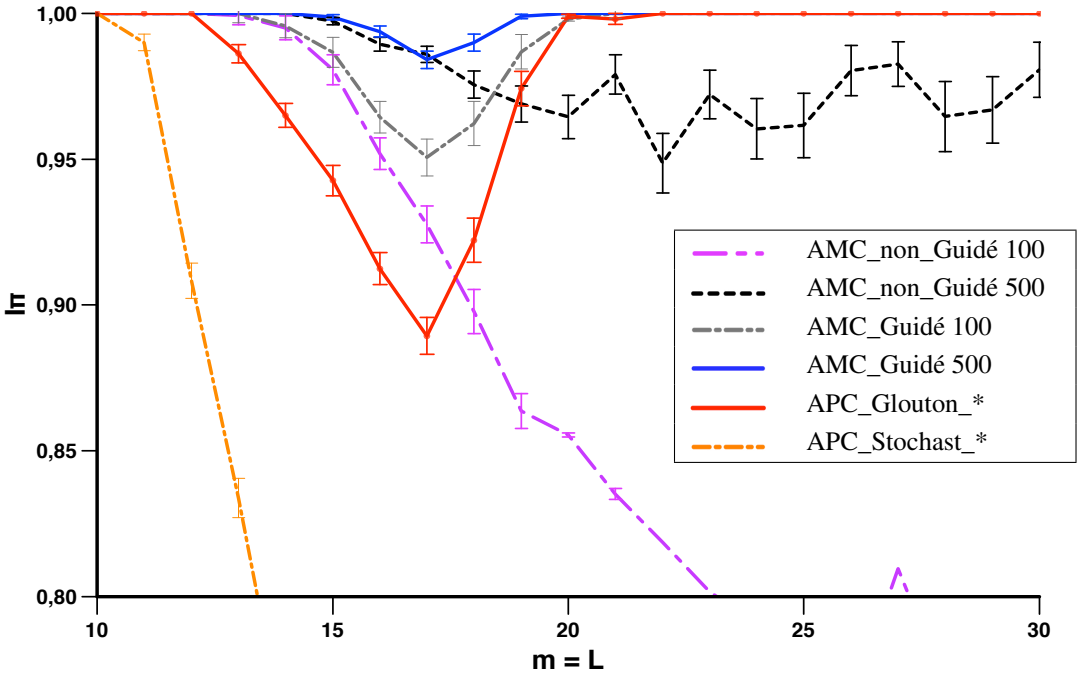


FIGURE 4.3 – Comparaison des variantes de l'algorithme I_π sur des instances satisfiables, sur la droite $m = L$. Les nombres 100 et 500 figurant à côté des variantes de l'AMC correspondent au nombre d'itérations max_{Iter}.

Les autres algorithmes trouvent la solution dans la région OUI et dans la région NON mais échouent dans la région de la TP ; la largeur de la zone où $I_\pi < 1$ et la valeur de I_π dépendent très sensiblement de l'algorithme. L'algorithme APC_Glouton_* est moins performant que la méthode AMC_non_Guidé dans la région OUI, mais il retrouve la solution dans la région NON dès qu'on s'éloigne suffisamment de la transition de phase ($L = m > 20$).

Finalement, la méthode la plus satisfaisante est AMC_Guidé : avec $max_{Iter} = 500$ itérations, elle est la plus performante en termes de I_π pour toutes les valeurs de m et L étudiées (elle retrouve en moyenne plus de 97% de la solution complète). C'est aussi la méthode pour laquelle la région de défaillance est la moins étendue : sa largeur est équivalente à l'indétermination dans la région de la transition ($16 \leq L = m \leq 18$). Cependant, nous observons que, pour un nombre de variables plus grand ($n = 10$ et $n = 14$, non présentés ici), la largeur et la profondeur de la dégradation augmentent.

On peut remarquer que, pour les variantes qui présentent une zone de défaillance au niveau de la TP (APC_Glouton_* et AMC_Guidé 100), la dégradation n'est pas toujours symétrique par rapport à la TP (figure 4.3). Pour APC_Glouton_* et AMC_Guidé 100, la dégradation commence à $m = L = 12$ et se termine à $m = L = 20$. Autour de la TP, ces

algorithmes ont plus de difficulté à trouver la solution dans la région OUI que dans la région NON.

De manière plus générale, on peut se demander pourquoi la solution cachée est retrouvée dans la région NON alors qu'elle ne l'est pas dans la région de la TP. En effet, dans le cas d'un algorithme incomplet, la complexité augmente avec la taille de l'espace de recherche et on se serait attendu à ce que la solution soit moins bien retrouvée dans cette région.

Pour répondre à ces questions, nous détaillons les solutions identifiées par l'algorithme. Dans la région OUI, lorsqu'une solution complète a été trouvée, on vérifie expérimentalement que ce n'est pas systématiquement la solution qui avait été cachée. En effet, on sait que, dans cette région, de nombreuses solutions existent et l'algorithme en trouve une parmi celles-ci. Par contre, dans la région NON, il existe très peu de solutions et l'algorithme retrouve bien la solution cachée. D'après [Smith, 1994], on peut définir le nombre de solutions attendu $E(Nb_{Sol})$ dans une instance du modèle B, par :

$$E(Nb_{Sol}) = L^n\left(\frac{N}{L^2}\right)^m \tag{4.3}$$

c'est-à-dire le nombre d'appariements possibles de L valeurs sur n variables, multiplié par la probabilité qu'un appariement choisi aléatoirement soit cohérent. Si $N = L^2$, il n'y a aucune paire de valeurs incohérente et $E(Nb_{Sol}) = m^n$. Quand $N = 0$, il n'y a pas de solutions. Le nombre de solutions décroît très rapidement de m^n à 0 lorsque N/L^2 croît.

On peut relier le nombre de solutions à la présence de valeurs égales à 1 dans la matrice INC. En effet, une solution complète est caractérisée par des '1' dans la matrice pour les paires de variables appariées et un degré d'inclusion égal à 1. En examinant le contenu de la matrice INC pour différentes valeurs de m et L (voir les matrices de la figure 4.4), on constate effectivement que la proportion de valeurs égales à 1 n'est pas identique, comme le montrent les trois matrices ci-dessous.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \end{pmatrix}$$

(a) Matrice pour m=L=10

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 \\ 0.9 & 1 & 0.9 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0.9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 1 & 0.9 & 1 & 1 & 0.9 & 1 & \mathbf{1} & 0.9 & 1 & 1 & 1 & 0.9 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 1 \\ \mathbf{1} & 1 & 0.9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.9 & 1 & 1 & 0.9 & 1 \end{pmatrix}$$

(b) Matrice pour m=L=17

$$\begin{pmatrix} \mathbf{1} & 0.8 & 0.9 & 0.9 & 1 & 0.8 & 0.9 & 0.9 & 0.9 & 0.9 & 0.8 & 0.8 & 0.9 & 0.8 & 0.9 & 0.8 & \dots \\ 0.8 & \mathbf{1} & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.8 & 1 & 0.8 & 0.9 & 0.9 & 0.9 & 0.9 & 0.8 & 1 & \dots \\ 0.9 & 0.8 & \mathbf{1} & 0.9 & 0.9 & 0.9 & 0.8 & 0.8 & 0.9 & 0.9 & 0.8 & 0.9 & 0.8 & 0.8 & 0.9 & 0.9 & \dots \\ 0.9 & 0.8 & 0.9 & \mathbf{1} & 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.9 & 0.9 & 0.9 & 0.9 & 0.8 & 0.9 & 0.8 & \dots \\ 0.8 & 0.8 & 0.7 & 0.8 & \mathbf{1} & 0.8 & 0.9 & 0.8 & 0.8 & 0.9 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & \dots \\ 0.9 & 0.9 & 0.9 & 0.9 & 0.8 & \mathbf{1} & 0.8 & 0.9 & 0.9 & 0.9 & 1 & 0.9 & 0.9 & 0.8 & 0.9 & 0.9 & \dots \end{pmatrix}$$

(c) Matrice pour m=L=25

FIGURE 4.4 – Exemple de matrices INC pour différentes tailles d'hypothèse et d'exemple. (a) La matrice contient 100% de 1 et la valeur de I_π entre les deux clauses est égale à '1'. (b) La matrice contient 80% de 1 et la valeur de I_π entre les deux clauses est égale à 0.9. (c) La matrice contient 4% de 1 et la valeur de I_π entre les deux clauses est égale à 1. Les valeurs en gras correspondent aux appariements réalisés entre les variables de l'hypothèse et de l'exemple.

La figure 4.5 ci-dessous montre le pourcentage de '1' dans la matrice INC sur la droite $m = L$ pour des valeurs comprises entre 10 et 30. On voit que, dans la région OUI, les matrices contiennent une grande proportion de 1 alors qu'il y en a très peu dans la région NON. La proportion de 1 chute brutalement au niveau de la TP.

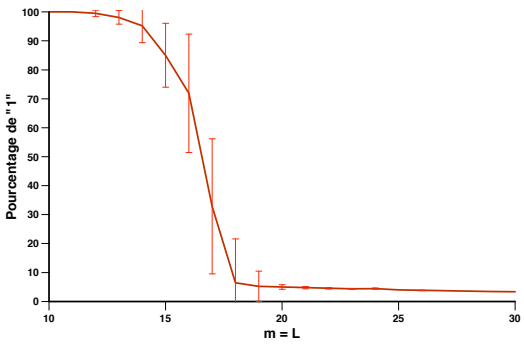


FIGURE 4.5 – Pourcentage du nombre de 1 dans la matrice INC en fonction des paramètres $m = L$.

Dans la matrice INC, les valeurs égales à 1 signifient que, du point de vue de la recherche locale, pour chaque variable concernée dans *h*, il existe de nombreuses constantes candidates à l'appariement dans *ex*. Dans la région OUI, il y a donc de très nombreuses possibilités d'appariements et la recherche d'une solution complète semble *a priori* facile. En détaillant les solutions identifiées dans la zone de transition de phase, on voit que même si l'algorithme choisit des appariements avec une valeur de 1, le degré d'inclusion final n'est pas toujours égal à 1. Dans ces cas, il reste effectivement des conflits : l'algortihme n'arrive pas à trouver de solution complète au bout des max_{Iter} itérations.

En fait, plus la matrice contient des '1' (comme dans la région OUI), plus les choix d'appariements sont nombreux. Alors que dans le cas de la région NON, il y a finalement très peu de choix et l'appariement est trouvé rapidement car il est mieux guidé par les valeurs significatives de la matrice.

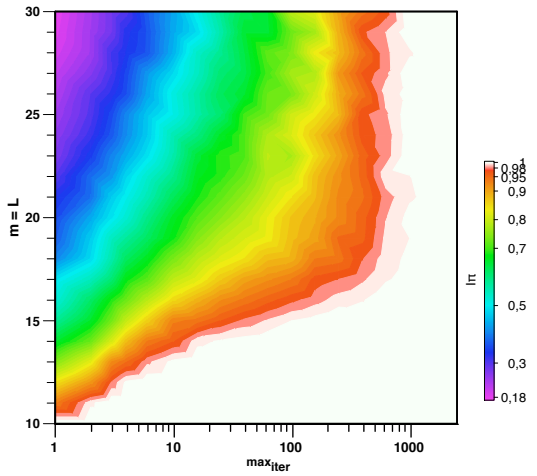
Deux phénomènes peuvent expliquer cette grande proportion de 1. La présence de plusieurs solutions dans la région OUI augmente effectivement le nombre de 1 dans la matrice. Nous avons vérifié expérimentalement l'existence de plusieurs solutions complètes dans cette région et celle qui est identifiée par l'algorithme n'est pas forcément celle qui a été initialement cachée.

On peut également relier cela au fait que la recherche du meilleur appariement local entre les variables voisines sur-évalue la valeur d'inclusion et donc, la matrice se remplit encore plus de '1' qui ne sont donc pas toutes cohérentes avec les structures réelles.

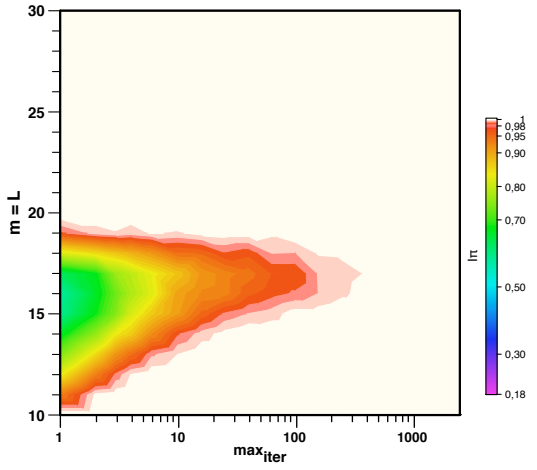
4.2.3 Approfondir l'exploration de l'espace de recherche

Dans la recherche de la solution cachée, le fait que $I_\pi < 1$ dans la région de la TP est dû à la forte concentration de problèmes difficiles dans cette région, ce qui rend notre heuristique moins efficace pour explorer l'espace de recherche. Pour améliorer ces résultats, on peut alors parcourir de façon plus poussée l'espace de recherche en augmentant le nombre d'itérations max_{Iter} dans les algorithmes AMC_Guidé et AMC_non_Guidé. C'est ce que nous avons réalisé dans l'expérience suivante où le nombre d'itérations varie de 100 à 2500 pour étudier son influence sur les performances de ces deux algorithmes. Les résultats sont présentés dans la figure 4.6.

On observe que, dans la région OUI (valeurs de $m \leq 16$), le comportement des deux algorithmes est sensiblement le même : la matrice INC contenant une grande proportion d'éléments égaux à 1, l'appariement est faiblement guidé par la matrice INC. Dans les deux cas, un nombre d'itérations généralement inférieur à 100 permet de retrouver une des solutions complètes.



(a) Variante unGuided



(b) Variante guided

FIGURE 4.6 – Evolution de l'indice I_π en fonction des paramètres $m = L$ et du nombre d'itérations max_{Iter} ; l'échelle en abscisse est logarithmique. (a) Algorithme AMC_non_Guidé. (b) Algorithme AMC_Guidé. Les lignes en pointillés délimitent la zone de transition de phase de la figure 4.3.

Dans la région de la TP ($16 \leq m = L \leq 18$), les deux algorithmes deviennent moins performants. Par contre, l'algorithme guidé par la matrice INC s'avère meilleur que la version non guidée dans le sens où d'une part, il trouve une solution complète ($I_\pi = 1$) pour un nombre d'itérations toujours inférieur à celui qui est nécessaire à l'algorithme non guidé et d'autre part, pour un nombre d'itérations donné, la solution trouvée est meilleure (valeur de I_π plus grande) que celle de l'algorithme AMC_non_Guidé.

Enfin, dans la région NON ($m = L \geq 18$), le comportement des deux algorithmes est totalement différent : l'algorithme AMC_non_Guidé n'est capable de retrouver l'hypothèse qu'avec un nombre d'itérations de l'ordre de 1200. A l'inverse, l'algorithme

AMC_Guidé retrouve toujours la solution cachée et ce, de plus en plus rapidement ; pour des valeurs $m = L \geq 20$, la solution est retrouvée directement avec l'état initial. Ceci s'explique par le fait que, dans cette région, la matrice INC contient moins de valeurs égales à 1 ce qui améliore beaucoup leur significativité.

4.3 Conclusion

Le comportement de plusieurs variantes de l'algorithme I_π a été étudié sur différentes régions de la TP, dans le cas du test de couverture. Ce paradigme est intéressant car il propose un ensemble d'instances faciles et difficiles, satisfiables et non satisfiables, ce qui constitue un excellent jeu de données pour tester les performances des algorithmes.

Dans la région OUI où les instances aléatoires sont faciles et satisfiables, l'algorithme trouve bien une solution complète tandis que dans la région NON, il identifie des solutions partielles (pour des instances aléatoires, il n'existe en effet pas de solution complète dans cette région). Plus les instances s'éloignent de la région de la TP, plus les solutions partielles sont petites et comme l'indice I_π quantifie la taille des solutions, sa valeur décroît. Le saut brutal que l'on observe avec la θ-subsomption au niveau de la TP n'existe donc plus.

Des instances de problèmes contenant une solution cachée ont été utilisées pour observer les limites de l'algorithme et déterminer les variantes les plus performantes. Dans la région OUI qui contient de nombreuses autres solutions, l'algorithme retrouve facilement une solution complète. Dans la région NON, l'exploration de l'espace de recherche est rendue plus efficace lorsque l'algorithme est guidé par les informations de la matrice INC. Dans la région de la TP, il est nécessaire de parcourir l'espace de recherche de manière plus poussée pour trouver une solution ceci en raison de la difficulté des problèmes dans cette zone.

Ce travail propose une heuristique intéressante pour guider l'apprentissage de concepts et notamment dans les régions où les algorithmes d'apprentissage comme FOIL échouent. En effet, dans ces régions, les hypothèses candidates sont suffisamment spécialisées pour couvrir les exemples et rejeter la majorité des contre-exemples : l'algorithme s'arrête alors. Si l'on souhaite poursuivre la spécialisation de l'hypothèse, alors le choix du nouveau littéral se fera de manière aléatoire car les indices de couverture des littéraux candidats auront la même valeur. Dans ce contexte, l'indice I_π pourrait être utilisé pour définir une mesure de couverture qui tienne compte de la proximité de l'hypothèse courante avec les exemples et contre-exemples. De la sorte, il serait possible de choisir les hypothèses candidates qui rejettent en priorité les contre-exemples

les plus proches des exemples (*i.e.* les near-misses). Cette approche rejoint, d'une manière différente, celle qui est proposée par [Alphonse and Osmani, 2008].

De manière complémentaire, il devrait être également possible d'apprendre des définitions "approximatives" de concepts en essayant de construire des clauses qui restent les plus similaires possible (en fait, les plus incluses possible) aux exemples, tout en maximisant la distance avec les contre-exemples. La clause ainsi produite ne serait évidemment pas entièrement opérationnelle (dans le sens où l'unificatinon avec certains exemples peut échouer), mais dans des domaines comme la chimie où l'utilisateur souhaite visualiser des structures (scaffold) les plus complètes possibles, cette démarche permettrait de construire des modèles pertinents pour une interaction homme-machine.

Deuxième partie

Application à la chimie

Chapitre 5

Similarité structurale entre molécules

La similarité moléculaire est un domaine de recherche étudié depuis plusieurs dizaines d'années et en constante évolution [Nikolova and Jaworska, 2003]. Nous nous intéressons dans ce chapitre à l'évaluation de la similarité entre molécules sans connaissance *a priori*, c'est-à-dire en comparant la structure des molécules sans introduire d'informations sur la significativité chimique des parties des molécules. Dans ce domaine, la notion de similarité structurelle est généralement basée sur l'hypothèse que deux molécules dont les structures sont similaires ont probablement des propriétés physico-chimiques voisines ainsi qu'une activité similaire [Johnson and Maggiora, 1990]. Elle prend donc toute son importance dès lors que l'on cherche à prédire l'activité des molécules dans des applications comme le criblage virtuel ou l'analyse de criblages à haut débit par exemple dans l'industrie pharmaceutique ou en chimie médicinale [Sheridan, 2002], [Schuffenhauer *et al.*, 2000].

L'algorithme proposé étant basé sur une comparaison des structures des objets, nous nous focalisons sur cet aspect dans la comparaison avec d'autres méthodes. De plus, pour s'assurer que toutes les méthodes comparées prennent en compte les mêmes informations, nous restreignons volontairement la représentation des molécules à la structure la plus simple, c'est-à-dire la structure 2D des molécules où seuls sont utilisés le type d'atome et la valence des liaisons.

Depuis quelques années, l'intérêt grandissant pour les machines à vecteurs support [Cristiani, 2000], [Scholkopf *et al.*, 2004] en raison de leurs performances en classification supervisée a fortement motivé le développement de fonctions noyau sur des graphes. De plus en plus d'applications de ces fonctions sont publiées dans le domaine de la bioinformatique : [Ben-Hur and Noble, 2005], [Kashima *et al.*, 2004], [Ceroni *et al.*, 2007], [Swamidass *et al.*, 2005], [Qiu *et al.*, 2007] et en chemo-informatique [Azencott *et al.*, 2007], [Mohr *et al.*, 2008], [Mahé *et al.*, 2006].

Dans la section 5.1, nous présentons quelques mesures de similarité de deux molécules basées sur la notion de Sous-Structure Commune Maximale (SSCM). Ces méthodes présentent certains inconvénients que nous adressons dans la section 5.2 en proposant une mesure de similarité basée sur le degré d'inclusion entre deux données structurées. Des mesures basées sur une représentation propositionnelle des molécules sont décrites dans la section 5.3. Enfin, dans la section 5.4, nous nous intéressons aux fonctions noyau sur des graphes.

5.1 Distance basée sur la plus grande Sous-Structure Commune

Comme nous l'avons évoqué dans le chapitre 1, la structure 2D d'une molécule peut être représentée sous forme de graphe moléculaire, c'est-à-dire comme un graphe étiqueté, non orienté dont les sommets correspondent aux atomes de la molécule et les arêtes aux liaisons covalentes. Chaque sommet peut être étiqueté par les propriétés des atomes (type d'atome, charge électronique, ...) et chaque arête par les caractéristiques des liaisons (valence, liaison aromatique, ...).

La taille de la SSCM (au sens du nombre d'atomes) entre deux graphes moléculaires peut donner une idée intuitive de leur similarité [Cao *et al.*, 2008] : plus les molécules partagent une grande SSCM, plus elles sont ressemblantes. Depuis [Cone *et al.*, 1977], ce concept est à la base d'une grande variété d'applications en chimie et en biologie [Hagadone, 1992], [Willett, 1999].

Différents types de SSC peuvent être définies entre deux graphes : par exemple, des SSCM connexes [Bayada *et al.*, 1992], [Takahashi *et al.*, 1987] qui correspondent au plus grand sous-graphe connexe commun à deux graphes donnés ou des SSCM non connexes [McGregor and Willett, 1981]. Nous nous focalisons ici sur les SSCM induites et connexes entre deux graphes moléculaires (voir la section 2.1.3 du chapitre 2), c'est-à-dire celles dont l'ensemble des sommets est inclus dans l'ensemble de sommets de chacun des graphes.

Il existe une grande variété d'algorithmes de recherche de la SSCM [Conte *et al.*, 2004] ; ce problème est NP-complet pour des graphes quelconques. Un exemple d'approche classique en chimie consiste à transformer le problème en recherche de la clique maximale [10] dans le graphe dual entre deux graphes donnés [Barker *et al.*, 2006], [Raymond *et al.*, 2002a], [Raymond *et al.*, 2002b].

10. Une clique est un graphe dans lequel l'ensemble des sommets sont deux-à-deux adjacents

Dans le cas des graphes moléculaires, la complexité des algorithmes peut être réduite en utilisant une connaissance particulière sur ces données. D'abord, le fait de représenter une molécule par un graphe dont les sommets sont étiquetés réduit considérablement les possibilités d'appariement entre atomes des deux molécules (dans le cas d'appariements exacts, on n'apparie que des atomes identiques). De plus, le nombre d'atomes voisins pour un atome donné est borné par une petite constante (généralement au plus 4, rarement au-delà). Par ailleurs, les temps de calcul sont également réduits car les petites molécules organiques sont des graphes de petite taille et comptent en moyenne entre 40 et 60 atomes.

La SSCM est ensuite utilisée comme base pour évaluer la similarité entre les molécules. Plusieurs coefficients de similarité [Bunke and Shearer, 1998] ont ainsi été proposés à partir de la taille (en nombre d'atomes) de la SSCM par rapport à la taille des molécules m_1 et m_2 :

$$S_1(m_1, m_2) = \frac{|SSCM(m_1, m_2)|}{max(|m_1|, |m_2|)} \tag{5.1}$$

$$S_2(m_1, m_2) = \frac{|SSCM(m_1, m_2)|}{min(|m_1|, |m_2|)} \tag{5.2}$$

Ces mesures dérivent directement de la mesure de similarité de Tversky (équation 2.3). [Raymond *et al.*, 2002a] propose également la mesure suivante :

$$S_3(m_1, m_2) = \frac{\Big(n_a(ssc) + n_l(ssc)\Big)^2}{\Big(n_a(m_1) + n_l(m_1)\Big).\Big(n_a(m_2) + n_l(m_2)\Big)} \tag{5.3}$$

où n_a (resp. n_l) correspond au nombre d'atomes (resp. de liaisons) et ssc représente la SSCM entre les molécules m_1 et m_2. Des algorithmes basés sur une distance construite à partir de la SSCM [Raymond and Willett, 2002], [Ruiz *et al.*, 2005] sont de plus en plus utilisés pour analyser les résultats de criblages à haut débit mais s'avèrent encore très lents sur de grandes bases de composés.

De plus, les mesures de similarité présentées ci-dessus ont l'inconvénient d'être sensibles à la taille et à la complexité des molécules ; c'est ce qu'illustre l'exemple de la figure 5.1 où la molécule m est comparée aux molécules m_1, m_2 et m_3. D'après l'équation 5.1, on a :

- $S_1(m, m_1) = 9/15 = 0{,}6$;
- $S_1(m, m_2) = 9/15 = 0,6$;
- $S_1(m, m_3) = 9/25 = 0,36$.

La courbe illustrée montre l'évolution de la mesure de similarité $S_1(m, m')$ en fonction de la taille de la molécule source m'. Nous avons repris l'exemple illustré sur la figure 5.1 en fixant la taille de la SSCM à 9 et la taille de la molécule de référence m à 15 puis en faisant varier la taille de la molécule m' : la courbe correspond donc à la fonction $S_1(m, m') = 9/\max(15, |m'|)$.

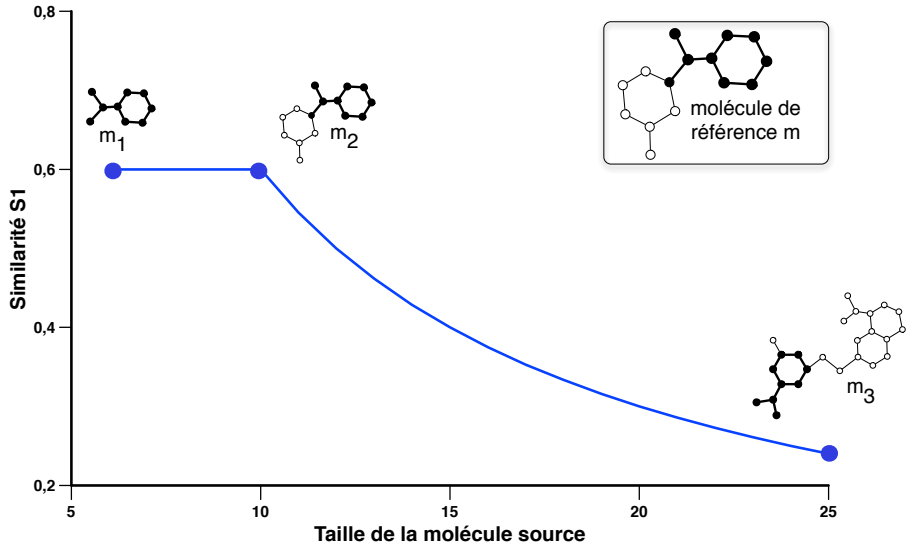


FIGURE 5.1 – Influence de la taille des molécules sur la similarité basée sur la SSCM, évolution de la similarité $S_1(m, m')$ en fonction de la taille de la molécule source m'. Dans cet exemple, on a $|SSCM| = 9$, $|m| = 15$, $|m_1| = 9$, $|m_2| = 15$ et $|m_3| = 25$. D'après l'équation 5.1, la similarité S_1 entre ces molécules est différente malgré le fait qu'elles partagent la même SSCM.

Sur cet exemple, on peut remarquer que :

- $S_1(m, m_1) = S_1(m, m_2)$ bien que m_1 et m_2 soient deux molécules (de taille) différentes ;
- $S_1(m, m_2) > S_1(m, m_3)$ bien que la SSCM entre ces deux paires de molécules soit identique.

On voit clairement que la similarité est toujours égale à 0,6 pour des molécules m' dont la taille est comprise entre 9 et 15 atomes. Dans ce cas, même si les molécules partagent la même SSCM, la mesure S_1 n'est pas en mesure de différencier des molécules pourtant différentes. Au-delà de 15 atomes, plus la taille de la molécule m' augmente, plus la valeur de similarité diminue.

Cela pose également un problème si l'on considère que la SSCM représente la sous-partie des molécules qui les rend similaires pour un expert comme, par exemple, un châssis moléculaire commun aux molécules d'une même famille chimique. Ainsi, même

si la SSCM identifiée correspond effectivement à un châssis moléculaire entre les molécules, leur ressemblance ne sera pas correctement prise en compte dans le calcul de leur similarité.

5.2 Similarité basée sur le degré d'inclusion

La sensibilité des mesures de similarité présentées ci-dessus est due au facteur de normalisation qui, selon les molécules, tient compte soit de la molécule m_1 soit de la molécule m_2. Pour diminuer cette sensibilité, nous proposons d'évaluer la taille de la SSCM relativement à chacune des molécules m et m', indépendamment l'une de l'autre. Cette définition est précisément celle du degré d'inclusion I_π d'un graphe moléculaire dans l'autre, défini dans le chapitre 3 :

$$I_\pi(m_1, m_2) = \frac{\left|SSCM(m_1, m_2)\right|}{\left|m_1\right|} \quad \text{et} \quad I_\pi(m_2, m_1) = \frac{\left|SSCM(m_2, m_1)\right|}{\left|m_2\right|}$$

Cette mesure étant asymétrique, nous définissons la fonction $\mathrm{Sim}_{I_\pi}(m_1, m_2)$ qui est une mesure de similarité symétrique entre les deux molécules m_1 et m_2, définie comme une fonction des deux degrés d'inclusion. Plusieurs définitions peuvent être utilisées comme le min, le max, la moyenne ou le produit (tableau 5.1).

Fonction	Définition
Moyenne	$\frac{1}{2}\Big(I_\pi(m_1, m_2) + I_\pi(m_2, m_1)\Big)$
Produit	$I_\pi(m_1, m_2).I_\pi(m_2, m_1)$
Min	$min\Big(I_\pi(m_1, m_2), I_\pi(m_2, m_1)\Big)$
Max	$max\Big(I_\pi(m_1, m_2), I_\pi(m_2, m_1)\Big)$

Tableau 5.1 – Quelques fonctions de similarité symétriques basées sur deux degrés d'inclusion asymétriques.

Ces quatre fonctions sont illustrées sur la figure 5.2 où la taille de la SSCM est égale à 9 et la taille de la molécule m est fixée à 20 : seule la taille de la molécule m' varie de 10 à 30 (axe des abscisses).

L'utilisation des fonctions *max* et *min* conduit au même problème que précédemment : pour une molécule m_1 dont la taille est comprise entre 9 et 15 (pour la fonction

min) ou supérieure à 15 (pour la fonction *max*), la valeur de la similarité est constante. On n'observe pas ce plateau avec le produit ou la moyenne des degrés d'inclusion. La valeur de similarité est moins sous-évaluée dans le cas de la moyenne des degrés d'inclusion que pour le produit : c'est donc cette fonction que nous avons retenue.

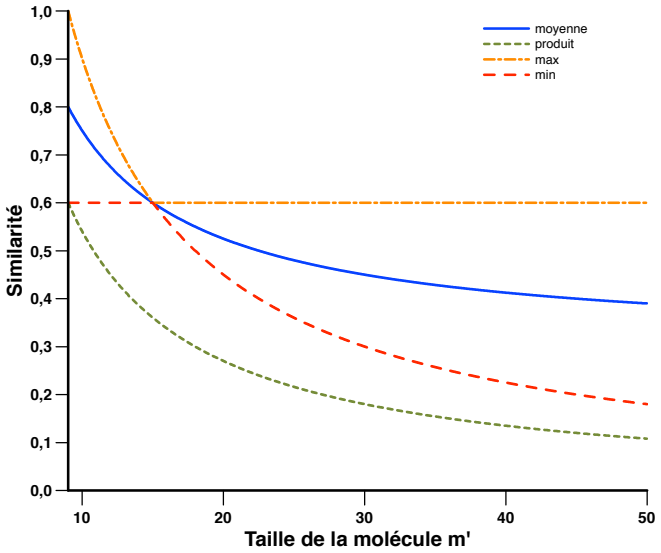


FIGURE 5.2 – Influence de la taille des molécules pour différentes fonctions combinant les degrés d'inclusion $I_\pi(m_1, m_2)$ et $I_\pi(m_2, m_1)$.

5.3 Distances basées sur les empreintes moléculaires

La structure 2D d'une molécule peut également se représenter sous forme d'un vecteur de bits (appelé *clé structurelle* ou *empreinte moléculaire*) dont chaque élément indique, par une valeur booléenne, la présence ou l'absence d'un fragment de la molécule (un sous-graphe ou un chemin), comme dans l'exemple de la figure 5.3. Ces vecteurs sont de taille finie, généralement comprise entre 512 à 2048 bits. Lorsque le nombre de fragments est supérieur à la taille du vecteur, une fonction de hachage est utilisée pour encoder les informations.

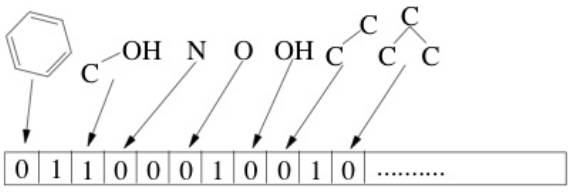


FIGURE 5.3 – Représentation de la structure 2D d'une molécule par une empreinte moléculaire.

Les mesures de similarité basées sur les empreintes moléculaires évaluent le recouvrement entre les vecteurs de bits et renvoient une valeur dans l'intervalle $[0,1]$ où la valeur 1 exprime le fait que des molécules ont les mêmes empreintes. Le *coefficient de Tanimoto* (aussi appelé coefficient de Jaccard) est un exemple classique de ce type de mesure et conforme au modèle de Tversky. Il est défini comme une fonction des fragments présents dans chacune des deux molécules (*i.e.* le nombre de bits à 1) et des fragments partagés par les deux molécules (*i.e.* le nombre de bits à 1 communs aux deux molécules). Etant donné les empreintes e_{m_1} et e_{m_2} de deux molécules m_1 et m_2, le coefficient de Tanimoto, noté C_T, s'écrit :

$$C_T(e_{m_1}, e_{m_2}) = \frac{c}{a+b+c} \tag{5.4}$$

où :

- a est le nombre de bits à 1 dans e_{m_1} et à 0 dans e_{m_2} ;
- b est le nombre de bits à 1 dans e_{m_2} et à 0 dans e_{m_1} ;
- c est le nombre de bits à 1 dans e_{m_1} et e_{m_2}.

Ce type de mesure permet d'analyser rapidement de grandes chimiothèques, c'est pourquoi il est largement utilisé dans la fouille de bases de données chimiques [Willett, 2006], [Eckert and Bajorath, 2007], la recherche de (sous-)structure et la recherche par similarité. Ces distances ont été évaluées dans plusieurs travaux, notamment [Willett *et al.*, 1986] qui montre que le coefficient de Tanimoto donne en moyenne les meilleures performances pour la classification structurale de molécules.

Cependant, bien qu'elles soient simples d'un point de vue calculatoire et efficaces, ces méthodes possèdent des propriétés intrinsèques qui biaisent les résultats des recherches par similarité [Flower, 1998], [Dixon and Koehler, 1999], [Xue *et al.*, 2003], [Yuan and Bajorath, 2008]. En effet, des molécules complexes (représentées par des graphes denses) ont tendance à générer des empreintes à plus forte densité de bits que des molécules plus simples, ce qui conduit souvent à des valeurs de similarités faussement élevées. De plus, de petits changements dans la structure moléculaire n'entraînent pas systématiquement de grandes modifications dans les empreintes alors que l'activité des molécules peut être différente.

5.4 Les fonctions noyau

Les fonctions noyau sont des fonctions qui permettent de transformer l'espace de représentation des objets en un espace de plus grande dimension [Schölkopf and Smola, 2002], [Scholkopf *et al.*, 2004]. Etant donné deux vecteurs A et B appartenant à l'espace $\mathcal{X}$ représentant deux objets et une application $\phi : \mathcal{X} \rightarrow \mathcal{H}$ qui permet de passer de l'espace de représentation des objets à l'espace des caractéristiques $\mathcal{H}$ muni du produit scalaire.

Une fonction noyau $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{R}$ est définie comme le produit scalaire, noté $\langle ., . \rangle$, entre les images des deux objets A et B dans l'espace des caractéristiques $\mathcal{H}$:

$$k(A,B) = \Big\langle \phi(A), \phi(B) \Big\rangle \tag{5.5}$$

La valeur retournée par le produit scalaire peut être vue comme la distance (non normalisée) entre les vecteurs A et B dans l'espace des caractéristiques. En effet, si les vecteurs sont orthogonaux, leur produit scalaire est nul ; au contraire, il est égal à 1 en cas d'égalité des deux vecteurs. Un des avantages des fonctions noyau est qu'il n'est pas nécessaire d'expliciter la transformation ϕ, le calcul du produit scalaire étant réalisé dans l'espace des entrées.

A partir de cette définition générale, le choix de l'application ϕ laisse toute latitude pour définir une large variété de mesures de similarité. Depuis ces dernières années, de nombreux travaux ont eu pour objectif de définir des fonctions noyau sur des données structurées comme les graphes [Gärtner, 2003b], [Borgwardt, 2007], [Kozak *et al.*, 2007].

Parmi celles-ci, on distingue deux familles de fonctions : les *noyaux de convolution* qui décomposent les graphes en sous-parties et définissent un noyau à partir de noyaux entre ces sous-parties et les *noyaux de diffusion* qui prennent en compte la structure globale des graphes.

5.4.1 Noyaux de convolution sur des graphes

A partir du travail séminal de [Haussler, 1999] qui a défini les noyaux de convolution, plusieurs chercheurs ont proposé ces noyaux sur divers types de données structurées telles que les chaînes, les arbres et les graphes [Gärtner *et al.*, 2003].

L'idée générale de cette approche est la suivante. Chaque objet structuré A et B, représenté par un graphe, est décomposé en un ensemble $\mathcal{P}$ de sous-parties. La forme générale d'un noyau de convolution entre ces objets, noté $k(A,B)$, est définie comme la somme du noyau k_s entre chaque paire de sous-parties (p_i, p_j) :

$$k(A,B) = \sum_{\substack{p_i \in \mathcal{P}(A) \\ p_j \in \mathcal{P}(B)}} k_s(p_i, p_j) \tag{5.6}$$

Une forme basique du noyau k_s qui évalue la ressemblance entre les parties p_i et p_j est le *noyau de Dirac*, défini comme suit :

$$k_s(p_i, p_j) = \begin{cases} 1 & \text{si } p_i = p_j \\ 0 & \text{sinon} \end{cases} \tag{5.7}$$

Les noyaux de convolution se différencient par la manière de décomposer une structure en sous-structures. Dans le cas de graphes, on utilise souvent des parcours dans les graphes (séquences, éventuellement infinies, de sommets obtenus en parcourant les arêtes dans le graphe) :

- les parcours déterministes : noyaux basés sur les puissances de matrices d'adjacence [Gärtner, 2003a] ;
- les parcours aléatoires : le noyau marginalisé [Kashima *et al.*, 2003], [Mahé *et al.*, 2005] ;
- les parcours en profondeur d'abord comme le noyau de Tanimoto [Ralaivola *et al.*, 2005].

5.4.1.1 Le noyau de Tanimoto

Le noyau de Tanimoto [Ralaivola *et al.*, 2005] est une adaptation sous forme de fonction noyau de la distance de Tanimoto (section 5.3). Les molécules sont représentées sous forme de vecteurs composés de chemins extraits dans les graphes moléculaires. Etant donné deux graphes G et G', le noyau $k_d(G,G')$ compte le nombre de chemins communs de longueur d entre les graphes G et G'. Le noyau de Tanimoto, noté $k_d^t(G,G')$, est alors défini par :

$$k_d^t(G,G') = \frac{k_d(G,G')}{k_d(G,G) + k_d(G',G') - k_d(G,G')} \tag{5.8}$$

où $k_d(G,G)$ (resp. $k_d(G',G')$) est le nombre de chemins de longueur d dans le graphe G (resp. le graphe G').

Cette fonction noyau a été récemment utilisée dans le domaine de la chémogénomique [Jacob and Vert, 2008] et dans les études de relation structure-activité [Ning *et al.*, 2008].

5.4.1.2 Le noyau spectral

Le noyau spectral [Leslie *et al.*, 2002] compare le nombre d'occurrences de chemins communs entre deux graphes. Pour une paire de graphes m_1 et m_2, sa forme générale est la suivante :

$$k(m_1,m_2) = \sum_{c \in \mathcal{P}} \phi_c(m_1).\phi_c(m_2) \tag{5.9}$$

où c représente un chemin commun à m_1 et m_2 et $\phi_{m_1}(c)$ (resp. $\phi_{m_2}(c)$) correspond au nombre d'occurrences du fragment c dans le graphe m_1 (resp. dans le graphe m_2). L'ensemble $\mathcal{P}$, de taille finie, contient des fragments moléculaires extraits des graphes m_1 et

m_2. On peut, par exemple, construire l'ensemble des fragments possibles de longueur n ou encore l'ensemble des fragments possibles de longueur comprise entre 0 et n.

5.4.1.3 Le noyau de Décomposition Pondérée (DP)

Dans la fonction noyau proposée par [Menchetti *et al.*, 2005], chaque molécule m_1 est décrite par un ensemble de sélecteurs (des sous-graphes s du graphe moléculaire m_1) associés à leur environnement respectif (appelé le contexte de l'occurrence de s dans le graphe m_1). La similarité entre deux graphes m_1 et m_2 est alors basée sur la ressemblance entre ces sélecteurs mais aussi par la ressemblance entre leur contexte. Ce noyau a été utilisé pour la discrimination de molécules dans [Ceroni *et al.*, 2007].

Etant donné un graphe m_1, le contexte z d'un sélecteur s est le sous-graphe $G(s,l)$ de m_1 induit par l'ensemble des sommets qui sont atteignables à partir de s par un chemin de longueur au plus égal à l et par l'ensemble des arêtes qui ont au moins un sommet dans l'ensemble des sommets de $G(s,l)$. La forme générale du noyau DP, noté $k_{dp}(m_1,m_2)$, entre deux graphes moléculaires m_1 et m_2 s'écrit :

$$k_{dp}(m_1,m_2) = \sum_{\substack{(s,z)\in\mathcal{P} \\ (s',z')\in\mathcal{P}'}} \delta(s,s').\kappa(z,z') \tag{5.10}$$

où le noyau δ est le noyau de Dirac qui évalue la ressemblance entre les sélecteurs s et s' et le noyau κ qui évalue la ressemblance entre les contextes z et z' des sélecteurs s et s'. La figure 5.4 illustre ce calcul.

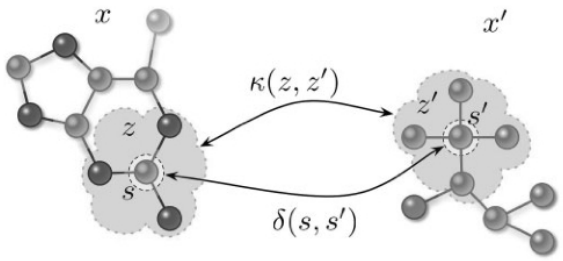


FIGURE 5.4 – Comparaison de sous-structures dans le noyau DP (exemple extrait de [Menchetti *et al.*, 2005]). Les sélecteurs sont ici de simples sommets et leur contexte est le sous-graphe induit par les voisins atteignables par une longueur égale à 1.

Lorsque les sélecteurs sont de simples sommets, le noyau δ peut être calculé en temps constant (exemple de la figure 5.4). Pour les contextes dont la taille est plus importante, les auteurs font l'hypothèse que les graphes sont étiquetés et construisent, pour chaque sélecteur, un index des étiquettes du contexte triées par ordre lexicographique. Ils ramènent ainsi la comparaison des contextes à la comparaison de deux index

ordonnés au lieu de comparer des sous-graphes, ce qui rend le calcul du noyau κ plus efficace. Cependant, la quantité d'information topologique capturée reste limitée puisque les comparaisons entre les contextes sont réalisées sur la base de listes ordonnées d'étiquettes.

5.4.2 Noyaux à diffusion

Les noyaux à diffusion forment une classe de fonctions noyau dans laquelle les structures ne sont pas décomposées en sous-parties mais sont comparées directement. L'approche consiste à capturer une information globale sur la structure de l'instance dans une fonction noyau simplement basée sur la description du voisinage des entités composant l'objet. Par exemple, [Kondor and Lafferty, 2002] construit le noyau entre deux graphes comme une fonction du noyau k_s entre chaque paire de sommets dans les graphes :

$$k(m_1, m_2) = f_{i,j}\Big(k_s(m_{1i}, m_{2j})\Big) \tag{5.11}$$

où k_s est un noyau qui évalue la similarité entre les sommets m_{1i} et m_{2j} appartenant respectivement aux objets m_1 et m_2.

5.4.2.1 Le noyau d'Appariement Optimal

Le noyau AO, proposé par [Fröhlich *et al.*, 2005b], calcule la similarité entre deux graphes m_1 et m_2 en tenant compte de la similarité entre chaque paire de sommets dans les graphes ainsi que la similarité entre leurs voisins. Le calcul se déroule en deux étapes conceptuellement proches de la méthode I_π.

La première étape consiste à évaluer la matrice K, de taille $|m_1|.|m_2|$, dont chaque élément $K_{(a_i,a'_m)}$ correspond à la similarité entre les atomes a_i et a'_m, évaluée par la fonction noyau $k_s(a_i, a'_m)$:

$$K = \Big(k_s(a_i, a'_m)\Big)_{i,j} \tag{5.12}$$

Le calcul nécessite de définir les fonctions de base suivantes. Le noyau $k_{atome}(a_i, a'_m)$ exprime la similarité entre les atomes a_i et a'_m du point de vue de leurs attributs. Le noyau $k_{liaison}(a_i \to v_j, a'_m \to v'_n)$ quantifie la similarité entre les deux liaisons $a_i \to v_j$ et $a'_m \to v'_n$ où v_j et v'_n sont des atomes voisins respectivement de a_i et a'_m. Ces fonctions peuvent être définies, par exemple, par la fonction de Dirac.

La fonction R_0 évalue la ressemblance entre les ensembles d'atomes voisins V et V' des atomes a_i et a'_m. Cette valeur correspond au meilleur appariement π entre les éléments de V et V', c'est-à-dire celui qui maximise la somme des $k_{atome}(a_i, a'_m).k_{liaison}(a_i \to v_j, a'_m \to v'_n)$. Cette somme est ensuite normalisée par la taille du plus grand voisinage :

$$R_0(a_i, a'_m) = \frac{1}{max(|V|,|V'|)} \max_{\pi} \sum_{j=1}^{min(|V|,|V'|)} \left(k_{atome}\left(v_j, \pi(v_j)\right).k_{liaison}\left(a_i \rightarrow v_j, a'_m \rightarrow \pi(v_j)\right)\right) \quad (5.13)$$

La recherche du meilleur appariement est réalisée par l'algorithme hongrois. L'auteur réduit la complexité de cet appariement en pré-calculant toutes les combinaisons d'appariement possibles entre les éléments de V et V' ; la recherche de l'appariement peut alors être réalisée en $O(1)$.

L'auteur tient compte également de la ressemblance entre les voisins indirects, c'est-à-dire ceux situés à une distance l des atomes a_i et a'_m. Pour cela, il définit la fonction R_l qui calcule la moyenne de tous les R_0 évalués pour chaque paire d'atomes voisins situés à une distance topologique l des atomes a_i et a'_m.

$$R_l(a_i, a'_m) = \frac{1}{|V|.|V'|} \sum_{\substack{v_j \in V \\ v'_n \in V'}} R_{l-1}(v_j, v'_n) \quad (5.14)$$

Exemple
Considérons le calcul de la fonction R_l pour L=2 entre les atomes a_3 et a'_5 des molécules de la figure 5.5.

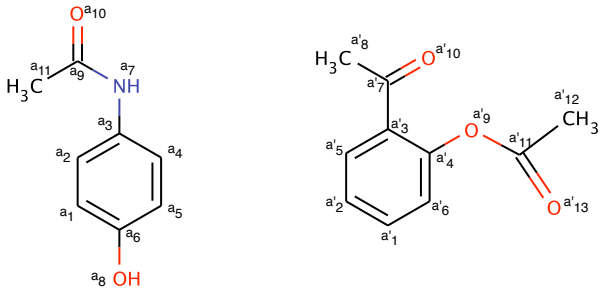


FIGURE 5.5 – Voisins directs et indirects des atomes a_3 et a'_5 (Exemple extrait de [Fröhlich *et al.*, 2005b]

La première étape consiste à évaluer la moyenne des 6 combinaisons possibles de R_1 entre les voisins directs : $\{a_2, a_4, a_7\}$ pour l'atome a_3 et $\{a'_2, a'_3\}$ pour l'atome a'_5 dans la molécule de droite. Ensuite, pour chaque R_1, on calcule la moyenne des 9 combinaisons possibles entre les voisins des atomes a_3 et a'_5 situés à une distance l=2 : $\{a_1, a_5, a_9\}$ et $\{a'_1, a'_4, a'_7\}$. On a donc :

$$
\begin{aligned}
R_2(a_3, a'_5) &= \frac{1}{3.2} \sum_{\substack{v \in \{a_2, a_4, a_7\} \\ v' \in \{a'_2, a'_3\}}} R_1(v, v') \\
\text{avec } R_1(v, v') &= \frac{1}{3.3} \left(\sum_{\substack{w \in \{a_1, a_5, a_9\} \\ w' \in \{a'_1, a'_4, a'_7\}}} R_0(w, w') \right)
\end{aligned}
$$

Finalement, le noyau k_s entre deux atomes a_i et a'_m est défini par :

$$k_s(a_i, a'_m) = k_{atome}(a_i, a'_m) + R_0(a_i, a'_m) + \sum_{l=1}^{L} \gamma(l) R_l(a_i, a'_m) \tag{5.15}$$

où le facteur $\gamma(l)$ réduit l'influence des voisins les plus éloignés. Etant donné la taille L maximale des environnements pris en compte par l'algorithme, le facteur est défini par :

$$\gamma(l) = \left(1 - \frac{l+1}{L}\right)^2 \tag{5.16}$$

Il est à noter que le deuxième terme de l'équation 5.15 qui donne la ressemblance entre les voisins immédiats n'est pas impacté par ce coefficient. Pour chaque paire d'atomes dans les molécules, les deux premiers termes de la somme peuvent être calculés en $O(1)$. En effet, le nombre de voisins des atomes est borné par une petite valeur, il est donc possible de pré-calculer la valeur de tous les appariements possibles entre les ensembles de voisins directs. De même, comme on n'utilise ici que les valeurs locales statiques, il est possible de pré-calculer l'ensemble des combinaisons pour le calcul des R_l, ce qui réduit fortement la complexité. Soit $n = max(|m_1|, |m_2|)$, la complexité du calcul de la matrice K est en $O(n^2)$.

Cette approche s'apparente à la méthode I_π dans le sens où les deux graphes moléculaires sont comparés directement et la similarité entre chaque paire tient compte aussi des voisins à une distance topologique l. Cependant, la grande différence entre les deux approches est qu'ici, le calcul de la similarité des voisins (R_l) est statique et non pas dynamique : les valeurs prises en compte dans le calcul ne sont pas mises à jour contrairement au calcul de la matrice INC. C'est entre autre pour cette raison que la complexité du noyau AO est très inférieure à celle de l'algorithme I_π.

Lorsque la matrice K est calculée, la seconde étape consiste à trouver le meilleur appariement entre les atomes a_i de la molécule m_1 avec les atomes a'_m de la molécule

m_2. En considérant ce problème comme un graphe bipartite où les poids associés aux appariements sont donnés par la matrice K, on cherche alors à maximiser la somme des $k_{entite}(m_{1i}, m_{2j})$. Pour ce faire, les auteurs utilisent l'algorithme hongrois [Kuhn, 1955] dont la complexité est en $O(max(|m_1|, |m_2|)^3)$.

5.4.2.2 Le noyau Itérative Similarity Optimal Assignment

[Rupp *et al.*, 2007] propose de redéfinir le noyau k_s (équation 5.15) du noyau AO ; l'idée est similaire à [Bisson, 1995], [Aci *et al.*, 2007] où les similarités locales sont propagées à travers la structure des molécules. Le calcul de ce noyau est défini de manière récursive où la similarité de chaque atome prend en compte celle de ses voisins. A l'itération n, le noyau $k_s^{(n)}(a_i, a'_m)$ est défini de la façon suivante :

$$k_s^{(n)}(a_i, a'_m) = (1-\alpha)k_{atome}(a_i, a'_m) + \alpha.\left(\frac{1}{\max(|V|, |V'|)}\max_{\pi} \sum_{\substack{v_j \in V \\ \pi(v_j) \in V'}} Sim_{voisins}\Big(v_j, \pi(v_j), a_i, a'_m\Big)\right) \quad (5.17)$$

avec :

$$Sim_{voisins}(v_j, v'_n, a_i, a'_m) = k_s^{(n-1)}(v_j, v'_n).k_{liaison}\Big((a_i, v_j), (a'_m, v'_n)\Big) \quad (5.18)$$

où π est la relation d'appariement entre un atome de m_1 et un atome de m_2 et les ensembles V et V' correspondent aux ensembles de voisins des atomes a_i et a'_m respectivement. Le coefficient α prend ses valeurs dans $]0, 1[$ et permet de donner plus d'importance au terme constant ou récursif. Le premier terme de la somme est une constante et correspond à la ressemblance entre les atomes a_i et a'_m. Le deuxième terme, qui est récursif, évalue la similarité entre les voisinages des atomes a_i et a'_m en recherchant le meilleur appariement (basée sur l'algorithme hongrois). A chaque itération et pour chaque paire d'atomes a_i et a'_m, un appariement optimal est recherché en se basant sur les similarités de l'itération précédente. Le calcul s'arrête lorsque le système s'est stabilisé (déterminé par une valeur seuil).

Dans le cas de graphes moléculaires, le degré des sommets étant borné par une petite constante, chaque appariement peut être déterminé en temps constant en pré-calculant, pour chaque itération, l'ensemble des appariements possibles dans le terme récursif, tout comme pour la méthode I_π. Comme les fonctions k_{atome} et $k_{liaison}$ peuvent également être calculées en temps constant, le calcul de la matrice est alors en $O(N.|V|.|V'|)$ où N est le nombre d'itérations.

Finalement, cette fonction noyau est un cas particulier de l'algorithme I_π dans lequel :
- la fonction f qui évalue la ressemblance entre deux voisins (équation 3.2) est définie par le produit de la similarité des voisins et de la similarité des liaisons (dans I_π, nous avons utilisé la moyenne) ;
- la procédure d'appariement global entre les atomes est réalisée ici par l'algorithme hongrois alors que dans I_π, cette opération est réalisée par un parcours de graphes.

5.5 Autres approches

D'autres mesures de similarité moléculaire se basent sur la ressemblance locale entre les atomes. Un des premiers travaux dans ce sens est celui de [Morgan, 1965] qui propose une façon canonique d'étiqueter les atomes d'une molécule. La méthode consiste à étiqueter chaque atome en fonction de l'étiquette de ses voisins. Le processus est itératif : à l'initialisation, chaque atome a l'étiquette 1 et à chaque itération, l'étiquette d'un atome est la somme des étiquettes de ses voisins.

Plus généralement, l'évaluation de la similarité en tenant compte du voisinage des atomes est fréquemment utilisée dans le domaine de la chemo-informatique. On peut citer par exemple l'outil MOLDRINT2D [Bender *et al.*, 2004] qui se base sur une décomposition des graphes moléculaires en sous-graphes canoniques [Faulon *et al.*, 2004]. Ces sous-graphes permettent de construire des vecteurs, appelés *signatures moléculaires*, dont les composants correspondent au nombre d'occurrences d'un graphe canonique autour d'un atome particulier. La comparaison de ces signatures définit alors une mesure de similarité moléculaire. Récemment, [Faulon *et al.*, 2008] a proposé une fonction noyau à partir de ces signatures.

Chapitre 6

Classification de molécules

L'algorithme proposé dans les chapitres précédents identifie un ensemble de Sous-Structure Commune (SSC) entre deux graphes A et B et mesure le degré d'inclusion du graphe A dans le graphe B. Un certain nombre de paramètres et de stratégies sont proposées à différents niveaux de l'algorithme : pour les appariements locaux entre sommets, le critère d'arrêt de la recherche des Arbre Partiellement Couvrant (APC) ou encore la manière de sélectionner les graines d'appariement. Les combinaisons sont nombreuses et nous voulons tester, sur un problème réel, le comportement de ces variantes de façon à identifier les stratégies qui semblent les plus pertinentes. Dans ce chapitre, nous explorons donc les différentes possibilités sur un problème de classification de molécules.

L'application des méthodes de catégorisation à des composés chimiques a démarré dans les années 1980 avec l'augmentation de la taille des chimiothèques et la nécessité, pour les utilisateurs, de disposer d'outils d'analyse automatique pour sélectionner des composés dans les chimiothèques ou analyser la diversité des chimiothèques. Les premiers travaux sur la classification de structures chimiques sont dus à [Adamson, 1973], [Adamson and Bush, 1975] et [Willett, 1987]. Plusieurs articles de revue ont suivi : [Bratchell, 1989], [Barnard and Downs, 1992], [Downs and Barnard, 2002]. On observe ces dernières années un regain d'intérêt pour la classification de molécules [Reynolds *et al.*, 1998], [Nicolaou *et al.*, 2002], [Holliday *et al.*, 2004], [Ott *et al.*, 2004], [Stahl and Mauser, 2005].

L'objectif de ce chapitre est d'appliquer l'algorithme I_π à cette problématique pour évaluer sa capacité à évaluer la similarité entre les composés et de la comparer à d'autres mesures de similarité entre composés. Cette étude est d'autant plus intéressante qu'il y a peu de travaux sur la classification de molécules et que les études comparatives dans ce domaine sont rares. Nous pouvons citer [Adamson and Bawden, 1981] qui compare plusieurs algorithmes de classification hiérarchique en utilisant la distance euclidienne et [Raymond *et al.*, 2003] qui compare les empreintes moléculaires avec les graphes

moléculaires en utilisant plusieurs algorithmes de classification. Plus récemment, [Haranczyk and Holliday, 2008] a comparé plusieurs coefficients de similarité basés sur des empreintes moléculaires pour classer des molécules avec un algorithme hiérarchique et un algorithme non hiérarchique. Cependant, dans le cas de la classification automatique, il n'y a, à notre connaissance, pas d'études comparatives entre les fonctions noyau de graphes.

La section 6.1 décrit le cadre expérimental mis en place pour réaliser les expériences en catégorisation de molécules. Dans la section 6.2, nous testons les différents paramètres de l'algorithme : la stratégie d'appariement local, le critère d'arrêt ainsi que le choix des graines. Ensuite, dans la section 6.3, notre méthode est comparée à plusieurs mesures de similarité sur le problème de la classification de composés. Enfin, dans la section 6.4, nous présentons une étude plus approfondie qui vise à comparer l'algorithme I_π avec le noyau Appariement Optimal (AO) et le noyau Itérative Similarity Optimal Assignment (ISOA).

6.1 Démarche expérimentale

Le processus de classification automatique (ou catégorisation) mis en place dans ce travail se déroule en plusieurs étapes, illustrées sur la figure 6.1.

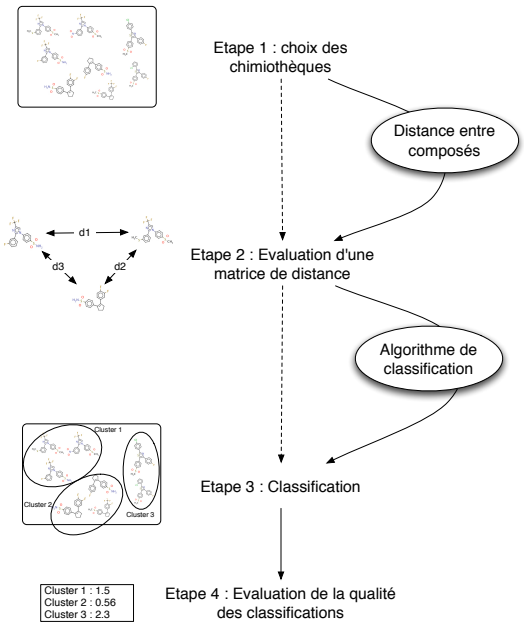


FIGURE 6.1 – Processus mis en place pour classer des molécules.

A partir d'un ensemble de molécules (appelé chimiothèque), la distance entre chaque paire de composés est évaluée puis stockée dans une matrice. Un algorithme de classification est ensuite utilisé pour regrouper les molécules en classes de façon que chaque classe contienne un ensemble de molécules les plus similaires entre elles. Nous détaillons dans cette section, les choix effectués pour chaque étape de l'expérimentation.

6.1.1 Chimiothèques utilisées

Les *chimiothèques* sont des collections de molécules provenant de sources multiples comme des industries pharmaceutiques ou des laboratoires publics et qui sont notamment utilisées dans le criblage à haut débit. Une chimiothèque peut être caractérisée par sa diversité, c'est-à-dire par le fait qu'elle contienne des molécules très différentes les unes des autres (utiles dans le cas où l'on souhaite identifier un composé sans disposer pour autant de beaucoup d'information sur sa structure) ou au contraire, des molécules très similaires (utilisées lorsque l'on a déjà une idée de la structure d'une molécule bio-active et que l'on désire tester des analogues structuraux) [Dubois *et al.*, 2008].

Dans les chimiothèques, les molécules sont représentées par leur structure, un certain nombre de propriétés physico-chimiques et parfois des informations sur leur activité biologique. On peut également trouver des informations sur les familles chimiques auxquelles les molécules appartiennent ; ce sont ces chimiothèques qui vont nous intéresser pour réaliser les tests de classification.

En effet, comme le souligne [Candellier *et al.*, 2006], l'évaluation des résultats d'une classification est délicate en l'absence de critère de validation. Ce problème vient du fait qu'une classification ne produit pas un résultat "juste" ou "faux" mais un résultat significatif ou non. On peut définir des critères d'évaluation "internes" pour piloter la classification (*e.g.* inertie) et/ou "externes" pour la valider. Dans le cas ou l'on mesure l'adéquation entre les classifications apprises et données par l'expert, on dispose d'une mesure "objective", au moins vis-à-vis du domaine d'application. C'est pourquoi les chimiothèques dont les molécules sont regroupées en familles structurales sont intéressantes pour nos expérimentations : les familles servant de classification de référence.

Parmi les collections publiques répondant à ces critères, notre choix s'est arrêté sur un ensemble de quatre chimiothèques, de diversité différente, rassemblées par [Sutherland *et al.*, 2003]. Les quatre bases sont listées ci-dessous, par ordre de diversité croissante :

- la base COX2 contient 467 inhibiteurs de la cyclooxygénase-2 et a été assemblée à partir de tests d'activité *in vitro*. Elle est composée de 13 familles ;
- la base BZR contient 405 ligands du récepteur de la benzodiazépine répartis en 14 familles ;

– la base DHFR contient 756 inhibiteurs de la dihydrofolate réductase, répartis en 18 familles dont l'une d'elles contient des molécules très diverses sans châssis moléculaire commun. Pour ne conserver que des familles structurales cohérentes (représentées par un squelette unique), nous ne tenons pas compte de cette famille. La base que nous utilisons contient donc 724 composés répartis en 17 familles ;
– la base ER contient 1009 de récepteurs de l'œstrogène, assemblés à partir de sources diverses. Une première compilation de 616 composés a été préparée par le National Toxicology Program au National Institute of Environmental Health Sciences. Un ensemble de 393 composés, extraits de la littérature pour leur intérêt pharmaceutique, ont ensuite été ajoutés. Pour notre étude, nous ne considérons que ces 393 composés, répartis en 3 familles structurales, car les 616 premiers composés forment plutôt des singletons que des familles.

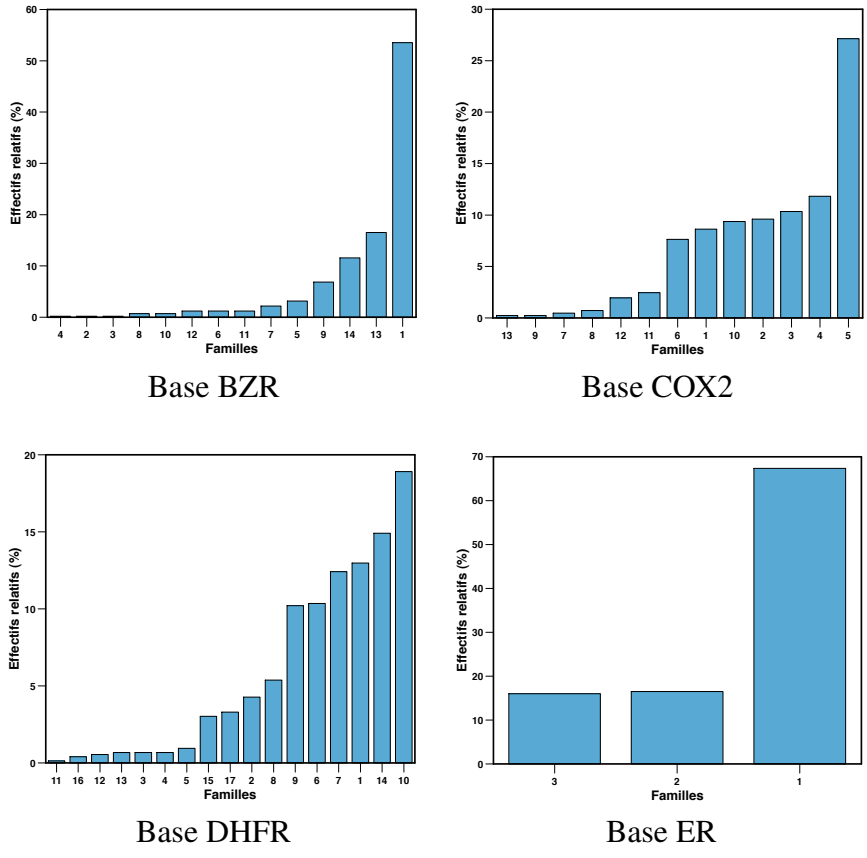


FIGURE 6.2 – Répartition des effectifs dans les familles chimiques. Bases BZR et COX2 : une grande famille regroupe respectivement plus de 53% pour BZR et 27% des molécules, le reste des molécules étant réparties dans les autres familles de manière peu homogène. Base DHFR : 6 familles sur 17 contiennent entre 10 et 20% des composés. Base ER : une famille contient plus de 67% des molécules, le reste étant réparti de manière équivalente sur les deux autres familles de la base.

Bien qu'il existe de nombreuses autres chimiothèques publiques largement utilisées et validées dans la littérature, ces 4 bases présentent l'avantage d'avoir été divisées en familles chimiques bien définies et d'être décrites de façon très précise par des experts. Elles sont par ailleurs assez hétérogènes les unes par rapport aux autres, tant d'un point de vue diversité chimique que du point de vue de la répartition des effectifs au sein des familles (figure 6.2) ce qui placent les algorithmes de classification et les mesures de distance dans des conditions d'apprentissage différentes. De ce fait, malgré le faible nombre de bases, ce jeu de données est un bon échantillon pour notre étude comparative.

Dans ces chimiothèques, les structures moléculaires sont décrites au format *sdf* [MDL, 2009]. Les descripteurs utilisés ont été réduits au minimum pour que les mesures de similarité prennent en compte des informations identiques sur la structure des molécules. On considère donc que les molécules sont représentées par un graphe moléculaire 2D où les atomes sont décrits par leur type et les liaisons par leur valence.

6.1.2 Choix des mesures de similarité

Le but de cette étude est de tester les différentes possibilités de la méthodeI_π puis de comparer les mesures de similarité décrites dans le chapitre 5. Les mesures de similarité utilisées dans cette étude sont les suivantes : Sim_{I_π}, une mesure basée sur les empreintes moléculaires, le noyau AO, le noyau ISOA, le noyau DP, le noyau spectral et le noyau de Tanimoto.

Sim_{I_π}
La mesure de similarité Sim_{I_π} est basée sur la quantification du degré d'inclusion d'un graphe dans un autre et est définie comme la moyenne des deux degrés d'inclusion (tableau 5.1, section 5.2). L'algorithme a été implémenté en C++ et contient des modules qui correspondent aux différentes stratégies d'appariement possibles (voir section 3.3).

Comme l'appariement global entre les atomes est de type $1-1$ et que l'on cherche en priorité des Sous-Structure Commune (SSC) connexes, les algorithmes hongrois, glouton et Algorithme de Minimisation des Conflits (AMC) ne peuvent pas être utilisés pour la phase d'appariement global. Pour cette étape, on utilisera donc la recherche d'APC pour laquelle plusieurs variantes (identifiées dans le tableau 6.1) ont été testées en faisant varier la stratégie d'appariement local des voisins et l'arrêt de l'algorithme [11].

11. Pour rappel, dans les variantes "stochastiques", l'appariement (local) des atomes voisins est réalisé de manière aléatoire sans utiliser la matrice INC. Dans les variantes gloutonne et hongroise respectivement, l'appariement est réalisé par un algorithme glouton et l'algorithme hongrois [Kuhn, 1955]. Les variantes "Avec arrêt" s'arrêtent dès lors qu'un atome ne peut pas être apparié tandis que dans les variantes "SansArrêt", l'exploration des graphes se poursuit jusqu'à ce qu'il n'y ait plus aucun atome à apparier.

Appariement local des voisins	Stratégie d'arrêt de l'algorithme	
	Avec arrêt sur erreur	Sans arrêt sur erreur
glouton	APC_Glouton_AvecArret	APC_Glouton_SansArret
hongrois	APC_Kuhn_AvecArret	APC_Kuhn_SansArret
stochastique	APC_Stochast_AvecArret	APC_Stochast_AvecArret

Tableau 6.1 – Variantes de l'algorithme I_π, décrites dans le chapitre 3.

Pour éviter la multiplication des paramètres et limiter la complexité de l'analyse des résultats, toutes ces variantes utilisent la totalité des graines d'appariement disponibles. Des tests supplémentaires pour évaluer les stratégies de sélection des graines ont été menés et sont discutés dans la section 6.2.4. En ce qui concerne le calcul de la matrice INC, le nombre d'itérations max_{Iter} a été déterminé expérimentalement. Sur un large échantillon de composés, nous avons observé les matrices INC$^(k)$ pour k allant de 1 à 20 itérations. Pour des valeurs de k supérieures à 5, les matrices restent égales à INC^5 ; c'est donc cette valeur que nous avons considéré pour arrêter le calcul de la matrice INC.

Mesure basée sur les empreintes moléculaires
Pour la similarité basée sur des empreintes moléculaires, nous avons choisi un outil intégré dans un logiciel de la société Chemaxon [12], spécialisée dans l'édition de logiciels pour la chémo-informatique. Ces outils sont écrits en Java.

Le module JChem [http ://www.chemaxon.com/product/jc_base.html,] propose un module de génération des signatures moléculaires qui détecte tous les chemins de longueur au plus égale à n ainsi que tous les cycles. Les paramètres sont la longueur des vecteurs de bits et la taille maximale des motifs considérés. Ils doivent être ajustés pour chaque base de façon à trouver le meilleur compromis entre collecter le maximum d'information dans le vecteur de bits et limiter les phénomènes de collision dans les vecteurs. L'éditeur de l'outil préconise les caractéristiques suivantes :

- le pourcentage maximal de bits à 1 dans le vecteur ne doit pas excéder 80%. Dans le cas contraire, le contenu informatif d'une signature est diminué et alors, dans la recherche de sous-structure par exemple, des molécules similaires mais pas identiques ne pourront pas être distinguées grâce à leur signature ;
- le pourcentage moyen de bits à 1 dépend beaucoup de l'application et d'une base particulière (*e.g.* la diversité totale d'une chimiothèque influence grandement la quantité de 1 dans la signature). Le contenu est optimal à un pourcentage moyen de 1 de 50% bien qu'en général, la quantité de 1 ne devrait pas excéder 40% pour éviter les collisions fréquentes.

12. http ://www.chemaxon.com

Comme la matrice de similarité entre les composés d'une chimiothèque ne peut pas être exportée à partir de cet outil, nous avons donc dû utiliser l'algorithme de classification intégré dans la suite logicielle (module JKlustor [JKlustor,]) pour classer les molécules. Cet algorithme est de type hiérarchique et utilise l'indice d'agrégation de Ward. C'est la seule exception au protocole expérimental. Dans la suite de ce document, cette mesure de similarité est appelée *chemaxonWard* .

Le noyau AO (section 5.4.2.1) [Fröhlich *et al.*, 2005b]
Les paramètres de ce noyau sont les noyaux k_{atome} et $k_{liaison}$ définis ici par le noyau de Dirac. Le paramètre L qui définit la taille maximale du voisinage des atomes a été fixé à 5 pour être en accord avec la valeur de max_{Iter} dans l'algorithme I_π. Nous avons utilisé l'implémentation en langage Java fournie par l'auteur.

Le noyau ISOA (section 5.4.2.2) [Rupp *et al.*, 2007]
L'implémentation de l'auteur (en langage Java) a également été utilisée dans cette expérience. Comme pour le noyau AO, les noyaux k_{atome} et $k_{liaison}$ sont les noyaux de Dirac.

Le noyau DP (section 5.4.1.3) [Menchetti *et al.*, 2005]
Cet algorithme est implémenté en C++ par l'auteur. Le paramètre à ajuster est la taille l des contextes des sélecteurs, c'est-à-dire la longueur maximale des chemins dans le sous-graphe correspondant au contexte d'un sélecteur. Pour les expérimentations, la valeur par défaut ($l=3$) a été utilisée.

Le noyau spectral (section 5.4.1.2) [Leslie *et al.*, 2002] et le **noyau de Tanimoto** (section 5.4.1.1) [Ralaivola *et al.*, 2005]
Pour ces deux algorithmes, nous avons utilisé l'implémentation en C++ disponible dans le logiciel ChemCPP [Mahé *et al.*, 2006]. Pour chacun des algorithmes, deux stratégies (utilisant respectivement les paramètres l et u) permettent de construire l'ensemble de chemins extraits des graphes moléculaires : l'ensemble des chemins de longueur égale à l ou l'ensemble des chemins de longueur comprise entre 1 et u.

6.1.3 Choix de l'algorithme de classification

De nombreuses méthodes de classification sont disponibles et certaines ont déjà été appliquées à la classification de composés chimiques [Downs and Barnard, 2002]. Un important travail d'évaluation des algorithmes de classification a été réalisé dans les années 1980 par [Rubin and Willett, 1983] et [Willett, 1984] et [Willett, 1987]. Willett a comparé trente méthodes hiérarchiques et non hiérarchiques en se basant sur les empreintes moléculaires 2D. Les résultats ont montré que les approches hiérarchiques, comme la Classification Ascendante Hiérarchique (CAH), donnaient de loin les meilleures performances, résultat confirmé plus tard par [Downs *et al.*, 1994].

Compte tenu des bonnes performances des méthodes hiérarchiques relatées dans la littérature, nous avons choisi d'utiliser la CAH qui construit une hiérarchie des composés ; un exemple de ce type de classification, appelé un dendrogramme, est illustré sur la figure 6.3. Cet algorithme présente l'avantage d'avoir peu de paramètres explicites. En effet, dans un cas réel (où l'on ne connait pas les classifications initiales), l'utilisateur est pénalisé s'il y a trop de paramètres mais surtout si les paramètres ne peuvent pas être ajustés automatiquement à partir des caractéristiques des données.

En pratique, pour chaque tuple <chimiothèque, mesure de similarité>, un dendrogramme est construit à l'aide de la fonction "hclust" implémentée dans le logiciel R [13]. La fonction d'agglomération utilisée est l'indice de Ward qui correspond à l'ordonnée sur la représentation du dendrogramme. Cependant, nous considérerons ici que la hauteur *h* du dendrogramme (l'échelle de l'ordonnée) représente le nombre de classes dans une classification. Il est possible d'accéder à une classification particulière en utilisant la fonction "cutree" (également disponible dans le logiciel R) qui prend en argument le nombre de classes désiré et coupe l'arbre à la hauteur *h* correspondante.

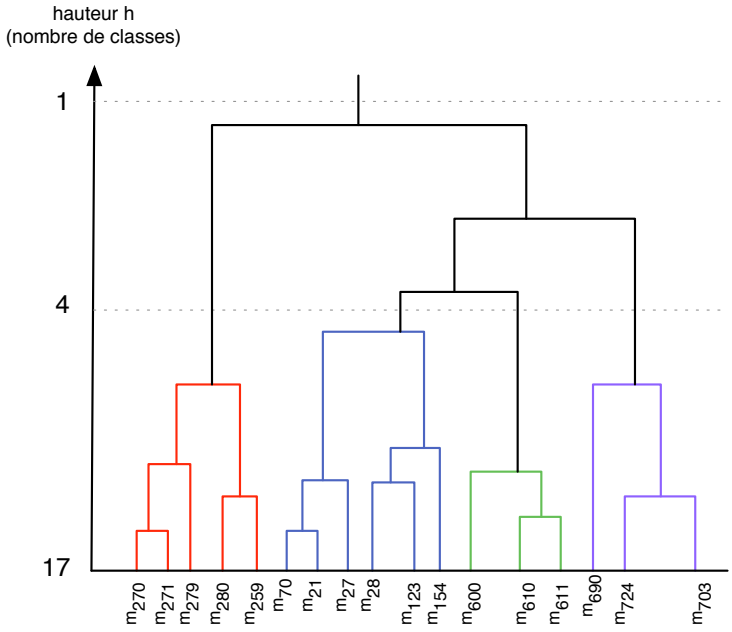


FIGURE 6.3 – Exemple de classification extraite d'un dendrogramme. L'arbre a été coupé à une hauteur correspondant à h=4 classes. Chacune des classes identifiée est représentée dans une couleur différente.

13. http ://www.r-project.org/

6.1.4 Evaluation des classifications

Lorsque les dendrogrammes ont été construits, l'analyse des résultats consiste à vérifier, pour chaque classification dans les dendrogrammes, la qualité des classes apprises. Comme nous disposons ici d'une classification de référence (les familles chimiques dans les chimiothèques), nous pouvons la comparer à la classification apprise.

Parmi les approches possibles, [Raymond *et al.*, 2003] utilise l'indice de Jaccard [Milligan, 1980] et une distance qui indique le nombre de molécules mal classées d'une classification à l'autre. [Gusfield, 2002] définit la distance entre deux classifications C et O comme le nombre minimal de molécules qu'il faut enlever aux deux classifications pour que les deux classifications induites soient identiques. [Stahl *et al.*, 2005] évalue les classifications en comptant le nombre et la taille des classes pures, le nombre de singletons ainsi que le nombre et la taille des classes mixtes. On trouve dans la littérature d'autres approches pour mesurer la qualité d'une classification [Gluck and Corter, 1985], [Corter and Gluck, 1992], [Youness, 2004] ou encore [Denoeud and Guénoche, 2006].

Dans notre cas, nous nous basons sur la *matrice de confusion* qui quantifie le résultat d'une classification (figure 6.4) et permet de mesurer la différence entre la classification C trouvée et la classification de référence O. Dans cette matrice, notée M, où les classes O_i correspondent aux familles originales et les classes C_j aux classes construites par l'algorithme de classification, chacune des valeurs $M_{i,j}$ représente le nombre de molécules simultanément présentes dans les classes O_i et C_j.

$$
\begin{array}{c}
 \\
O_1 \\
\dots \\
O_m \\
O_n \\
O_o \\
\dots \\
O_p
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
C_1 & \dots & & C_u & C_v & C_w & \dots & & C_q
\end{array} \\
\left(\begin{array}{ccccccccc}
M_{1,1} & & & 0 & 0 & 0 & & & \\
 & & & 0 & 0 & 0 & & & \\
0 & & 0 & \mathbf{10} & \mathbf{10} & 0 & 0 & & 0 \\
 & & & 0 & 0 & \mathbf{10} & & & \\
 & & & 0 & 0 & \mathbf{10} & & & \\
 & & & 0 & 0 & 0 & & & \\
 & & & 0 & 0 & 0 & & & M_{p,q}
\end{array}\right)
\end{array}
$$

FIGURE 6.4 – Exemple de matrice de confusion.

Deux classifications sont identiques lorsque cette matrice ne contient qu'une seule valeur non nulle pour chaque ligne et chaque colonne ; autrement dit, la matrice est diagonale à un ensemble de permutations près. Une manière simple de quantifier la qualité d'une classification est alors d'évaluer les entropies moyennes associées aux lignes et aux colonnes de la matrice M. Toutefois, il est nécessaire de distinguer les lignes et les colonnes car elles représentent des informations de nature différente. En

effet, au cours d'une classification, deux phénomènes différents peuvent se produire, éventuellement de façon simultanée :

- une *erreur de segmentation* lorsqu'une famille initiale est divisée en plusieurs classes. C'est le cas par exemple de la famille O_m qui a simplement été divisée en deux nouvelles classes C_u et C_v. Cette classification n'est pas incorrecte dans le sens où chaque classe reste cohérente, c'est-à-dire que toutes les molécules qu'elle contient correspondent à une même famille ;
- une *erreur de confusion* si plusieurs familles originales sont fusionnées en une seule classe. C'est le cas de la colonne C_w où les deux classes initiales O_o et O_n sont confondues, ce qui est plus gênant car des molécules de familles différentes sont regroupées.

Pour évaluer l'importance de ces phénomènes, nous considérons les deux mesures suivantes :

- l'Indice de Confusion (IC) qui quantifie le nombre de classes initiales fusionnées en une seule ;
- l'Indice de Segmentation (IS) qui quantifie le nombre de classes initiales subdivisées en plusieurs classes.

Comme il a été mentionné ci-dessus, si aucune erreur de confusion (resp. de segmentation) n'a été commise, alors chaque colonne (resp. ligne) ne contient qu'une seule valeur non nulle ce qui peut être quantifié par l'entropie conditionnelle de chaque colonne (resp. ligne). Soit $H(O|C_j)$, l'entropie conditionnelle de la ligne O par rapport à la colonne C_j, définie par :

$$H(O|C_j) = -\sum_{i=1}^{p} \frac{M_{i,j}}{|C_j|} \cdot log_2\left(\frac{M_{i,j}}{|C_j|}\right) \tag{6.1}$$

où $|O_i|$ est la somme des valeurs de la ligne O_i. De même, l'entropie conditionnelle $H(C|O_i)$ de la colonne C étant donné la ligne O_i, s'écrit :

$$H(C|O_i) = -\sum_{j=1}^{q} \frac{M_{i,j}}{|O_i|} \cdot log_2\left(\frac{M_{i,j}}{|O_i|}\right) \tag{6.2}$$

où $|C_j|$ est la somme des valeurs de la colonne C_j.

L'indice IC est alors défini comme la moyenne pondérée des entropies conditionnelles de chacune des q colonnes dans la matrice de confusion :

$$IC = \frac{1}{N} \sum_{j=1}^{q} |C_j|.H(O|C_j) \tag{6.3}$$

où N est la somme de toutes les valeurs de la matrice M. De manière duale, l'indice IS est défini par la moyenne pondérée des entropies conditionnelles de chacune des p lignes :

$$IS = \frac{1}{N} \sum_{i=1}^{p} |O_i|.H(C|O_i) \tag{6.4}$$

Suivant les valeurs de IC et IS, nous pouvons répertorier quatre situations extrêmes :

IC	IS	Interprétation
$\rightarrow log_2(p)$	$\rightarrow log_2(q)$	Les deux classifications sont totalement orthogonales.
	$\rightarrow 0$	La classification trouvée est plus générale que la classification de référence : certaines molécules de familles différentes ont été rassemblées dans les mêmes classes.
$\rightarrow 0$	$\rightarrow log_2(q)$	La classification trouvée est plus fine que la classification de référence : certaines molécules d'une même famille ont été classées dans des classes différentes.
	$\rightarrow 0$	Les deux classifications (celle apprise par l'algorithme de classification et celle de référence) sont identiques.

L'indice le plus important reste toutefois l'indice IC puisqu'il indique si la classification initiale a été retrouvée par l'algorithme de classification.

Pour chaque tuple <chimiothèque, variante de I_π>, les indices IC et IS correspondants sont tracés en fonction du nombre h de classes construites (*i.e.* la hauteur de coupe dans le dendrogramme). Les fonctions $IC = f(h)$ et $IS = f(h)$ sont respectivement décroissante et croissante. Pour $h = 1$ (une seule classe contient tous les composés de la base), la valeur de IC (resp. IS) est maximale (resp. nulle) tandis que pour $h = N$ (N classes contenant chacune un seul composé), la valeur de IC (resp. IS) est nulle (resp. maximale). Cette représentation permet également d'évaluer la qualité des classifications trouvées : plus la décroissance de l'indice IC est rapide, plus vite l'algorithme trouve de bonnes classifications. Enfin, le point où $h = NRef$, qui correspond au nombre de classes originales, est particulièrement intéressant car, à cet endroit du dendrogramme, une classification sans erreur donne $IC = IS = 0$.

6.2 Comparaison des variantes de I_π

Ainsi que nous l'avons écrit dans la section 6.1.2, nous étudions ici les différentes variantes de l'algorithme des APC (voir section 3.3) en combinant la stratégie d'appariement local (stochastique, gloutonne ou hongroise) avec le critère d'arrêt de l'algorithme.

Pour chaque variante, l'algorithme I_π génère un appariement pour chacune des graines possibles et retient celui qui a la plus forte valeur du degré d'inclusion. L'évaluation des classifications par les indices IC est présentée dans les figures 6.5 et 6.6. [14]

La première constatation que l'on peut faire en observant les courbes est que les résultats sont assez différents d'une base à l'autre. Sur les bases BZR et COX2, les résultats produits par les différentes variantes de Ipi ont des profils similaires, ce qui n'est pas le cas des bases DHFR et ER. Pour BZR, la valeur de l'indice IC est quasiment identique pour toutes les variantes à l'endroit du nombre de classes originaux ; la différence entre les courbes étant un peu plus importante sur la base COX2. Pour la base DHFR, les courbes sont nettement différentes. Enfin, pour la base ER, plusieurs classifications se superposent : les deux variantes stochastiques ainsi que APC_Glouton_AvecArret sont identiques, de même que les trois autres variantes.
Les valeurs numériques de l'indice IC correspondant au nombre h de classes dans la classification de référence sont indiquées dans le tableau 6.2.

Variantes de I_π	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
APC_Kuhn_AvecArret	0.36	0.23	0.99	0.21
APC_Kuhn_SansArret	0.35	0.35	1,03	0.21
APC_Glouton_AvecArret	0.38	**0.13**	**0.46**	**0.04**
APC_Glouton_SansArret	0.35	0.35	0.95	0.21
APC_Stochast_AvecArret	**0.34**	0.17	**0.46**	0.21
APC_Stochast_SansArret	**0.34**	0.31	0.49	0.21

Tableau 6.2 – Valeurs de l'indice IC des variantes de l'algorithme I_π pour $h = NRef$. Les meilleures valeurs (les plus petites) sont indiquées en caractères gras.

Les méthodes les plus performantes sur la base BZR sont les deux versions stochastiques de l'algorithme puis viennent les deux méthodes sans arrêt (APC_Glouton_SansArret et APC_Kuhn_SansArret). Pour les bases COX2 et DHFR, les meilleures méthodes sont les variantes APC_*_AvecArret, l'écart entre les variantes est plus important sur DHFR que sur COX2.

14. Pour l'ensemble des résultats obtenus dans ce travail, l'ordre des courbes pour l'indice IS est identique à celui des courbes de l'indice IC. L'indice IC étant le plus important, nous ne présentons pas ici les courbes correspondant à l'indice IS.

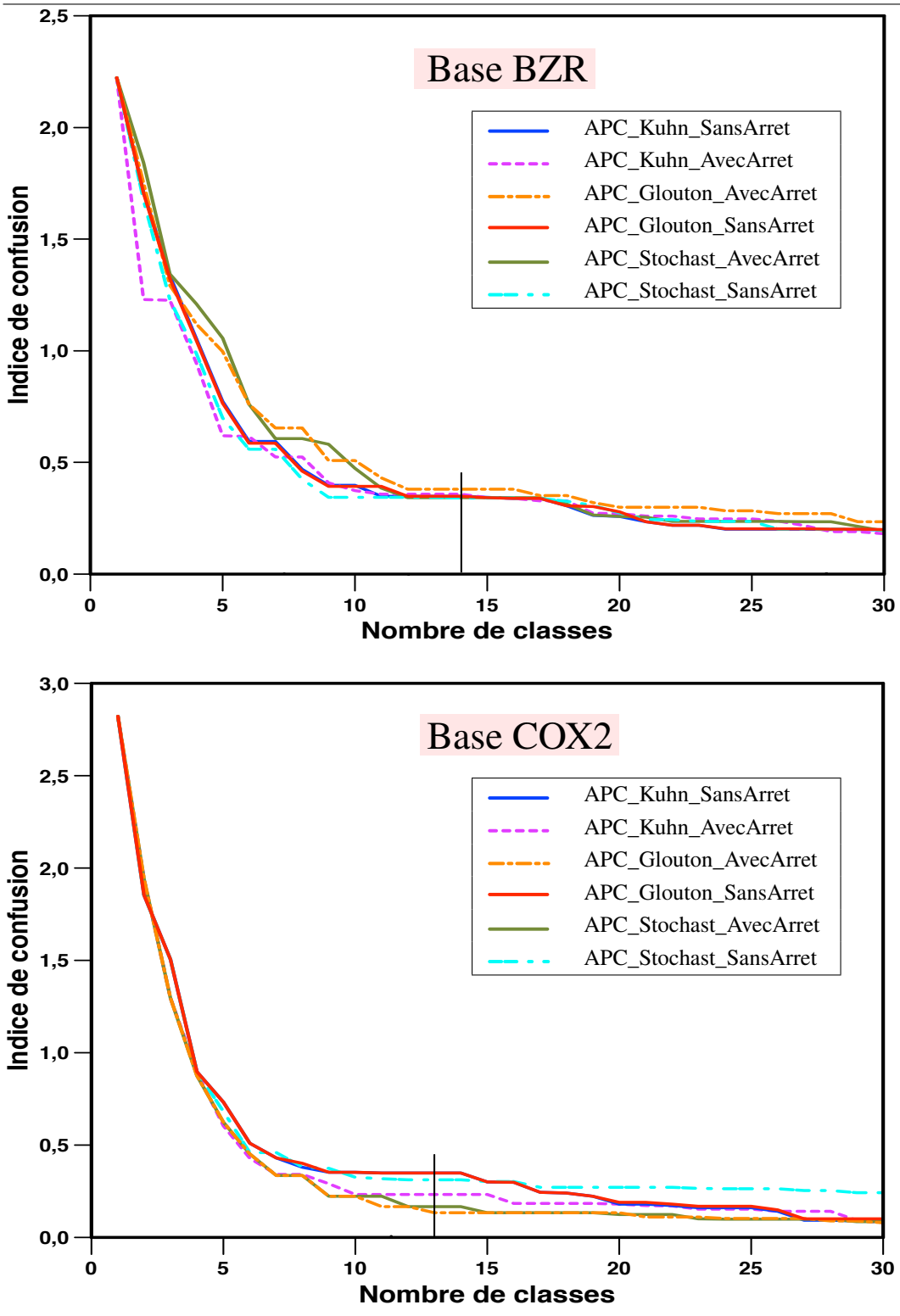


FIGURE 6.5 – Indice IC des variantes de l'algorithme I_π pour les bases BZR et COX2.

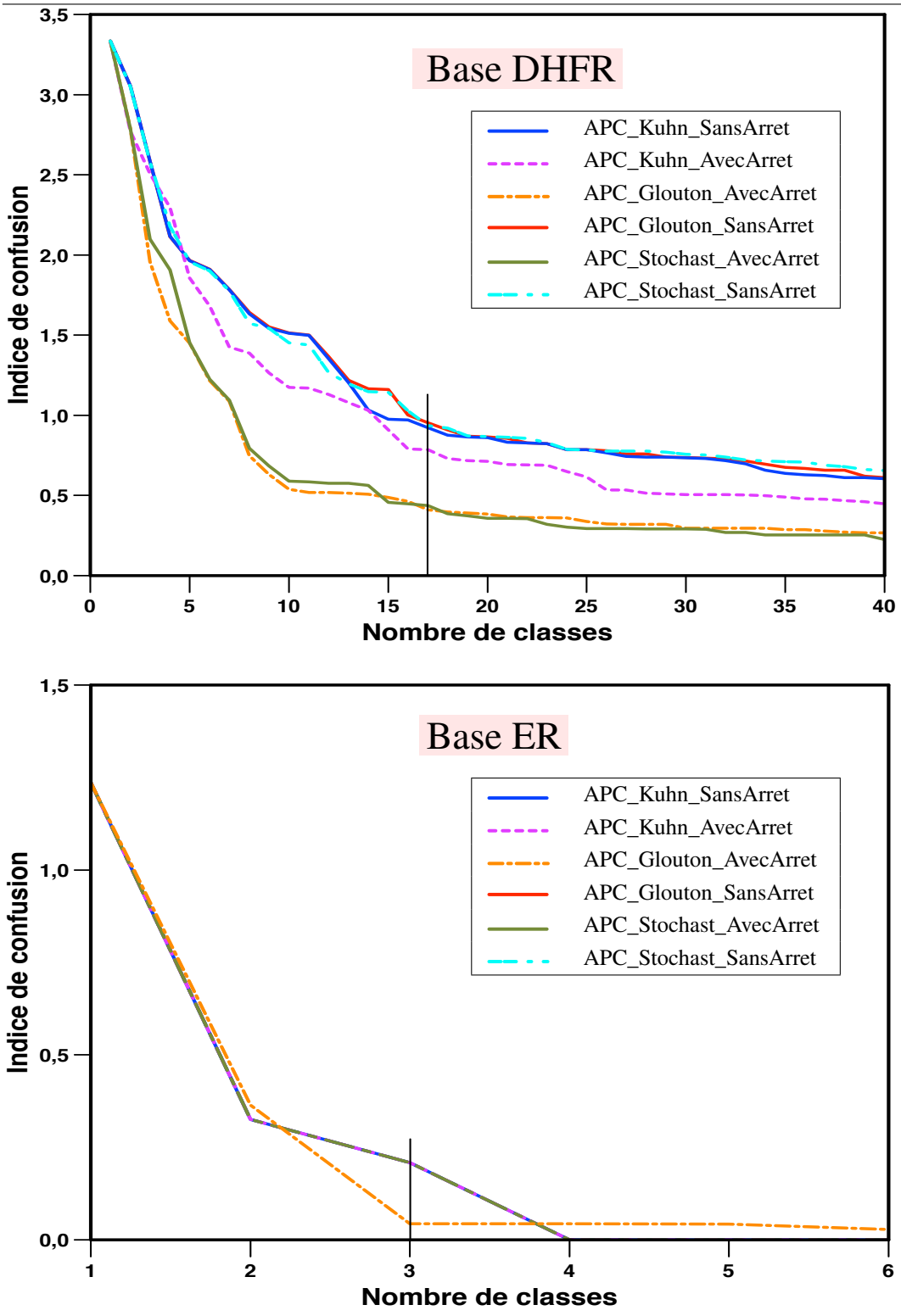


FIGURE 6.6 – Indice IC des variantes de l'algorithme I_π pour les bases DHFR et ER. Sur ER, toutes les courbes, exceptée celle de la variante APC_Glouton_AvecArret, sont superposées.

En ce qui concerne la base ER, c'est la variante APC_Glouton_AvecArret qui trouve plus rapidement une bonne classification : pour $h = 3$ classes, la valeur de IC est plus faible qu'avec les autres variantes. Par contre, dès l'identification de 4 classes, la tendance s'inverse : toutes les autres variantes retrouvent la classification originale sans erreur de confusion ($IC = 0$) alors que des erreurs subsistent pour APC_Glouton_AvecArret ($IC \neq 0$).

De ces premiers résultats, on peut remarquer que l'algorithme d'appariement local stochastique semble important pour une bonne classification sur la base BZR alors que ce sont les variantes les versions "AvecArrêt" qui sont les plus performantes sur les bases COX2 et DHFR. En regardant les deux paramètres en même temps, on observe que, sur COX2 et DHFR, les variantes APC_Glouton_AvecArret et APC_Stochast_AvecArret donnent les meilleurs résultats.

6.2.1 Influence de la stratégie d'appariement local

En ce qui concerne les variantes des appariements locaux [15], il peut sembler surprenant à première vue que les variantes stochastiques soient aussi performantes voire parfois meilleures que les variantes qui utilisent l'algorithme hongrois ou l'algorithme glouton. On pourrait en effet penser qu'un meilleur appariement des atomes dans les molécules (surtout avec l'algorithme hongrois) permettrait de mieux apparier les molécules et conduirait ainsi à une meilleure classification.

Pour étudier le comportement des algorithmes utilisés pour les appariements locaux (variantes APC_Stochast_SansArret, APC_Kuhn_SansArret et APC_Glouton_SansArret), nous analysons en détail le déroulement de l'algorithme de recherche d'un APC sur un exemple concret en focalisant notre attention sur les ensembles d'atomes voisins identifiés à chaque étape de l'algorithme, ceux-là même qui sont utilisés pour les algorithmes d'appariements locaux. La figure 6.7 illustre les deux molécules de la base DHFR (m_1 et m_{207}) que nous prenons pour l'exemple.

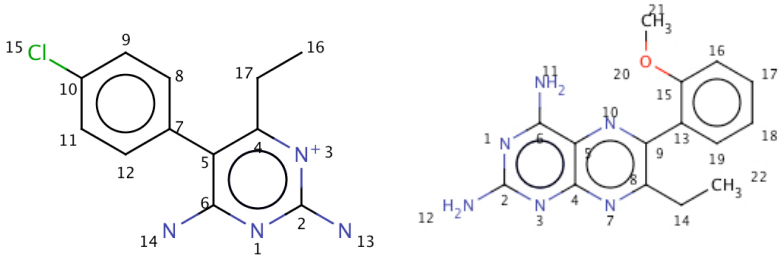


FIGURE 6.7 – Deux molécules (m_1 et m_{207}) de la base DHFR.

15. c'est-à-dire les différentes stratégies pour apparier les atomes voisins lors de la recherche d'un APC.

Le déroulement, étape par étape, de la recherche d'un APC est détaillé dans le tableau 6.3 où chaque ligne correspond à l'appariement d'une paire d'atomes a_i dans la molécule m_1 (partie de gauche) et a'_m dans la molécule m_{207} (partie de droite). Le tableau se lit de la manière suivante :

1. Atome de m_1 : numéro de l'atome qui est apparié ;
2. V : ensemble des voisins de l'atome indiqué dans la colonne précédente ;
3. V^* : sous-ensemble de V des voisins qui ne sont pas encore appariés. Cet ensemble est divisé en sous-ensembles suivant la nature de l'atome voisin et le type de liaison entre un voisin donné et l'atome qui est apparié (cf première colonne). Le code en début de sous-ensemble indique la valence de la liaison et le type d'atome du voisin. Par exemple, si on considère une liaison double entre les paires d'atomes (1, 2) et (1, 3), tous deux étant des atomes de carbone, l'ensemble des voisins de l'atome 1 s'écrit : 2C(2, 3) ;
4. Atome de m_{207} : numéro de l'atome de la molécule m_{207} qui est apparié avec celui de la colonne intitulée "Atome de m_1" ;
5. V' : ensemble des voisins de l'atome indiqué dans la colonne précédente ;
6. V'^* : sous-ensemble de V' des voisins de l'atome indiqué dans la colonne 4, qui ne sont pas encore appariés. Le code est identique à celui de la colonne 3.

Sur chaque ligne, la paire V et V' définit un graphe bipartite caractérisé par le nombre d'atomes à apparier dans chacun des ensembles. Dans le cas des ensembles de voisins filtrés V^* et V'^*, on doit tenir compte des sous-ensembles de voisins identifiés par un même code et générer un graphe bipartite par code. Par exemple, pour les ensembles $V^* = \{4\mathrm{N}(1), 4\mathrm{C}(13)\}$ et $V'^* = \{4\mathrm{N}(3), 4\mathrm{C}(5)\}$, deux graphes bipartites sont identifiés : le premier contient les éléments 4N(1) et 4N(3), le deuxième est composé de 4C(13) et 4C(5).

Ce qui nous intéresse ici est le nombre d'atomes à apparier car cela permet de déterminer le nombre de combinaisons possibles dans le graphe bipartite. Chaque graphe bipartite est donc caractérisé par son *type* noté (*m_n*) où *m* (resp. *n*) est le nombre d'atomes du premier ensemble (V ou V^*) (resp. du deuxième ensemble V' ou V'^*).

Atome de m_1	V	V^*	Atome de m_{207}	V'	V'^*
9	{8, 10}	**4C(8, 10)**	18	{17, 19}	**4C(17, 19)**
8	{9, 7}	4C(7)	19	{13, 18}	4C(13)
10	{9, 11, 15}	1Cl(15) 4C(11)	17	{16, 18}	4C(16)
7	{5, 8, 12}	4C(12) 1C (5)	13	{15, 9, 19}	4C (15) 1C(9)
5	{4, 6, 7}	**4C(4, 6)**	9	{10, 8, 13}	**4C(8)** 4N(10)
12	{7, 11}	4C(11)	15	{13, 16, 20}	4C(16) 1O(20)
4	{3, 5, 17}	1C(17) 4C(3)	8	{7, 9, 14}	1C(14) 4N(7)
3	{2, 4}	4C(2)	7	{4, 8}	4C(4)
17	{4, 16}	1C(16)	14	{8, 22}	1C(22)
2	{1, 3, 13}	4N(1), 1N(13)	4	{3, 5, 7}	4N(3) 4C(5)
16	{17}	-	22	{14}	-
1	{2, 6}	4C(6)	3	{2, 4}	4C(2)
6	{1, 5, 14}	**1N(14)**	2	{1, 3, 12}	4N(1), **1N(12)**
14	{6}	-	12	{2}	-
11	{10, 12}	-	16	{15, 17}	-

TABLEAU 6.3 – Etapes de la recherche d'un APC entre les molécules m_1 et m_{207} de la base DHFR.

Les graphes bipartites issus des ensembles de voisins peuvent être classés en fonction de l'algorithme "nécessaire et suffisant" pour résoudre le problème d'appariement posé. Lorsque $m = 0$, $n = 0$ ou $n = m = 1$, il n'y a en fait aucun choix d'appariement à effectuer : l'algorithme stochastique est alors nécessaire et suffisant ; les deux autres stratégies ne pouvant pas faire mieux. Dans l'exemple ci-dessus, l'appariement de la paire d'atomes (16, 22) ou de la paire (14, 12) (en gris dans le tableau) est trivial car il n'y a aucun choix possible. Les algorithmes glouton et hongrois sont nécessaires lorsque l'une des variables est égale à 1 et l'autre est supérieure ou égale à 2. Dans tous les autres cas ($n, m \geq 2$), il est nécessaire d'utiliser l'algorithme hongrois pour trouver l'appariement optimal comme c'est le cas pour la première étape dans l'exemple où le graphe bipartite est de type (2_2). Ceci n'est plus vrai si les degrés d'inclusion associés aux appariements à effectuer sont tous identiques. Dans ce cas, même si $n, m \geq 2$, l'algorithme hongrois ne pourra pas faire mieux qu'un simple algorithme stochastique.

Dans cet exemple, on voit que la recherche d'un APC diminue de façon importante le nombre de graphes bipartites nécessitant une optimisation. L'utilisation de l'algorithme hongrois n'est vraiment justifiée que dans 20% des cas (3 appariements sur les 15 réalisés). Pour les 80% autres cas, les algorithmes glouton et hongrois ne pouvent pas faire mieux que l'algorithme stochastique simplement parce que soit il n'y a pas de choix d'appariements possibles dans les graphes bipartites (la proportion de graphes bi-

partites à optimiser est en pratique relativement faible), soit les résultats sont identiques à ceux obtenus avec l'algorithme stochastique.

Ceci étant, cela n'explique en aucun cas pourquoi l'algorithme hongrois est moins bon (tableau 6.2) que les autres variantes dans 3 bases sur 4 (si l'on considère qu'il y a égalité pour BZR). Une hypothèse à vérifier et qui pourrait expliquer ces résultats est que le choix optimal (algorithme hongrois) est plutôt systématiquement faux et, dans le cas de DHFR, nettement plus que l'algorithme glouton. En d'autres termes, cela signifierait qu'il vaut mieux apparier correctement 1 ou 2 voisins plutôt que moyennement tous.

Nous allons maintenant observer plus généralement, sur chacune des chimiothèques, la distribution des types de graphes bipartites générés lors de la recherche d'un APC. Les histogrammes correspondants sont illustrés sur la figure 6.8. Les types nécessitant un algorithme hongrois (2_2, 2_3, 2_4, 3_3, 3_4, 4_4) sont en rouge. Ceux pour lesquels un algorithme glouton est suffisant (1_2, 1_3, 1_4) sont en vert. Enfin, les types de graphes bipartites utilisables avec un algorithme stochastique (1_1) sont en bleu.

Sur les quatre bases, on observe globalement la même tendance. Sur les histogrammes de BZR, COX2 et DHFR à gauche, la majorité des graphes bipartites sont de type 2_2 ; les autres types fréquents sont les types 1_1 et 1_2. Dans ER, les types 2_2 et 1_2 sont en nombre égal, les cas 1_1, 1_3 et 2_3 sont aussi nombreux. Les graphes bipartites de type 1_4, 2_4, 3_3 et 3_4 sont par ailleurs en plus grand nombre dans cette base que dans les trois précédentes. En l'absence de toute stratégie réduisant le nombre d'atomes à apparier dans les graphes bipartites, on voit qu'il est nécessaire d'utiliser l'algorithme hongrois pour trouver l'appariement optimal en raison du grand nombre de graphes bipartites avec $m, n \geq 2$.

Lors de la recherche d'APC, la distribution des types de graphes bipartites est très différente (histogrammes à droite sur la figure 6.8). Dans les histogrammes de gauche, on considère l'ensemble des possibilités d'appariements entre les atomes indépendamment de toute contrainte et l'on observe de nombreux graphes bipartites de type 2_2.

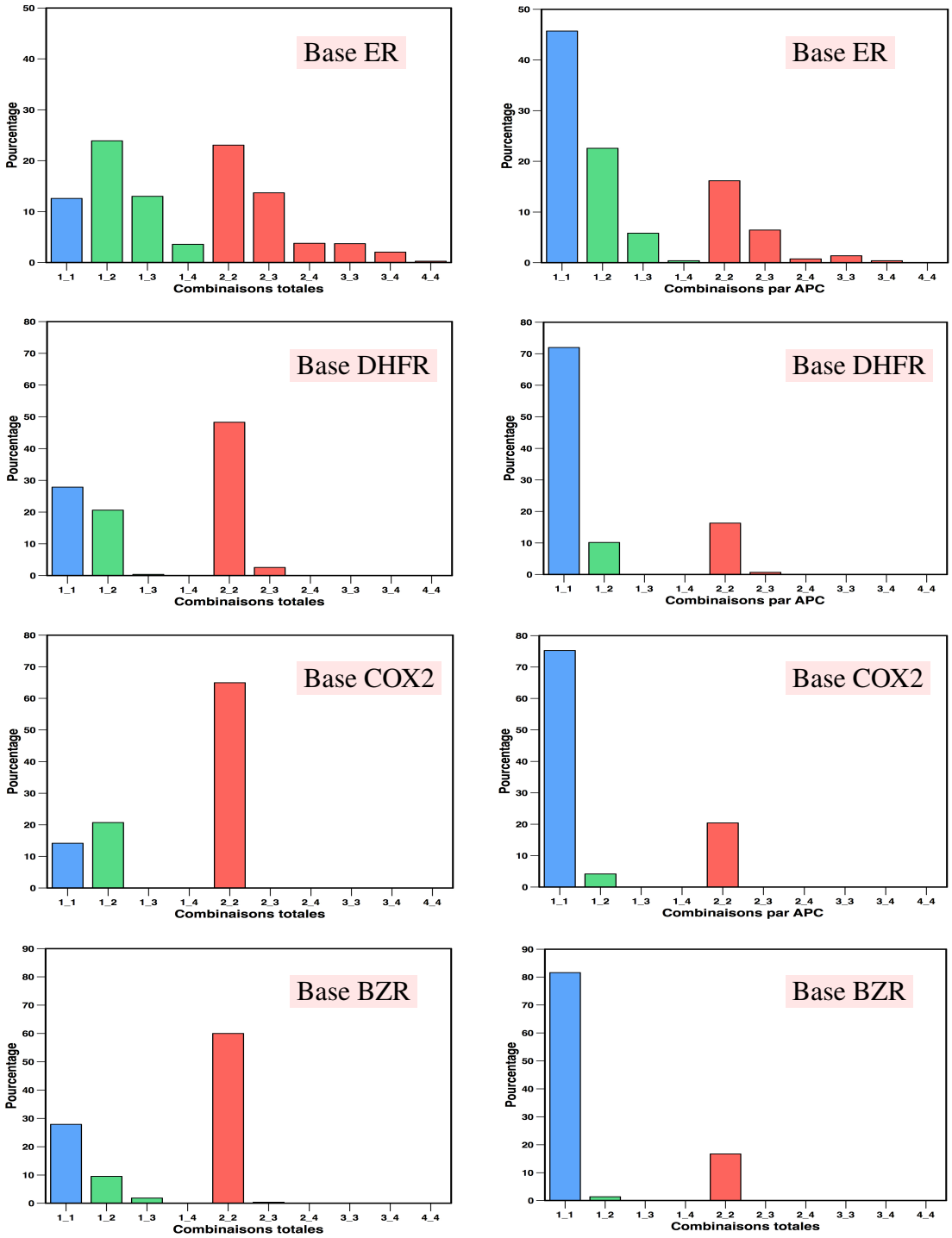


FIGURE 6.8 – Distribution des types de graphes bipartites pour l'appariement des voisins. A gauche, les histogrammes correspondent aux combinaisons des ensembles V et V'. A droite, les histogrammes correspondent aux combinaisons effectives relevées lors du parcours des graphes à partir des ensembles V^* et V'^*.

A droite, lorsque l'on utilise la recherche d'APC, hormis pour le premier appariement (celui de la graine), à chaque étape de l'algorithme, il y a déjà des atomes appariés, ce qui réduit le nombre de possibilités d'appariement. Un certain nombre de graphes bipartites initialement de type 2_2 sont donc transformés en type 1_1, ce qui explique la diminution des graphes bipartites de type 2_2 et l'augmentation des graphes bipartites de type 1_1. Quelle que soit la chimiothèque, les graphes bipartites de type 1_1 sont maintenant majoritaires avec plus de 70% des graphes bipartites identifiés pour les bases BZR, COX2 et DHFR ; les graphes bipartites de type 2_2 ne représentant qu'environ 20%. Pour ER, les graphes bipartites nécessitant un algorithme glouton ou hongrois (types 1_2 surtout, et 1_3) sont plus fréquents que dans les autres bases.

La grande proportion de graphes bipartites de type 1_1 dans l'algorithme d'APC peut donc expliquer les bonnes performances des variantes stochastiques car les possibilités d'erreur d'appariement (typiquement sur des graphes bipartites de type 2_2) sont réduites.

6.2.2 Influence de la stratégie d'arrêt sur erreur

Pour comprendre l'impact des critères d'arrêt de l'algorithme (arrêt sur erreur d'appariement ou arrêt lorsqu'il n'y a plus d'atomes à apparier dans la file First In First Out (FIFO)), nous allons examiner quelques exemples de molécules mal classées pour analyser les raisons de ces erreurs de classification.

6.2.2.1 Analyse d'une erreur de confusion sur DHFR

Dans la classification initiale de la base DHFR, les molécules m_{224} et m_{277} appartiennent à la même famille *G* tandis que la molécule m_{370} appartient à la famille *I*. Ces molécules ont été correctement classées par les variantes avec arrêt (APC_*_AvecArret) alors qu'une erreur a été commise par les variantes sans arrêt (APC_*_SansArret) : les molécules m_{224} et m_{370} ont été regroupées dans la même classe tandis que la molécule m_{277} a été classée à part. Cet exemple est illustré sur la figure 6.9.

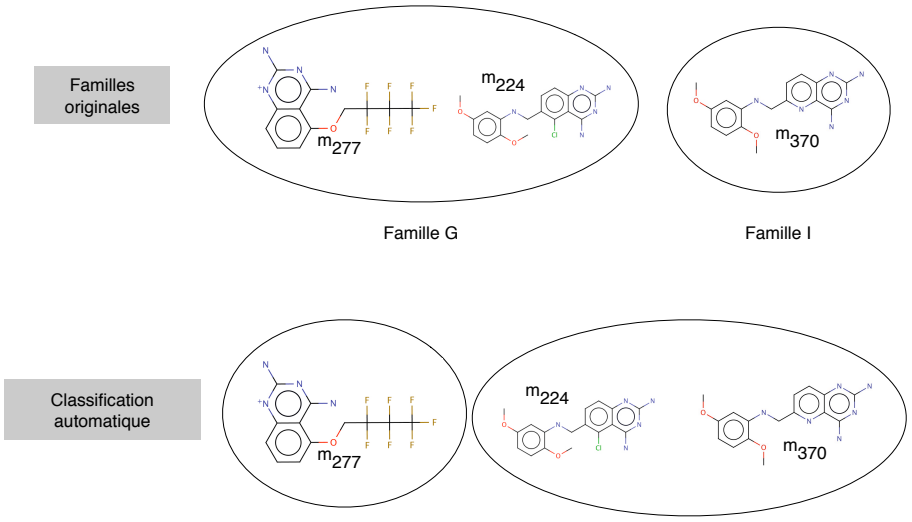
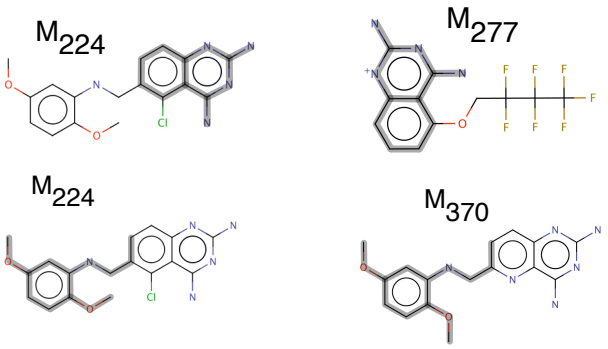
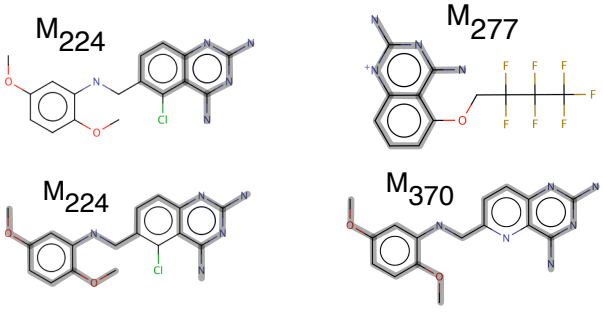


FIGURE 6.9 – Exemple d'erreur de confusion dans la base DHFR.



Variantes APC_*_AvecArret.



Variantes APC_*_SansArret.

FIGURE 6.10 – Plus grandes SSC obtenues avec les variantes APC_*_SansArret et APC_*_AvecArret Les liaisons grisées sont celles qui correspondent à la SSCM identifiée entre les paires de molécules.

Si l'on regarde les SSC trouvées entre les molécules m_{224} et m_{277} issues de la même famille (figure 6.10) par les deux variantes, on s'aperçoit qu'elles sont identiques dans les deux cas ; la similarité entre ces deux molécules est donc identique selon que l'on utilise les variantes APC_*_AvecArret ou APC_*_SansArret. Par contre, la SSC entre les molécules m_{224} et m_{370} est plus petite avec les variantes APC_*_AvecArret et la similarité est alors plus faible que celle donnée par l'algorithme sans arrêt. Dans les variantes APC_*_AvecArret, on a donc $\text{Sim}_{I_\pi}(m_{224}, m_{370}) < \text{Sim}_{I_\pi}(m_{224}, m_{277})$ tandis que dans les variantes APC_*_SansArret, c'est l'inverse : $\text{Sim}_{I_\pi}(m_{224}, m_{370}) > \text{Sim}_{I_\pi}(m_{224}, m_{277})$. Dans ces conditions, l'algorithme de classification a naturellement placé les molécules m_{224} et m_{370} dans la même classe.

On voit, sur cet exemple, que lorsque la construction des APC est arrêtée dès qu'un atome ne peut pas être apparié, les SSC identifiées sont plus petites que celles qui sont générées par les versions APC_*_SansArret. Par ailleurs, dans les variantes APC_*_AvecArret, la SSCM identifiée contient des sous-structures chimiquement significatives : en particulier, les cycles aromatiques sont complets alors que dans les versions APC_*_SansArret, il y a un cycle qui n'est pas complet. Les versions APC_*_SansArret ont ainsi tendance à "contourner" les cycles pour poursuivre l'appariement, ce qui conduit effectivement à des sous-structures plus grandes mais qui n'ont finalement pas toutes un sens chimique. C'est pourquoi, dans le cas des bases COX2 et DHFR, l'identification d'une "petite" SSC de bonne qualité (d'un point de vue chimique) conduit à moins d'erreurs de classification que l'identification d'une sous-structure certes plus grande (valeurs de IC plus petites dans le premier cas) mais qui est composée d'éléments non significatifs.

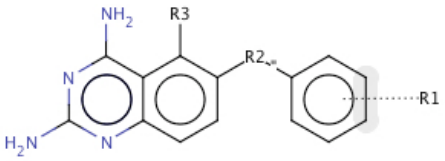
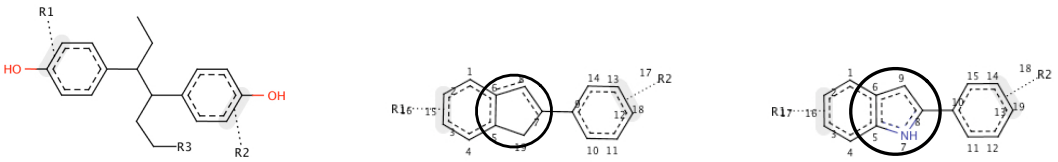


FIGURE 6.11 – Châssis moléculaire de la famile G dans la base DHFR.

6.2.3 Influence de la représentation de la structure moléculaire

En recherchant les molécules mal classées dans la base ER pour $h = 3$ classes, on remarque que l'ensemble des molécules appartenant à la sous-famille des indènes (sous-famille C_1) sont systématiquement mal classées : ces molécules ont été classées avec les hexestrols (sous-famille B_2) alors qu'elles auraient dues être classées avec les indoles (sous-famille C_2). Les indoles ont été considérées comme une classe à part

entière. Pour la variante APC_Glouton_AvecArret, ce sont les seules erreurs de classification : si elles n'avaient pas été commises, l'algorithme aurait trouvé une classification sans aucune confusion dès $h = 3$ classes. Il est donc très intéressant de connaître les causes de ces erreurs. La figure 6.12 montre les châssis moléculaires des sous-familles concernées.



(a) Hexestrols (b) Indènes/Indénones (c) Indoles

FIGURE 6.12 – Châssis moléculaires de trois sous-familles dans la base ER. Les hexestrols correspondent à la sous-famille B_2, les indènes/indénones à la sous-famille C_1 et les indoles à la sous-famille C_2.

Au cours de la standardisation des molécules, les cycles aromatiques ont été aromatisés (les liaisons appartenant à un cycle aromatique ont été délocalisées), ce qui se représente sous la forme d'un cercle en pointillés à l'intérieur du cycle. Au niveau de la représentation informatique, ces liaisons ont une valence égale à 4 (codification de la valence des liaisons aromatiques délocalisées). On remarque notamment (figure 6.12) que le cycle central des indoles est aromatique contrairement à celui des indénones.

Lors de la recherche d'un APC entre les indènes et les indoles, le parcours des graphes est stoppé sur le cycle central (composé de 5 atomes) qui apparaît différent au sens de la représentation des structures : la valence des liaisons n'est pas la même et donc, l'algorithme ne trouve plus de voisins à apparier. Supposons que l'algorithme ait déjà apparié le groupe R2 et le cycle aromatique attenant, il va ensuite apparier l'atome 7 du cycle central des indènes avec l'atome 8 des indoles. A l'étape suivante, il cherchera à apparier les voisins de l'atome 7 : {8, 19} avec ceux de l'atome 8 : {7, 9}. Or, il n'y a aucune liaison compatible car dans les indoles, elles sont toutes de valence 4 : l'algorithme s'arrête alors car la file FIFO contenant les atomes à apparier est vide. Dans ce cas, quelle que soit la variante utilisée (avec ou sans arrêt sur erreur), l'algorithme s'arrête et les SSC identifiées sont identiques.

Par contre, en appariant les atomes des indènes avec ceux des hexestrols, le comportement est différent selon le critère d'arrêt. Avec les variantes APC_*_AvecArret, l'algorithme s'arrête avant de parcourir le cycle central des indènes. Dans les variantes APC_*_SansArret, le cycle central des indènes est partiellement apparié (il est 'contourné') et l'algorithme poursuit l'exploration du graphe moléculaire sur le cycle suivant. La SSC identifiée entre les hexestrols et les indènes est alors plus grande que celle identi-

fiée entre les indènes et les indoles (figure 6.13), ce qui entraîne une similarité différente dans les deux cas.

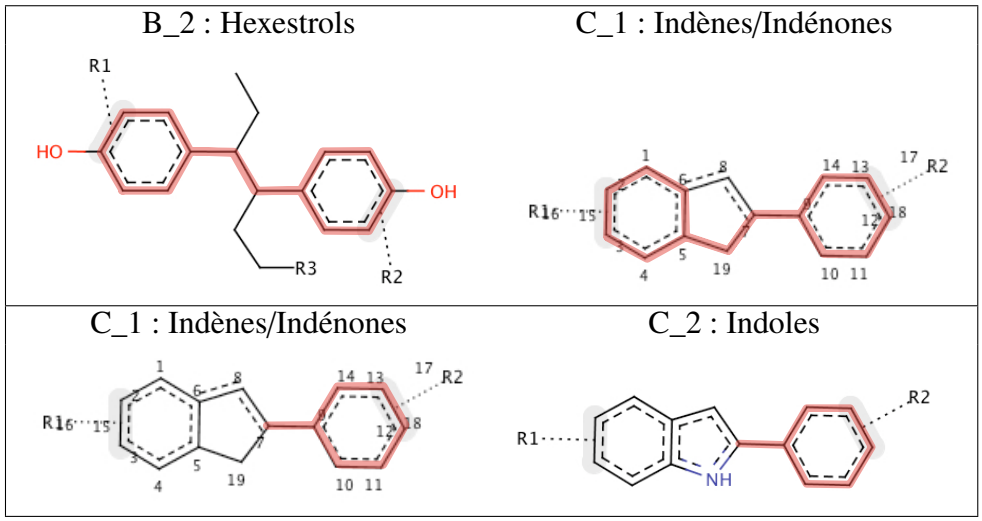


FIGURE 6.13 – En haut, la SSCM entre les hexestrols et les indènes. En bas, la SSCM entre les indènes et les indoles de la base ER.

Le problème rencontré est que, dans certains cas, des cycles peuvent être partiellement explorés alors que l'algorithme continue à apparier des atomes sans savoir qu'il génère une sous-structure qui n'a pas de signification chimique. Dans l'exemple de la figure 6.13, lorsque le cycle central des indènes n'est pas contourné, il n'y a pas d'erreur de classification. En chimie où les structures moléculaires sont elles-mêmes constituées de sous-structures ayant un sens bien défini, le critère d'arrêt sur erreur fonctionne mais est arbitraire dans le sens où il n'est pas directement associé à une hypothèse chimique.

Dans l'exemple de la figure 6.14, les différentes SSC entre deux molécules sont présentées : certaines sont correctes d'un point de vue chimique mais la plupart ne le sont pas. Bien évidemment, rien ne garantit que la plus grande SSC, celle dont on tient compte pour le calcul de l'indice I_π soit correcte.

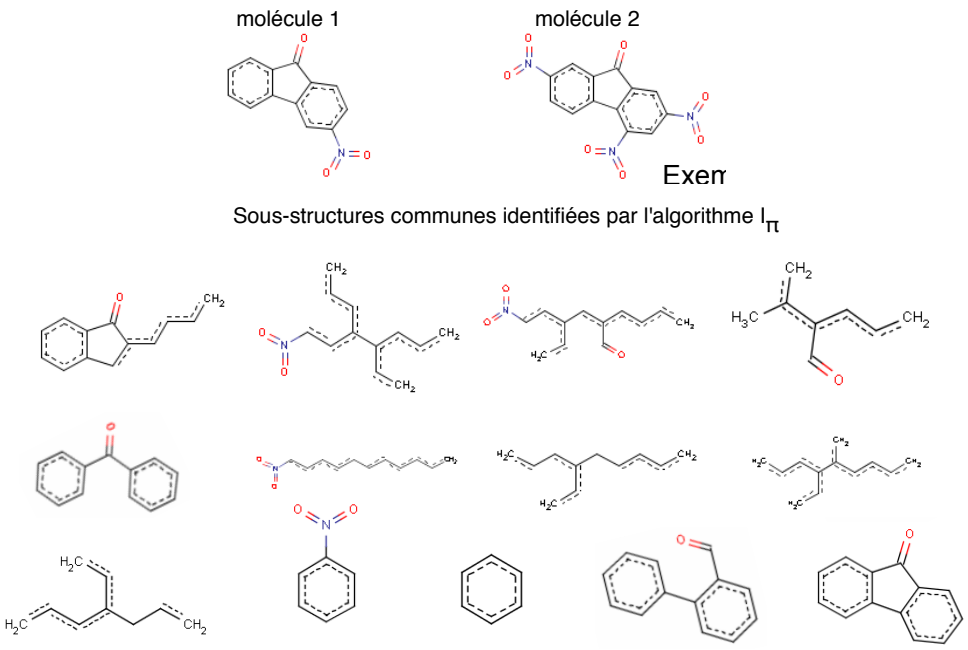


FIGURE 6.14 – Exemple de SSC entre deux molécules obtenues à partir de graines différentes.

Pour limiter le nombre d'appariements générant une SSC non significative chimiquement parlant, une solution pourrait consister à stopper l'appariement uniquement lorsqu'un atome appartenant à une sous-structure significative ne peut pas être apparié, ce qui permettrait de ne conserver que des éléments cohérents d'un point de vue chimique. Dans le cas des cycles aromatiques par exemple, il suffit pour cela de vérifier le type de liaison dans laquelle l'atome qui ne peut pas être apparié est impliqué : s'il s'agit d'une liaison aromatique, l'algorithme est alors arrêté. Le problème se complexifie lorsque plusieurs cycles ont des atomes partagés : on a alors le choix de ne conserver d'un seul cycle fermé ou plusieurs et dans ce cas, il faudrait pouvoir savoir lesquels conserver.

L'avantage de cette approche "détection de cycle non-fermé" est qu'elle peut être conçue comme un paramètre de la méthode d'appariement global. Dans le cas de la chimie, c'est une voie qui parait plus intéressante à explorer que l'arrêt sur échec (même si celui-ci peut aussi être un paramètre parfois intéressant) qui présente l'inconvénient important de dépendre de la graine d'appariement initiale.

6.2.4 Influence du choix des graines pour la recherche des APC

Nous allons maintenant étudier le dernier paramètre de la méthode I_π qui est la méthode de sélection des graines utilisées pour initier la recherche des APC (section 3.4.3) ; les stratégies proposées sont décrites dans la section suivante.

L'objectif de cette étude est de vérifier si notre hypothèse qui énonce qu'une graine ayant un grand degré d'inclusion a de grandes chances de générer une grande SSC est vraie et si une valeur maximale du degré d'inclusion entre deux molécules correspond toujours à une classification optimale. Cette expérience permet également de vérifier si l'échantillonnage des graines est suffisant pour capturer les graines intéressantes, c'est-à-dire celles qui vont conduire à des appariements permettant une bonne classification.

Pour réaliser ce test, deux variantes de l'algorithme I_π sont utilisées : la variante stochastique où l'appariement local entre les voisins lors de la recherche d'APC est réalisé de manière aléatoire et la variante utilisant l'algorithme hongrois pour cet appariement local. Les deux variantes sont sans arrêt sur erreur.

6.2.4.1 Stratégies de sélection des graines

Pour choisir les graines utilisées pour initier la recherche d'APC, deux approches sont possibles. La première consiste à utiliser l'ensemble L_G de toutes les graines, c'est-à-dire que, par construction, il contient la (ou les) graine(s) conduisant à la SSCM entre deux molécules [16].

La seconde stratégie consiste à ne sélectionner qu'un sous-ensemble de L_G ; le problème est alors de trouver le sous-ensemble minimal contenant la (ou les) graine(s) à l'origine de la SSCM entre deux molécules. Deux modes de sélection de graines sont testées pour déterminer si l'un d'eux répond à la problématique posée :

- *Sous-ensemble aléatoire de graines* : g graines sont sélectionnées dans la liste L_G. Dans les tests effectués, les valeurs de 25%, 50% et 75% de la taille de L_G ont été considérées ;
- *Filtrage des graines* : cette méthode de sélection des graines consiste à construire l'ensemble L_G des graines possibles puis à supprimer de cette liste, les graines qui ont été appariées dans des appariements précédents. Contrairement à la précédente méthode, le nombre de graines qui sera utilisé lors de l'exécution de l'algorithme n'est pas connu à l'avance.

La stratégie qui considère l'ensemble des graines de L_G sert de ligne de base pour les expériences. Dans le cas des stratégies basées sur un sous-ensemble de L_G, on peut

16. C'est cette stratégie qui a été utilisée dans les tests précédents.

supposer que les résultats de classification (valeurs de IC) sont au mieux identiques ou sinon moins bons que ceux obtenus avec la variante utilisant la totalité de l'ensemble L_G. En effet, une exploration partielle ne peut être que moins bonne ou égale à une exploration complète. C'est donc ce que nous voulons vérifier ici.

6.2.4.2 Appariement stochastique entre les voisins

Nous testons ici les 3 stratégies de sélection de graines avec l'algorithme stochastique ; la variante APC_Stochast_SansArret, qui utilise toutes les graines de L_G, sert de référence dans ce test. Pour la stratégie qui consiste à sélectionner g graines dans L_G, comme la variante APC_Stochast_* n'utilise pas la matrice INC, les graines sont choisies aléatoirement. Les variantes suivantes sont testées : APC_Stochast_25 qui sélectionne 25% des graines de L_G, APC_Stochast_50 qui en sélectionne 50% et APC_Stochast_75 qui en sélectionne 75%. Pour chacune de ces trois variantes, le processus de classification complet est répété 10 fois de façon à calculer la moyenne et l'écart-type des valeurs de l'indice IC. Enfin, la variante APC_Stochast_Filtre implémente la stratégie de filtrage des graines lors de l'exploration des graphes. Pour cette variante, une seule classification a été réalisée. En effet, une paire de molécules donne toujours la même matrice INC et les parcoures de graphes étant déterministes, les mêmes appariements sont identifiés d'un test à l'autre.

Pour faciliter la lecture et l'analyse des résultats, nous présentons, dans le tableau 6.4, les valeurs de l'indice IC obtenues pour le nombre de classes initial.

Variantes stochastiques de I_π	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
APC_Stochast_SansArret	0,34	0,31	0,49	0,04
APC_Stochast_75	0.32 (± 0.03)	0.24 (± 0.00)	0.45 (± 0.03)	0.21 (± 0.00)
APC_Stochast_50	0.35 (± 0.03)	0.23 (± 0.01)	0.59 (± 0.05)	0.21 (± 0.00)
APC_Stochast_25	0.54 (± 0.04)	0.25 (± 0.02)	1.08 (± 0.06)	0.22 (± 0.02)
APC_Stochast_Filtre	0.34 (26.9%)	0.24 (28.6%)	0.43 (32.2%)	0.21 (39.8%)

Tableau 6.4 – Moyenne de l'indice IC (sur 10 répétitions) pour les variantes stochastiques de la méthode I_π utilisant des échantillonnages de graines différents. La variante APC_Stochast_SansArret, qui sert de référence (cette version utilise 100% des graines), est signalée en gris. Pour la variante APC_Stochast_Filtre, les valeurs entre parenthèses correspondent au pourcentage de graines utilisées comme point de départ des appariements ; les graines ayant un degré d'inclusion nul ne sont pas testées.

Pour les bases BZR, DHFR et ER, les variantes qui utilisent une fraction de l'ensemble L_G sont moins performantes au fur et à mesure que la taille du sous-ensemble de graines diminue : de 75% à 25% de graines, les valeurs de l'indice IC augmentent ce qui semble cohérent dans la mesure où le nombre de graines est plus petit. Dans ce cas, l'algorithme a moins de chance de découvrir la SSCM. Sur les bases COX2 et ER, le

comportement est différent. Sur COX2, l'utilisation du sous-ensemble contenant 50% des graines de L_G donne en moyenne de meilleurs résultats que celui de 75% de graines. De plus, contrairement à ce que l'on pourrait logiquement penser, la variante de référence, qui utilise la totalité des graines possibles, ne donne pas toujours les meilleurs résultats. Elle est plus mauvaise pour les bases BZR et DHFR que la variante avec 75% des graines ; sur COX2, elle est plus mauvaise que les trois variantes utilisant respectivement 75%, 50% et 25% de graines de L_G.

Pour expliquer le phénomène, considérons l'exemple de la base COX2. Vu les valeurs d'indice IC, on sait que la SSC identifiée avec 75% des graines (*i.e.* celle qui a la plus forte valeur de I_π) a permis de mieux classer les molécules que la variante de référence. Ce qui signifie que la valeur du degré d'inclusion associé à la SSC identifié avec APC_Stochast_75 est inférieure au degré d'inclusion correspondant à la SSC trouvée avec APC_Stochast_SansArret (elle ne peut pas être égale sinon, les deux classifications auraient été identiques). Si tel est le cas, alors une bonne classification n'est pas forcément basée sur la SSC qui a la plus forte valeur de I_π (la SSCM, d'après l'hypothèse que nous avons faite). Cette explication est cohérente avec les résultats sur les bases BZR, COX2 et DHFR et le fait que la SSCM entre deux molécules est différente du châssis moléculaire.

Toutefois, pour valider cette hypothèse, il faudrait éliminer le biais statistique en répétant les tests un plus grand nombre de fois et en intégrant les variantes APC_Stochast_SansArret et APC_Stochast_Filtre pour vérifier si les valeurs obtenues avec x% des graines se rapprochent de celles de la variante APC_Stochast_SansArret.

On observe que la variante APC_Stochast_Filtre réduit effectivement de façon importante le nombre de graines testées tout en donnant de meilleurs résultats que la variante APC_Stochast_SansArret sur les bases BZR, COX2 et DHFR. On peut remarquer notamment que pour la base BZR, la variante APC_Stochast_Filtre qui utilise 26.9% des graines est bien meilleure que la variante APC_Stochast_25 qui n'en utilise que 25% ; ce qui tend à montrer que les graines utilisées dans la variante filtrée sont de meilleure "qualité" que celles des autres stratégies de sélection des graines. Cependant, ce résultat est à relativiser car l'utilisation de la variante "SansArrêt" a tendance à générer de plus grandes SSC, ce qui limite naturellement le nombre de paires d'atomes pouvant servir de graines.

6.2.4.3 Appariement optimal entre les voisins

Dans cette expérience, les appariements locaux, entre les atomes voisins sont réalisés par l'algorithme hongrois ; la variante APC_Kuhn_SansArret servant de référence. Comme cet algorithme d'appariement utilise la matrice INC, on va s'en servir pour trier les graines de L_G par ordre des degrés d'inclusion décroissants. Ainsi, pour la

stratégie qui sélectionne g graines au départ, on assure de choisir les graines qui ont la plus forte valeur dans la matrice INC (alors que dans la variante stochastique, ces graines étaient choisies aléatoirement). La variante APC_Kuhn_75 sélectionne les 75% de graines ayant les plus fortes valeurs dans I_π, la variante APC_Kuhn_50 utilise les 50% meilleures graines et APC_Kuhn_25 seulement les 25% meilleures. La variante APC_Kuhn_Filtre correspond à la stratégie de filtrage des graines lors de l'exploration des graphes.

Le tableau 6.5 ci-dessous présente les valeurs numériques de l'indice IC pour ces variantes. Les courbes correspondant aux indices IC sont affichées dans l'annexe B.

Variantes de I_π	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
APC_Kuhn_SansArret (100%)	0.35	0.35	1.03	0.21
APC_Kuhn_75	0.32	0.31	1.03	0.21
APC_Kuhn_50	0.27	0.25	1.03	0.21
APC_Kuhn_25	0.28	0.25	0.99	0.00
APC_Kuhn_Filtre	0.32 (27.8%)	0.31 (26.4%)	1.02 (32.5%)	0.21 (41.9%)

TABLEAU 6.5 – Valeur de l'indice IC pour les variantes de la méthode I_π utilisant l'algorithme hongrois avec des échantillonnages de graines différents. La variante APC_Kuhn_SansArret, qui sert de référence, est signalée en gris. Pour la variante variante APC_Kuhn_Filtre, les valeurs entre parenthèses correspondent au pourcentage de graines utilisées comme point de départ des appariements.

Le comportement de ces variantes est similaire à celles des variantes stochastiques : la diminution du nombre de graines prises en compte n'entraîne pas systématiquement une dégradation des résultats. Sur la base BZR, la variante APC_Kuhn_50 est plus performante que la variante APC_Kuhn_25 ; sur les bases COX2, DHFR et ER, les résultats sont meilleurs avec de petits sous-ensembles de graines. Comme les sous-ensembles de graines contiennent ici les g graines qui ont les plus grandes valeurs de I_π, les bons résultats des variantes avec ces sous-ensembles sont effectivement dus aux SSC issues des graines à forte valeur dans la matrice INC.

Par contre, cela signifie également que lorsque l'on sélectionne 25% des graines par exemple, il existe au moins un appariement dont le degré d'inclusion est plus grand que le maximum identifié avec les 75% premières graines, ce qui laisse à penser que les graines à forte valeur de I_π ne génèrent pas systématiquement les plus grandes SSC.

6.2.4.4 Correspondance entre degré d'inclusion de la graine et valeur d'inclusion globale

D'après les résultats de la section précédente, l'hypothèse que nous avions avancée stipulant que c'est la plus grande SSC qui possède la plus forte valeur de I_π est sans doute erronée. Pour le vérifier, nous observons expérimentalement la correspondance entre le degré d'inclusion d'une graine et la valeur de l'indice I_π de la SSC générée à partir de cette graine. L'idée générale est de regarder, pour chaque paire de molécules dans les bases, les couples <degré d'inclusion (graine), I_π(SSCM) associée>. Cependant, dans ce cas, on comparerait un grand nombre de molécules différentes ce qui conduirait à identifier de nombreuses petites SSCM (et donc à de faibles valeurs de I_π). Pour limiter ce cas de figure, nous nous intéressons uniquement aux paires de molécules similaires en sélectionnant, pour chaque molécule dans une base, ses k plus proches voisins (avec $k=3$). Les résultats obtenus avec la variante APC_Kuhn_SansArret et en utilisant l'ensemble des graines possibles sont affichés sur les graphiques à gauche sur la figure 6.15.

Quelles que soient les bases, on peut remarquer que les graines dont la valeur de I_π est proche de 1 ne génèrent pas uniquement de grandes SSCM (certaines SSC associées ont même un indice I_π est très faible), ce qui confirme les suppositions avancées précédemment. On observe également que des graines avec un certain degré d'inclusion génèrent des SSC de toute taille (valeurs de I_π comprises entre 0 et 1) ; ce phénomène est particulièrement fréquent sur la base ER.

Sur les quatre bases, on note la présence de nombreuses graines dont le degré d'inclusion est élevé (valeur supérieure à 0.9) et qui génèrent des SSC dont la valeur de I_π est proche de 0 (ce sont de petites sous-structures). Ces graines correspondent à de grandes ressemblances locales au niveau des atomes mais pas à une plus grande échelle lorsque l'algorithme intègre le voisinage étendu de ces graines : les différences sont donc repérées assez rapidement.

Sur les résultats affichés à droite sur la figure 6.15, on observe la correspondance entre le degré d'inclusion des graines et la valeur de l'indice I_π dans le cas où l'ensemble des graines est filtré (variante APC_Kuhn_Filtre). On observe une différence très nette avec le fait d'utiliser l'ensemble de graines. Pour les quatre bases, la quasi totalité des graines a un degré d'inclusion égale à 1 et la plupart des SSCM générées à partir de ces graines a une valeur de I_π supérieure à 0.6. Cette stratégie permet donc de bien filtrer les graines inutiles (c'est-à-dire celles qui identifient des zones non similaires dans les molécules et qui conduisent à de petites SSC) tout en identifiant correctement de grandes SSC. Elle semble performante car, sur les quatre chimiothèques testées, et dans les expériences réalisées ici, elle donne souvent de meilleurs résultats que l'utilisation de l'ensemble des graines possibles.

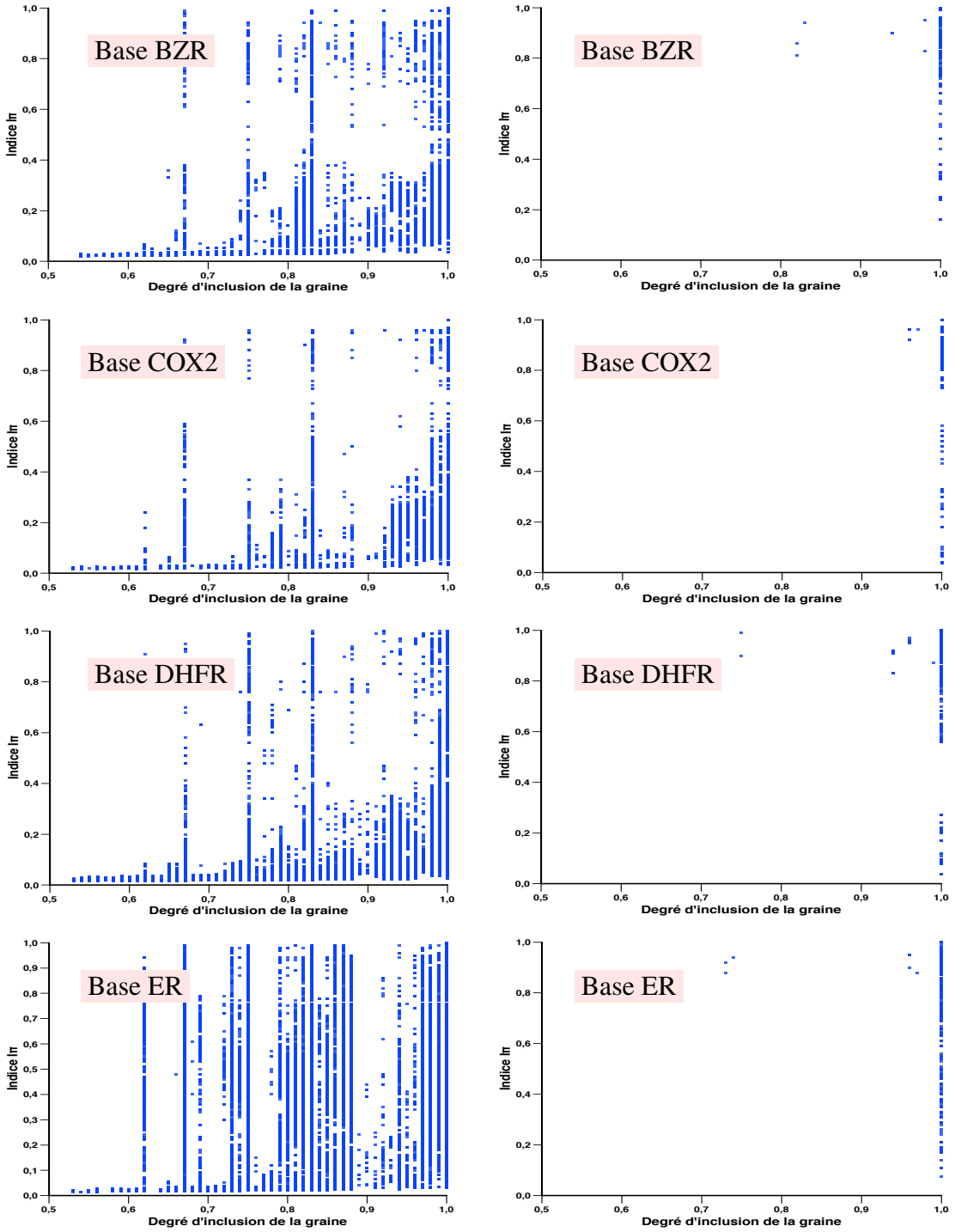


FIGURE 6.15 – Correspondance entre le degré d'inclusion d'une graine et l'indice I_π de la SSC générée à partir de cette graine. Les graphiques à gauche correspondent au cas où l'on tient compte de l'ensemble des graines possibles (variante APC_Kuhn_SansArret). Les graphiques à droite correspondent à la variante APC_Kuhn_Filtre où l'ensemble des graines est filtré.

6.2.5 Synthèse

Ces premiers tests sur les différentes possibilités de la méthode I_π ont permis de mieux comprendre l'influence de ces diverses stratégies. Vu les différences observées entre les chimiothèques, il est difficile d'identifier un paramétrage unique qui soit performant quelle que soit la chimiothèque utilisée.

Les variantes stochastiques, qui n'utilisent pas la matrice INC pour guider l'appariement local mais uniquement la structure du graphe moléculaire, sont compétitives par rapport aux méthodes guidées. En effet, les appariements entre voisins qui nécessitent un algorithme d'optimisation sont peu fréquents et les cas d'appariements où des erreurs sont possibles sont finalement très limités. Bien que les molécules soient représentées par une description succincte (structure 2D avec uniquement le nom des atomes et la valence des liaisons), ces résultats s'expliquent par le fait que les atomes et les liaisons sont typés, le problème d'appariement des atomes entre deux molécules est déjà suffisamment contraint et l'algorithme de recherche d'un APC ajoute encore un très grand nombre de contraintes. Les variantes stochastiques ne commettent donc pas autant d'erreurs qu'on aurait pu le penser.

A l'inverse, les choix d'appariements locaux optimaux ne sont pas toujours performants. La raison est peut-être dans le fait qu'ils se basent sur les valeurs de la matrice INC qui sont sur-évaluées. Il serait intéressant de mettre en place la mise à jour de cette matrice et de refaire les tests pour obtenir une éventuelle amélioration des résultats.

Pour trouver la plus grande sous-structure commune, les variantes APC_*_SansArret sont mieux adaptées car l'exploration des graphes moléculaires est poursuivie plus loin qu'avec les variantes APC_*_AvecArret. Suivant la stratégie utilisée, l'algorithme identifie des sous-structures qui sont des sur-ensembles (ou des sous-ensembles) du châssis moléculaire. Cependant, l'idée de baser la similarité entre deux molécules sur la plus grande sous-structure commune n'est pas systématiquement optimale du point de vue de la tâche de classification. Par exemple dans certaines bases comme BZR, l'utilisation des variantes "APC_*_SansArret " donne de meilleurs résultats que les variantes APC_*_AvecArret et dans d'autres cas (COX2 et DHFR), c'est l'inverse qui se produit.

Une approche alternative pourrait s'inspirer de [Stahl and Mauser, 2005] qui utilise un ensemble de plusieurs SSC entre deux molécules au lieu de ne considérer que la plus grande, ce qui revient revient à identifier des SSC non connexes. Une SSC entre deux molécules peut être disjointe, c'est-à-dire qu'elle est représentée par un graphe non connexe. Dans les cas où la ressemblance entre les molécules tient à plusieurs SSC au lieu d'un châssis moléculaire, il est effectivement intéressant de se baser sur

ces SSC disjointes pour évaluer la similarité entre les molécules. L'algorithme I_π permet naturellement de générer plusieurs SSC entre deux molécules, ces sous-structures étant éventuellement recouvrantes. Pour cela, il suffit de tenir compte des sous-graphes générés à partir de graines différentes lors de la phase d'appariement global. Tout le problème est de savoir comment tenir compte de ces informations. En effet, la gestion des parties recouvrantes n'est pas triviale : plus les sous-structures extraites se recoupent, moins l'ensemble est informatif. Il convient également de savoir quelles sous-structures conserver pour obtenir un pavage optimal des molécules.

6.3 Comparaison avec les noyaux de graphes

Dans cette partie, nous allons comparer la mesure de similarité Sim_{I_π} avec les mesures de similarité présentées dans le chapitre précédent. Pour les mesures/algorithmes qui nécessitent d'ajuster des paramètres, nous avons mené, sur chaque base, des tests de façon à identifier les valeurs des paramètres qui conduisent à la meilleure classification (*i.e.* les plus faibles valeurs de IC et IS) pour un algorithme donné. Ce sont ces valeurs que nous avons utilisées pour la comparaison des mesures de similarité entre elles.

Pour l'algorithme I_π, nous avons retenu les variantes qui donnaient les meilleurs résultats dans l'expérience de la section 6.2 : APC_Stochast_AvecArret pour la base BZR et APC_Glouton_AvecArret pour les trois autres bases en considérant l'ensemble des graines possibles. Pour les deux paramètres (l et u) du noyau de Tanimoto et du noyau spectral, nous avons exploré toutes les valeurs entières dans l'intervalle $[1, 20]$; le détail de ces résultats est présenté dans l'annexe A. Les valeurs retenues sont :

- base BZR : $u = 10$ pour le noyau spectral et le noyau de Tanimoto ;
- base COX2 : $l = 10$ pour le noyau spectral et $u = 10$ pour le noyau de Tanimoto ;
- base DHFR : $l = 5$ pour le noyau spectral et $u = 10$ pour le noyau de Tanimoto ;
- base ER : $l = 5$ pour le noyau spectral et $l = 20$ pour le noyau de Tanimoto.

Pour le noyau ISOA et le noyau AO, les paramètres par défaut proposés par les auteurs ont été utilisés car ils sont assez proches de notre méthode (voir section 6.1.2). Pour l'outil Ward de Chemaxon, une série d'expérimentations a également été réalisée pour déterminer les empreintes moléculaires optimales (voir section 6.1.2). La longueur des vecteurs est de 512 bits. Les valeurs des paramètres l (longueur maximale des motifs) et b (nombre maximal de bits pour chaque motif) sont les suivantes :

- base BZR : $l = 3$ et $b = 6$;
- base COX2 : $l = 2$ et $b = 8$;
- base DHFR : $l = 3$ et $b = 6$;
- base ER : $l = 7$ et $b = 2$.

Les valeurs de l'indice IC sont présentées sur les courbes des figures 6.16 et 6.17. Sur l'ensemble des quatre bases, la hiérarchie obtenue entre les distances par rapport à l'indice IS est exactement la même que pour l'indice IC. Les meilleures méthodes étant aussi celles qui ont le moins tendance à segmenter. Nous constatons une nouvelle fois une nette différence de comportement entre les mesures de similarité : sur BZR et ER, les valeurs des indices sont regroupées sur une petite plage de valeurs alors que les écarts sont nettement plus importants pour COX2 et DHFR. Ceci est dû à la composition des bases elles-mêmes : l'effectif et la diversité des familles qui sont différentes d'une base à l'autre.

Les valeurs numériques des indices IC pour les différentes méthodes pour un nombre de classes correspondant aux classes définies par les chimistes sont données dans le tableau 6.6.

Algorithme	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
I_π	**0.34**	0.13	**0.46**	0.04
noyau de Décomposition Pondérée (DP)	0.73	0.52	1.98	0.18
noyau spectral	0.40	**0.08**	0.56	0.21
noyau de Tanimoto	0.47	0.25	0.56	**0.00**
noyau AO	0.54	0.82	1.64	0.10
noyau ISOA	0.37	0.16	1.39	**0.00**
chemaxonWard	0.57	0.80	1.58	0.21

Tableau 6.6 – Valeur de l'indice IC au point du nombre de classes attendues pour différentes mesures de similarité.

Le noyau spectral, le noyau de Tanimoto et le noyau ISOA donnent des résultats moyens tandis que la mesure Sim_{I_π} est généralement bien placée. L'avantage de I_π est net pour DHFR mais plus faible pour BZR. Pour les quatre bases, le noyau DP et le noyau AO ont systématiquement des valeurs d'indice très nettement supérieures aux autres méthodes. Ces deux noyaux sont moins performants que la méthode I_π sauf sur la base ER où le noyau ISOA est meilleur. Notre hypothèse est que ce sont tous deux des algorithmes qui ne tiennent pas totalement compte de la topologie des graphes moléculaires (dans la phase d'appariement des atomes pour le noyau AO et dans le noyau entre les environnements des sélecteurs pour le noyau DP) contrairement à la recherche d'APC dans la méthode I_π. Cependant, ceci n'explique pas les bonnes performances de noyau ISOA sur BZR, COX2 et ER alors qu'il est très similaire au noyau AO.

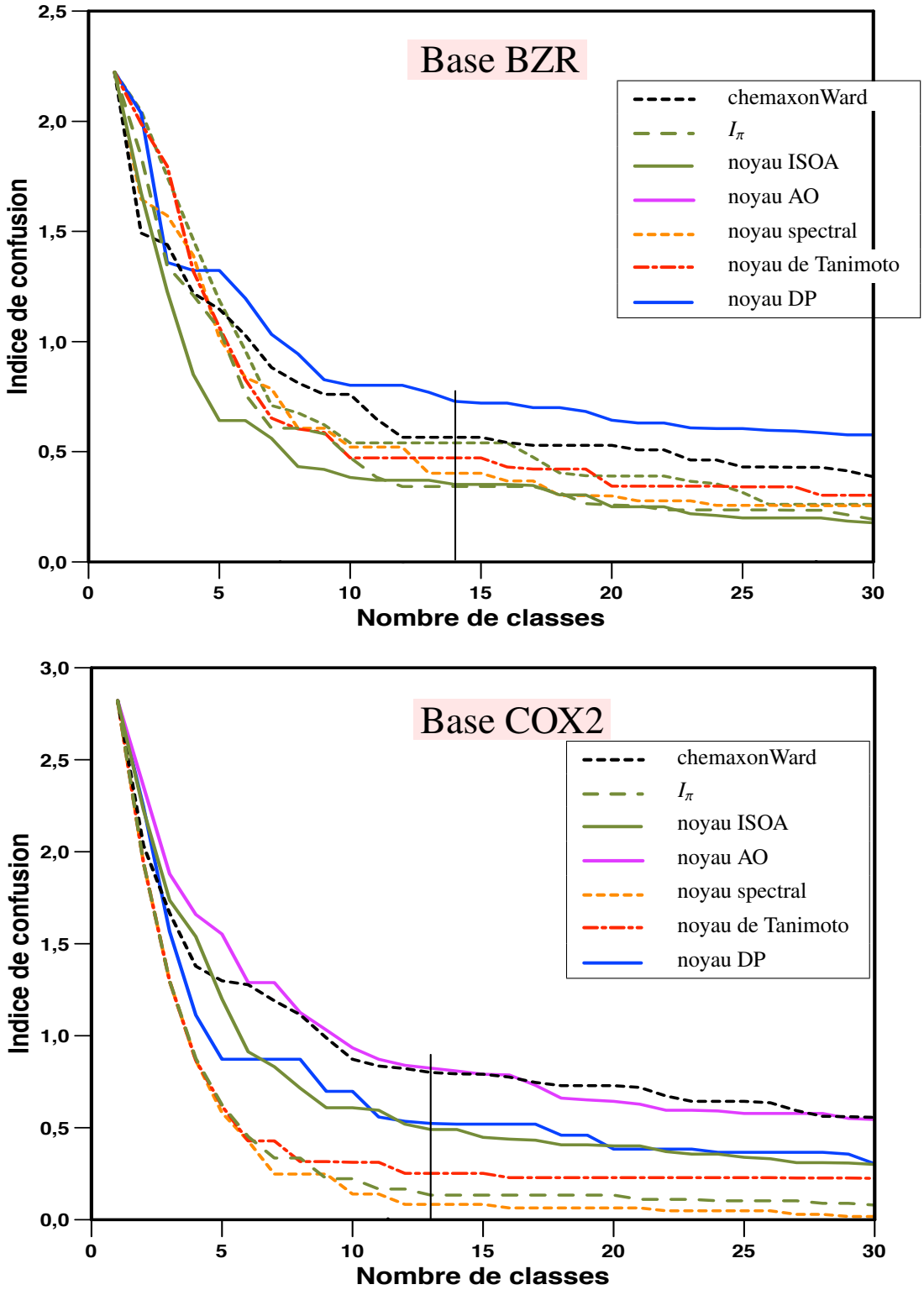
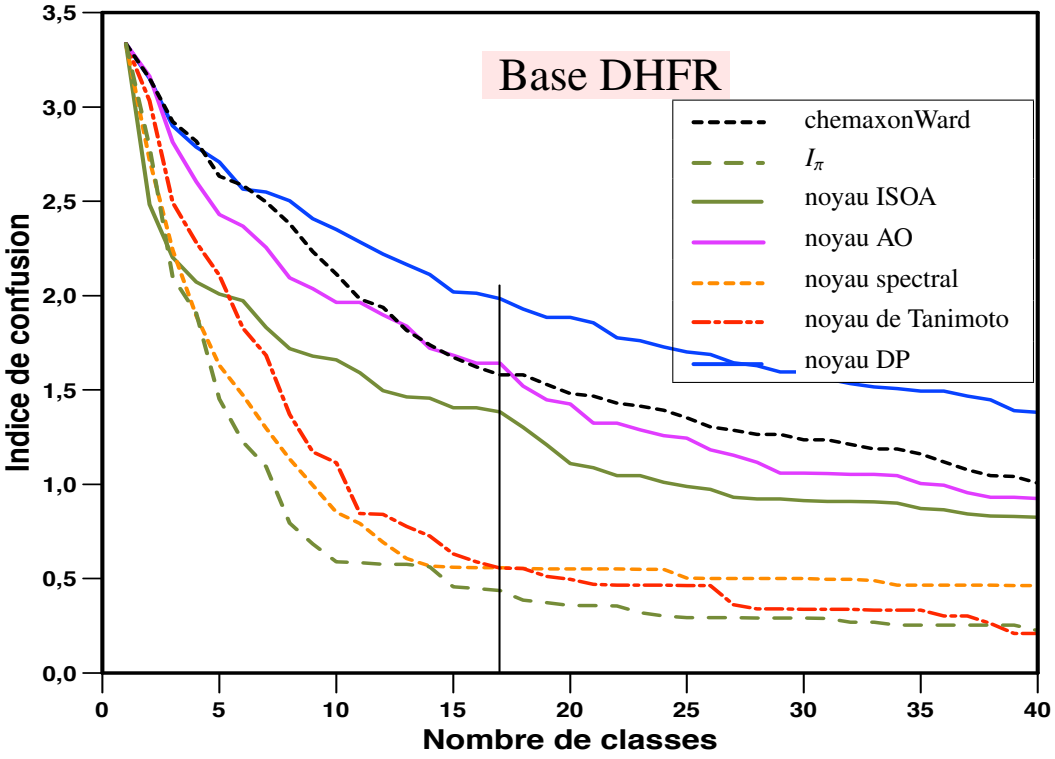


FIGURE 6.16 – Indices IC pour les mesures de similarité utilisées avec la CAH sur les chimiothèques. Le trait vertical représente le nombre de familles chimiques original.



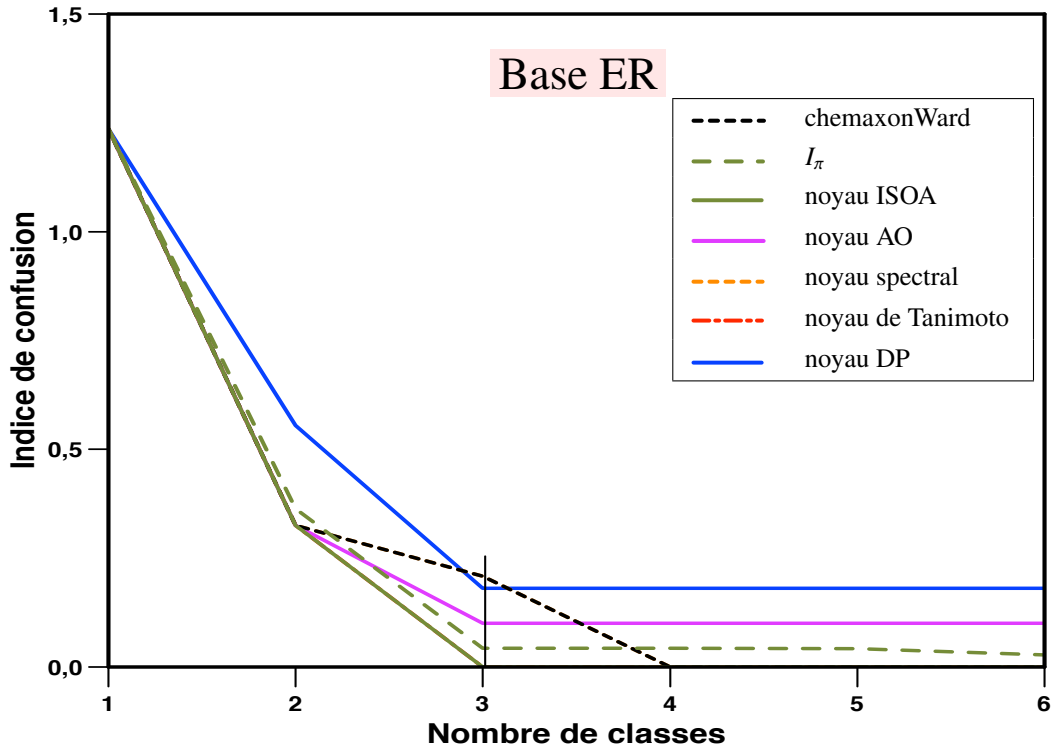


FIGURE 6.17 – Indices IC pour les mesures de similarité utilisées avec la CAH sur les chimiothèques. Le trait vertical représente le nombre de familles chimiques original. Sur la base ER, le noyau de Tanimoto et le noyau ISOA donnent les mêmes résultats : les courbes sont superposées.

Le noyau de Tanimoto et le noyau spectral ne tiennent pas non plus totalement compte de la topologie (dans le sens où ils se basent sur des chemins dans les graphes et non sur des sous-structures) mais l'ensemble des chemins considérés est de taille importante ce qui permet un échantillonnage suffisamment satisfaisant de la structure des graphes moléculaires pour capturer l'information structurale globale. Ce qui ne semble pas être le cas pour chemaxonWard qui a des performances très mauvaises sur les bases BZR, COX2 et DHFR. Par contre, il retrouve la bonne classification sur la base ER à partir de $h=4$.

Lorsque l'on examine le contenu de BZR et COX2, on constate qu'elles sont très homogènes et que les molécules se ressemblent toutes beaucoup. Les différentes familles ne diffèrent que par de petites modifications de leur châssis. Les résultats expérimentaux peuvent donc surprendre car on aurait pu penser que la tâche de classification était plus difficile dans un tel cas. L'explication pourrait être la suivante : il est plus facile pour les mesures de similarité « d'aligner » les parties communes des molécules car elles sont très similaires (Figure 6.18/haut) et de faire ainsi ressortir les nuances critiques entre les familles. Dans le cas de DHFR, qui est plus diverse, les appariements entre molécules sont moins faciles à déterminer (figure 6.18/bas) ce qui entraîne une confusion entre certaines familles. Dans ce dernier cas, on constate toutefois que la mesure « structurelle » I_π est plus apte à dégager les structures pertinentes que le noyau de Tanimoto ou chemaxonWard.

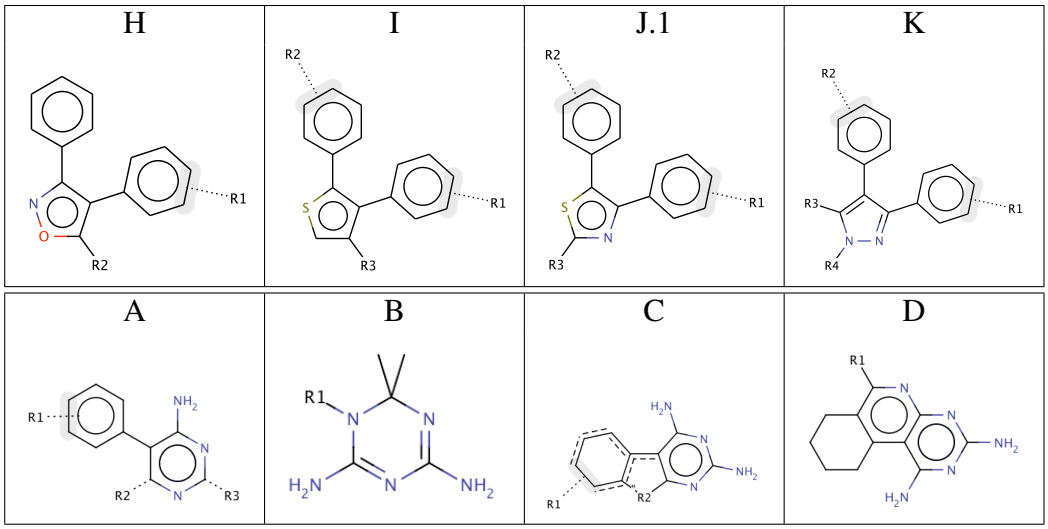


Figure 6.18 – Pour chacune des bases COX2 (haut) et DHFR (bas), on a ici un exemple de quatre familles décrites par leur châssis moléculaire. Les liaisons notées R_n indique que des « radicaux » (c'est-à-dire des fragments de molécules) variables peuvent se rattacher à ces emplacements. Comme on le constate, les familles de COX2 sont structurellement beaucoup plus similaires entre elles que dans le cas de DHFR.

Pour la base ER, on remarque que certaines mesures (le noyau ISOA et le noyau de Tanimoto) trouvent la bonne classification dès $h = 3$ classes et d'autres à $h = 4$ classes (chemaxonWard). Par contre, l'indice *IC* des autres mesures (le noyau spectral, le noyau AO et le noyau DP) reste constant, ce qui signifie que des erreurs de classification persistent.

Sur l'ensemble des chimiothèques, la méthode I_π est celle qui donne le plus souvent les meilleurs résultats. D'après les études précédentes, nous expliquons cela par le fait qu'elle recherche une SSC pour comparer les molécules contrairement au noyau AO et noyau ISOA qui ne tiennent pas explicitement compte de la structure des graphes lors de l'appariement global. Le noyau DP est similaire aux deux précédents dans la mesure où l'information structurelle n'est utilisée que pour mesurer la similarité entre chaque paire d'atomes ce qui n'est, de toute évidence, pas suffisant. A l'inverse, l'utilisation de l'APC, lors de la phase d'appariement dans I_π, permet de conserver une information structurelle suffisante.

La recherche d'un APC commun entre deux sous-structures combinée avec la notion d'inclusion n'est pas suffisante pour expliquer les bons résultats de la méthode I_π. Le noyau de Tanimoto donne des résultats proches de I_π et représente les molécules différemment. Malgré la simplicité de l'approche, on peut être surpris des bons résultats obtenus par le noyau de Tanimoto, qui est clairement moins complexe à calculer que l'indice I_π. Cependant, ce dernier présente l'avantage de prendre en compte plus facilement la structure des molécules et il n'est pas nécessaire de choisir manuellement la taille des clés structurelles à utiliser. Nous pouvons penser que l'échantillonnage de séquences dans les composés est suffisamment grand pour extraire l'information essentielle des molécules.

6.4 Etude comparative avec les noyaux AO et ISOA

Nous allons maintenant comparer la méthode I_π avec les noyaux AO de Froëlich et ISOA de Rupp. Cette étude est intéressante dans la mesure où ces trois méthodes sont assez semblables : elles sont basées d'une part sur une exploration locale de la correspondance entre les paires d'atomes qui composent deux molécules et d'autre part, sur la recherche du meilleur appariement global des atomes. Cela permet d'identifier quelles sont les stratégies (ou combinaisons de stratégies) implémentées dans I_π qui donnent les meilleurs résultats et de comprendre les raisons de ces performances.

Voici un rappel des différences entre ces méthodes. La première différence est que les noyaux AO et ISOA évaluent directement une mesure de similarité entre les molécules tandis que I_π utilise le degré d'inclusion qui est une mesure asymétrique. La deuxième différence se situe au niveau du calcul de la matrice de ressemblance entre

les atomes (matrice K pour les noyaux AO et ISOA et matrice INC pour I_π) où la fonction Di_{lit}, qui évalue la ressemblance entre deux liaisons du point de vue de deux atomes donnés, est définie par :

- la moyenne de ces deux termes dans l'algorithme I_π (équation 3.3, chapitre 3) ;
- le produit de la similarité des atomes voisins et la similarité des liaisons pour les noyaux AO (équation 5.13, chapitre 5) et ISOA (équation 5.18, chapitre 5).

Dans la phase d'appariement global entre les atomes, l'algorithme hongrois est utilisé dans les noyaux AO et ISOA tandis que l'algorithme I_π se base sur un parcours de graphes. Dans le noyau AO, la similarité du voisinage de deux atomes peut être précalculée avant d'être utilisée dans le calcul de la similarité entre ces deux atomes. Au contraire, dans le noyau ISOA et l'algorithme I_π, du fait de la définition récursive de la méthode, la ressemblance entre les voisinages est mise à jour à chaque itération. Les informations de similarité sont alors propagées le long des liaisons dans les molécules. Le tableau 6.7 résume les différences entre ces méthodes.

Méthode	fonction Di_{lit}	calcul de la matrice	Appariement des atomes	Type de mesure
I_π	moyenne	récursif	recherche d'APC	moyenne de 2 degrés d'inclusion
ISOA	produit	statique	Algorithme hongrois	similarité
AO	produit	récursif	Algorithme hongrois	similarité

TABLEAU 6.7 – Différences entre l'algorithme I_π et les fonctions noyau AO et ISOA.

6.4.1 Comparaison des variantes de I_π avec le noyau AO

Dans cette étude, nous comparons le noyau AO avec l'algorithme I_π dont nous modifions les deux paramètres suivants de façon à se rapprocher le plus possible du noyau AO :

- Mesure ressemblance entre les atomes/molécules, choix entre : le calcul d'une similarité ou d'un degré d'inclusion ;
- Appariement global : choix entre la recherche d'un APC et l'algorithme hongrois. Dans le cas de la recherche d'APC, l'exploration des graphes continue tant que les atomes peuvent être appariés (variante "sans_arrêt") de manière à se placer dans des conditions similaires à AO, même si, comme nous l'avons vu, ces conditions ne sont pas toujours optimales pour la chimie.

Les quatre variantes de I_π sont les suivantes.

I_π**_Inc_APC** L'appariement local entre les voisins lors de l'exploration des graphes est réalisé par l'algorithme hongrois et la mesure de similarité est la moyenne des degrés d'inclusion entre les deux molécules comparées. C'est la version "originale" de l'algorithme I_π.

I_π**_Sim_APC** Comme pour la variante précédente, l'appariement local entre les voisins est réalisé par l'algorithme hongrois. Par contre, la mesure de similarité entre les molécules est calculée directement sans utiliser les degrés d'inclusion. Pour cela, les facteurs de normalisation dans l'algorithme I_π ont été modifiés de façon à tenir compte des deux composés (voir les sections 2.4 page 30 et 3.1 page 38). On obtient ainsi une mesure de similarité symétrique.

I_π**_Sim_Kuhn** La mesure de similarité est calculée directement comme dans la version I_π_Sim_APC et l'appariement global entre les atomes des molécules est réalisé par l'algorithme hongrois.

I_π**_Inc_Kuhn** La mesure de similarité est la moyenne des degrés d'inclusion entre les deux molécules ; l'appariement global est réalisé par l'algorithme hongrois.

Les résultats (indice IC) des tests sur les quatre bases sont présentés sur les figures 6.19 et 6.20.

Le tableau 6.8 indique les valeurs de l'indice IC pour le nombre de classes attendues sur les quatre chimiothèques.

Algorithme I_π	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
I_π_Inc_Kuhn	0.60	1.16	1.93	0.33
I_π_Sim_Kuhn	0.64	1.45	2.23	0.33
I_π_Sim_APC	**0.30**	**0.26**	1.18	0.21
I_π_Inc_APC	0.35	0.35	**1.03**	0.21
noyau AO	0.54	0.82	1.64	**0.10**

Tableau 6.8 – Résultats numériques (indice IC) des variantes de l'algorithme I_π utilisées dans la comparaison avec le noyau AO au point de coupure du nombre de classes attendues.

Tout comme il a été observé dans l'expérimentation précédente, les valeurs de l'indice IC des méthodes testées sont plus rapprochées sur BZR et sur ER (où elles se superposent parfois) que sur COX2 ou DHFR. Quelles que soient les chimiothèques, les

variantes qui utilisent l'algorithme hongrois pour apparier les atomes (I_π_Inc_Kuhn et I_π_Sim_Kuhn) sont toujours moins performantes que les variantes recherchant un APC (I_π_Sim_APC et I_π_Inc_APC). Dans le cas des variantes asymétriques, cela confirme les résultats du chapitre 3 qui indiquait que les valeurs de la matrice INC sont sur-estimées et qu'elles ne contiennent pas les informations structurelles suffisantes pour apparier correctement les atomes. La recherche d'APC, qui explore effectivement la structure des graphes, complète efficacement l'utilisation de la matrice INC en tirant parti directement de la topologie des graphes.

Sur les quatre bases, on remarque que, pour les variantes Ipi_*_Kuhn qui utilisent l'algorithme hongrois pour l'appariement global, l'utilisation d'un double degré d'inclusion donne de meilleurs résultats que la similarité. Ce qui semble confirmer que, comme nous l'avions supposé, l'utilisation de la mesure basée sur un double degré d'inclusion est moins sensible à la taille et à la complexité des molécules qu'une mesure symétrique. Dans le cas des variantes utilisant la recherche d'APC, l'utilisation du double degré d'inclusion par rapport à la similarité ne conduit à de meilleurs résultats que sur la base DHFR et à des résultats équivalents sur ER : pour les bases BZR et COX2, c'est la variante I_π_Sim_APC qui obtient les plus faibles valeurs de IC.

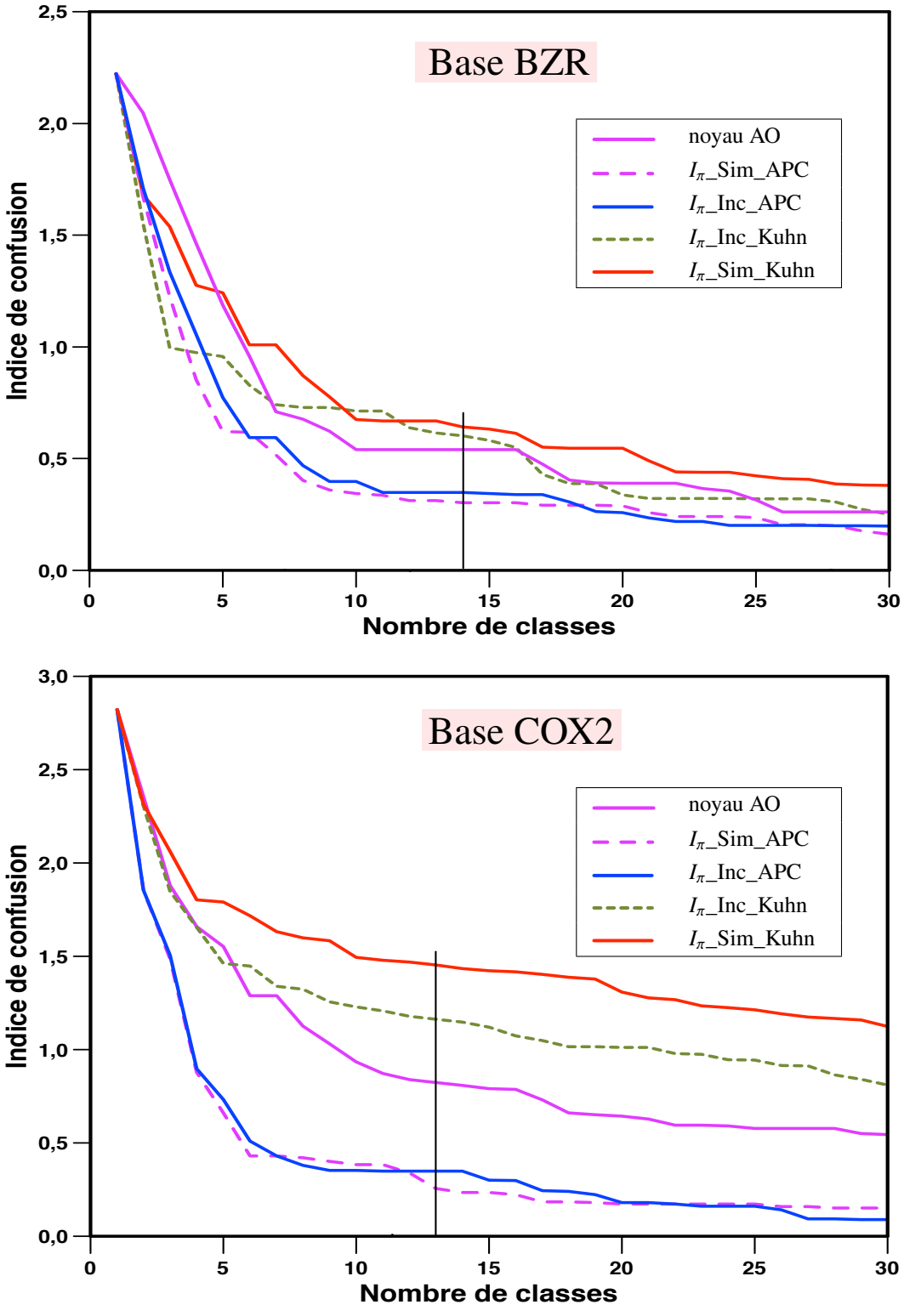


FIGURE 6.19 – Comparaison de l'algorithme I_π et du noyau AO pour les bases BZR et COX2.

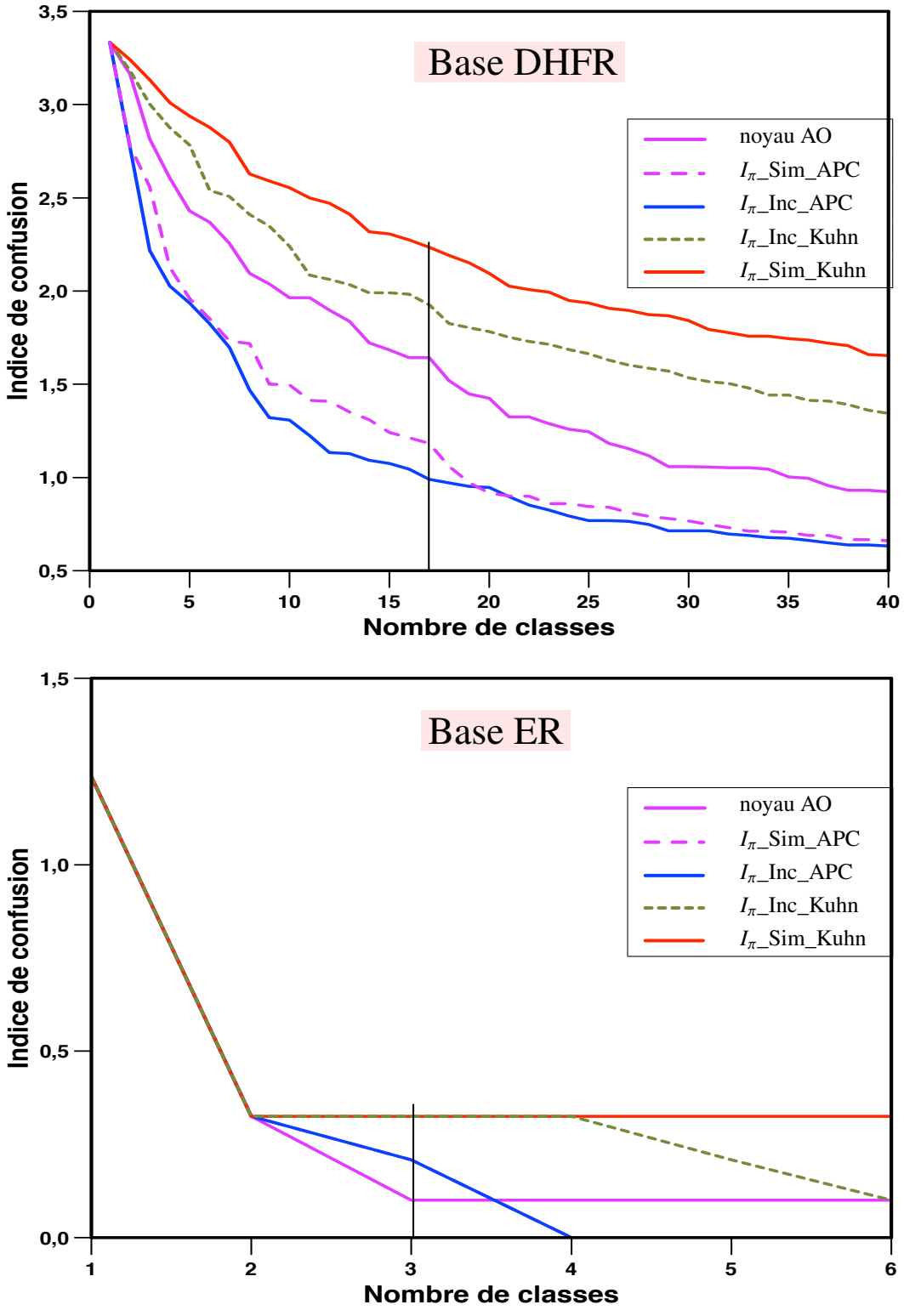


FIGURE 6.20 – Comparaison de l'algorithme I_π et du noyau AO pour les bases DHFR et ER. Pour ER, les courbes correspondant aux variantes I_π_Inc_APC et I_π_Sim_APC sont superposées.

Sur les quatre bases, le noyau AO est moins performant que les variantes Ipi_*_APC (I_π_Sim_APC et I_π_Inc_APC) et meilleur que les variantes Ipi_*_Kuhn (I_π_Inc_Kuhn et I_π_Sim_Kuhn). Les différences étant plus marquées sur les bases COX2 et DHFR. L'hypothèse que l'on peut avancer pour expliquer ces résultats est la manière dont la similarité entre les voisinages est évaluée : statique dans le cas du noyau AO et dynamique pour I_π. Considérons l'exemple illustré sur la figure 6.21.

FIGURE 6.21 – Ressemblance entre le voisinage de deux atomes a_1 et a'_1.

A l'itération *n*, le degré d'inclusion entre les paires d'atomes (a_1, a'_1) et (a_2, a'_2) est défini par les équations suivantes :

$$\begin{aligned}
\mathrm{INC}^{(n)}(a_1, a'_1) &= \frac{1}{2}\Big(\mathrm{INC}^{(n-1)}(a_2, a'_2) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big) \\
\mathrm{INC}^{(n)}(a_2, a'_2) &= \frac{1}{2}\Big(\mathrm{INC}^{(n-1)}(a_1, a'_1) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big)
\end{aligned}$$

A l'itération $n+1$, on a :

$$\begin{aligned}
\mathrm{INC}^{(n+1)}(a_1, a'_1) &= \frac{1}{2}\Big(\mathrm{INC}^{(n)}(a_2, a'_2) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big) \\
\mathrm{INC}^{(n+1)}(a_2, a'_2) &= \frac{1}{2}\Big(\mathrm{INC}^{(n)}(a_1, a'_1) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big)
\end{aligned}$$

Et en introduisant, dans les termes de droite, les valeurs de $\mathrm{INC}^{(n)}(a_2, a'_2)$ et $\mathrm{INC}^{(n)}(a_1, a'_1)$ à l'itération *n*, le système s'écrit :

$$\begin{aligned}
\mathrm{INC}^{(n+1)}(a_1, a'_1) &= \frac{1}{2}\Big(\mathrm{INC}^{(n-1)}(a_1, a'_1) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big) \\
\mathrm{INC}^{(n+1)}(a_2, a'_2) &= \frac{1}{2}\Big(\mathrm{INC}^{(n-1)}(a_2, a'_2) + Di_{lit}\big(\mathrm{INC}, \{a_1, a_2\}, \{a'_1, a'_2\}, a_2, a'_2\big)\Big)
\end{aligned}$$

On voit alors qu'à l'itération n, on intègre uniquement le degré d'inclusion des voisins immédiats mais à l'itération $n+1$, on intègre d'une certaine manière la valeur de son propre degré d'inclusion à l'itération précédente. La définition récursive de I_π entraîne que le degré d'inclusion de l'environnement d'une paire d'atomes (a_1, a_2) n'est jamais évalué indépendamment de son propre degré d'inclusion. On peut alors se demander si l'on se tient pas trop compte de la similarité de l'atome avec lui-même, au détriment de son voisinage. Ce phénomène n'apparaît pas avec le noyau AO qui tient compte de la similarité du voisinage de deux atomes, indépendamment de leur propre similarité.

Dans toutes les expérimentations réalisées, la valeur du coefficient α a été fixée à 0.5 (voir 3.2.3). Une manière de rétablir l'équilibre entre le degré d'inclusion des atomes et celui de leur environnement serait d'augmenter la valeur de ce coefficient. Un test avec la variante I_π_Sim_Kuhn et $\alpha = 0.75$ donne, pour le nombre de classes attendues, les valeurs de IC suivantes (tableau 6.9).

Algorithme	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
I_π_Sim_Kuhn (α = 0.25)	0.64	1.38	2.36	0.70
I_π_Sim_Kuhn (α = 0.50)	0.64	1.45	2.23	0.33
I_π_Sim_Kuhn (α = 0.75)	0.67	1.40	1.90	0.40
noyau AO	0.54	0.82	1.64	0.10

TABLEAU 6.9 – Comparaison de l'algorithme pour des valeurs différentes du paramètre α.

Sur la base DHFR, l'augmentation de la valeur de α conduit à de meilleurs résultats. Sur les bases COX2 et ER, le passage de 0.25 à 0.50 permet également d'observer de meilleurs résultats ; par contre, lorsque $\alpha = 0.75$, les performances se dégradent. Enfin, sur BZR, les performances se dégradent pour $\alpha = 0.75$. Les résultats s'approchent du noyau noyau AO sans toutefois les égaler. La prise en compte plus (α proche de 1) ou moins (α proche de 0) importante de la topologie des graphes moléculaires a manifestement une influence sur la qualité des classifications. Cependant, il reste difficile de déterminer *a priori* la valeur optimale de ce paramètre, qui dépend beaucoup de la nature des données. Dans des cas comme les bases COX2 ou ER, il semble qu'une bonne classification soit obtenue en tenant compte, à parts égales, de la ressemblance entre les atomes et celle de leur environnement. Ce n'est pas le cas pour la base DHFR.

Dans le noyau AO, le facteur de décroissance γ est utilisé pour pondérer la contribution des voisins situés à une distance l des atomes considérés (voir équation 5.16, section 5.4.2.1). Dans I_π, le facteur de décroissance est implicite du fait de la définition récursive : il est en α^n/V où n est le nombre d'itérations utilisées pour calculer la matrice INC et V le nombre moyen d'atomes voisins pour un atome donné.

Pour comparer ces deux facteurs de décroissance, nous prenons l'exemple suivant dans lequel on considère que le calcul se fait en 5 itérations ($0 \leq n \leq 4$), que les atomes ont en moyenne 4 voisins, et que $\alpha = 0.5$. Dans ce cas, le facteur de décroissance de I_π est en $(1/8)^n$ et le facteur γ s'écrit :

$$\gamma = \left(1 - \frac{n+1}{5}\right)^2$$

Les valeurs de ces facteurs, pour les itérations 0 à 4, sont les suivantes :
- pour I_π : 0.125, 15.10^{-3}, $1,95.10^{-3}$, 244.10^{-6}, 30.10^{-6} ;
- pour γ : 0.64, 0.36, 0.16, 0.04, 0

Le facteur de décroissance de I_π est toujours inférieur au coefficient γ, ce qui signifie qu'en pratique, le voisinage pris en compte par I_π n'est pas aussi étendu que dans le noyau AO, ce qui nuirait à la bonne prise en compte de la topologie locale des atomes.

6.4.2 Comparaison des variantes de I_π avec le noyau ISOA

Nous comparons maintenant le noyau ISOA avec les variantes de I_π qui s'en rapprochent le plus et notamment avec la variante I_π_Sim_Kuhn qui ne diffère du noyau ISOA que par la fonction Di_{lit}. Pour cette étude, les deux paramètres suivants de l'algorithme I_π sont modifiés :
- le calcul d'une similarité ou d'un degré d'inclusion au niveau de la ressemblance entre les atomes/molécules ;
- la fonction Di_{lit} qui évalue la ressemblance entre les liaisons entre deux atomes ainsi que leur voisins respectifs. Dans le noyau ISOA, cette fonction est définie par le produit entre la similarité des voisins et celle des liaisons. Dans l'algorithme I_π, nous testons la moyenne de ces deux termes ainsi que le produit.

Pour toutes les variantes de I_π, l'appariement global est réalisé par l'algorithme hongrois, comme c'est le cas pour le noyau ISOA. Les variantes I_π_Inc_Kuhn et I_π_Sim_Kuhn sont identiques à celles utilisées lors de la comparaison entre l'algorithme I_π et le noyau AO ; entre autres, la fonction Di_{lit} est définie par la moyenne. La variante I_π_Inc_Kuhn_P (resp. I_π_Sim_Kuhn_P) diffère de I_π_Inc_Kuhn (resp. I_π_Sim_Kuhn) par le calcul de la ressemblance entre les voisins de deux atomes donnés qui est définie ici par le produit entre la ressemblance des voisins et celle des liaisons au lieu de calculer la moyenne.

Le tableau 6.10 indique les valeurs de l'indice IC pour le nombre de classes attendues sur les quatre chimiothèques. Les courbes des indices IC sont affichées dans les figures 6.22 et 6.23.

Algorithme	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
I_π_Sim_Kuhn	0.64	1.45	2.23	0.33
I_π_Inc_Kuhn	0.60	1.16	1.93	0.33
I_π_Inc_Kuhn_P	0.43	0.31	**1.06**	0.23
I_π_Sim_Kuhn_P	**0.35**	**0.24**	1.14	0.21
noyau ISOA	**0.35**	0.49	1.38	**0.00**

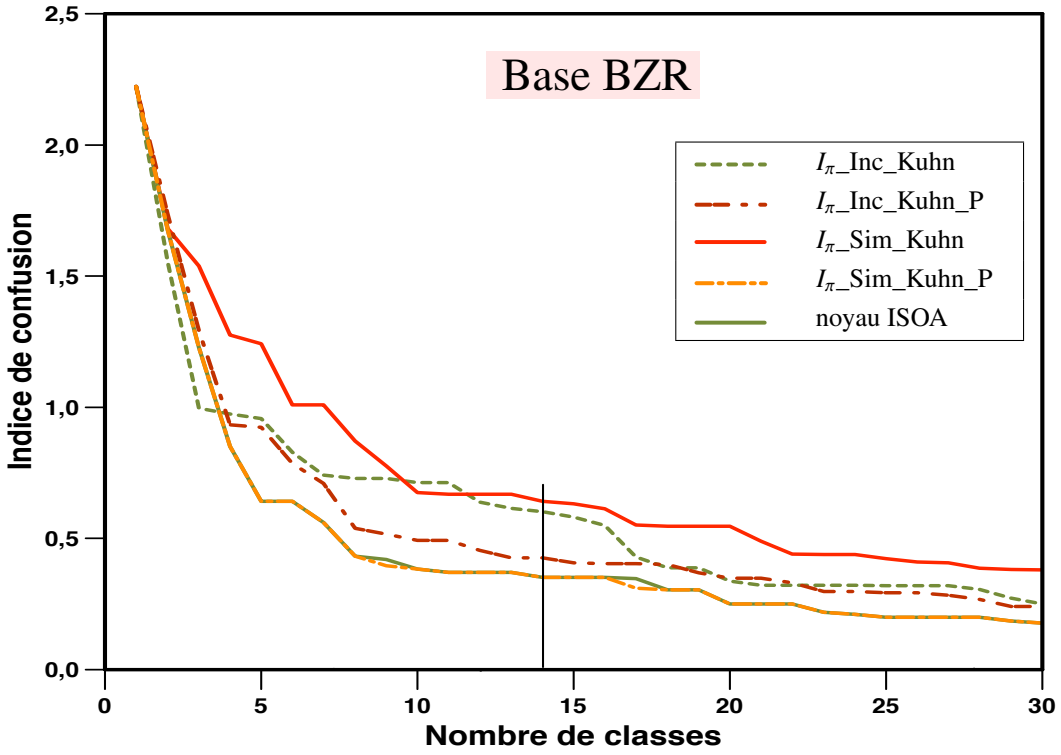
TABLEAU 6.10 – Résultats numériques (indice IC) des variantes de l'algorithme I_π dans la comparaison avec le noyau ISOA au point de coupure du nombre de classes attendues.

Ces résultats montrent que le changement de la définition de Di_{lit} (passer de la moyenne au produit) améliore très nettement les résultats. Sur l'ensemble des quatre bases, les indices IC des deux variantes utilisant le produit (I_π_Inc_Kuhn_P et I_π_Sim_Kuhn_P) sont meilleurs que les variantes correspondantes avec la moyenne (I_π_Sim_Kuhn et I_π_Inc_Kuhn). Les résultats de la variante I_π_Sim_Kuhn_P, qui est la plus proche du noyau ISOA, sont similaires à ce noyau sur les bases BZR et COX2 mais moins sur DHFR ou ER ; ce qui peut s'expliquer par des implémentations différentes. Contrairement à l'étude comparative entre I_π et le noyau AO où les variantes symétriques étaient systématiquement moins performantes que les variantes asymétriques, on n'observe pas ici cette dichotomie entre les variantes symétriques et asymétriques. Ici, la variante symétrique avec le produit (I_π_Sim_Kuhn_P) est la meilleure des variantes, excepté sur la base DHFR.

Pour comprendre ces résultats, nous nous intéressons au contenu des matrices. En ce qui concerne la fonction Di_{lit}, comme le produit met à zéro tous les voisinages qui correspondent à des liaisons différentes, la matrice K (ou la matrice INC pour la variante I_π_Sim_Kuhn) doit contenir plus de valeurs nulles que . Le tableau 6.11 montre le pourcentage de valeurs nulles pour la variante I_π_Sim_Kuhn dans les cas où la fonction Di_{lit} est définie par la moyenne ou le produit.

Définition de Di_{lit}	BZR	COX2	DHFR	ER
moyenne	0.0	0.0	0.0	0.0
produit	22.1 ± 2.8	27.7 ± 2.4	17.7 ± 3.8	4.8 ± 3.0

TABLEAU 6.11 – Pourcentage de valeurs nulles dans la matrice INC pour la variante I_π_Sim_Kuhn avec deux définitions de la fonction Di_{lit}.



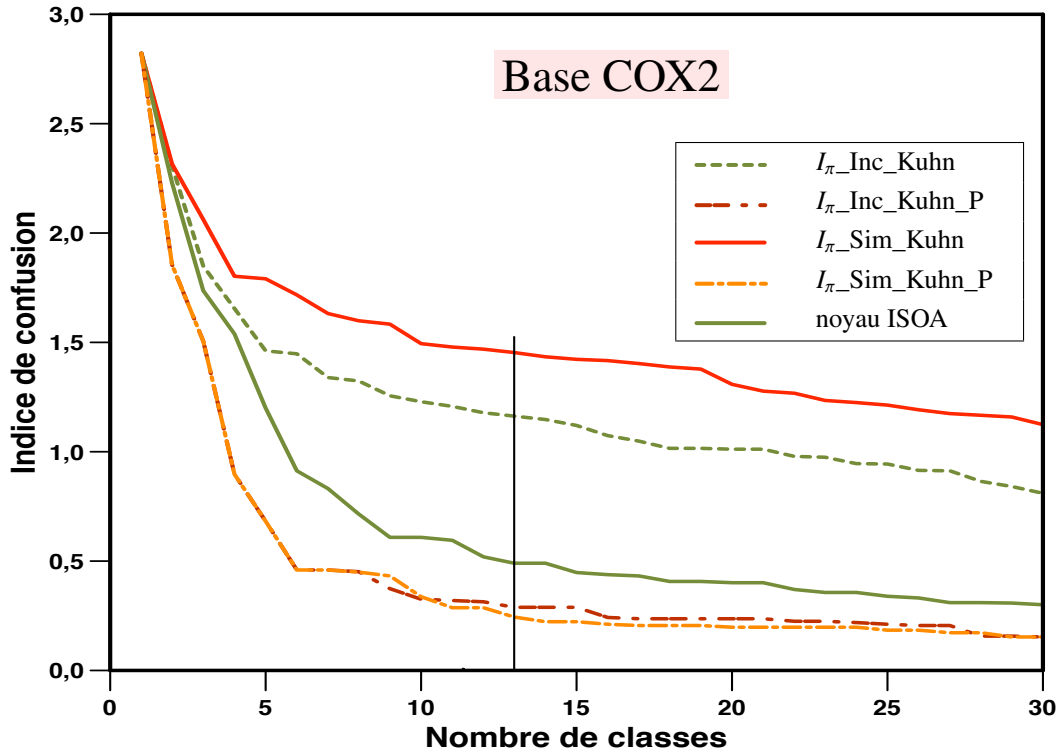


FIGURE 6.22 – Comparaison des variantes de l'algorithme I_π avec le noyau ISOA pour les bases BZR et COX2.

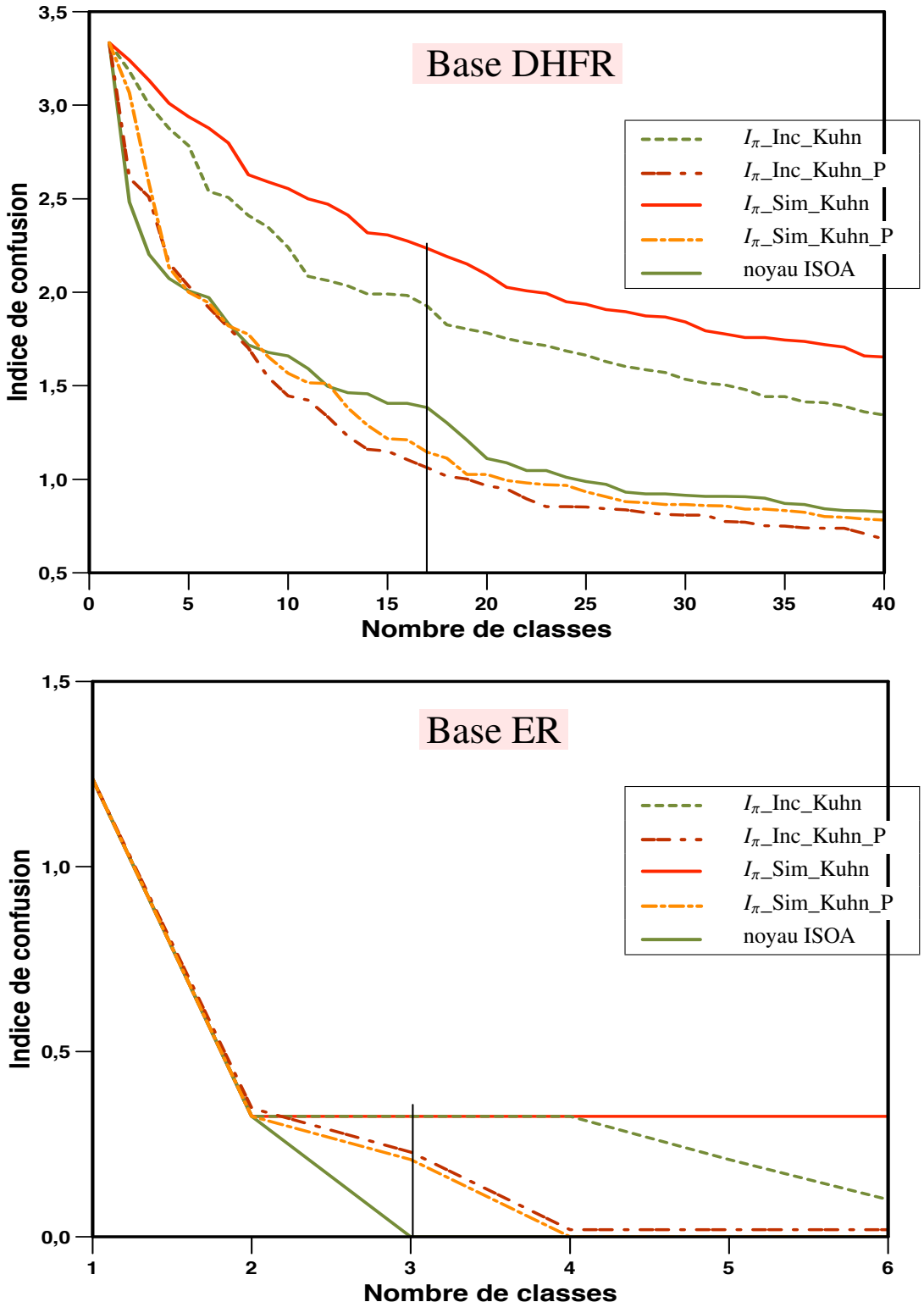


FIGURE 6.23 – Comparaison des variantes de l'algorithme I_π avec le noyau ISOA pour les bases DHFR et ER.

Lorsque la fonction Di_{lit} est définie par la moyenne, aucune matrice ne contient de valeurs nulles, contrairement aux matrices calculées avec le produit. Dans ce dernier cas, comme les valeurs nulles sont en moyenne peu pertinentes, cela évite de sélectionner de mauvais appariements ; c'est une manière de réduire encore le nombre de candidats à l'appariement. Ceci explique que l'appariement global avec l'algorithme hongrois donne de meilleurs résultats.

Par contre, lorsque l'appariement global est réalisé par le parcours de graphes, l'utilisation du produit dans la définition de la fonction Di_{lit} n'améliore pas réellement les résultats (voir l'annexe C). L'algorithme de recherche d'APC semble suffisamment efficace pour explorer l'espace de recherche sans qu'il soit nécessaire de compléter les informations par une matrice avec des valeurs contrastées. On retrouve ce que l'on observe dans les variantes APC_Glouton_AvecArret et APC_Stochast_AvecArret (voir section 6.2) qui obtenaient de très bons résultats sans utiliser la matrice INC.

6.5 Conclusion

Dans ce travail, la méthode I_π a été appliquée à un problème de classification automatique de molécules. Quatre chimiothèques, classiquement utilisées dans la littérature, ont servi de base pour réaliser ces expériences. Nous avons observé qu'une mesure de similarité moléculaire basée sur la SSCM ne permet pas toujours de capturer l'information utile à la reconstruction des familles chimiques. En effet, dans certaines bases, les sous-structures utiles à la classification sont des sous-ensembles du châssis moléculaire et que dans d'autres cas, ce sont des sur-ensembles.

La méthode a également été comparée à d'autres mesures de similarité (des fonctions noyau entre autres) et donne souvent les meilleurs résultats que ce soit sur des méthodes qui comparent directement les graphes moléculaires ou qui comparent des ensembles de chemins extraits des graphes. L'algorithme n'a pas de paramètres réellement explicites mais intègre plutôt différentes stratégies qui peuvent être utilisées suivant le domaine d'application. En particulier, la comparaison avec le noyau ISOA a permis de mettre en évidence quelques points intéressants.

L'utilisation du produit dans la fonction Di_{lit} améliore beaucoup les résultats lorsque l'appariement global est réalisé par l'algorithme hongrois car cela permet de rendre les valeurs dans la matrice plus contrastées. Malgré tout, la stratégie qui consiste à rechercher des APC (avec la fonction Di_{lit} définie par la moyenne) reste la plus performante ce qui indique que la prise en compte de la topologie des graphes moléculaires reste importante dans la mesure où la matrice INC ne contient pas suffisamment d'information structurelle sur les molécules. On s'oriente même vers une approche stochastique (variantes "stochastiques" de l'algorithme) de l'appariement.

De manière générale, l'avantage de I_π est qu'il est défini de manière générique et peut s'appliquer à de nombreux types d'objets structurés tandis que le noyau ISOA n'est défini que pour des graphes moléculaires.

Ce travail pourrait être complété par une étude comparative avec d'autres algorithmes de classification que la CAH. On pourrait par exemple tester l'algorithme de Jarvis-Patrick [Jarvis and Patrick, 1973], largement utilisé dans le domaine de la chimie. La difficulté cependant réside dans le fait qu'il possède plusieurs paramètres explicites, ce qui rend l'analyse et l'interprétation des résultats plus délicate.

Dans les expériences rapportées ici, la classification des experts était connue d'avance, ce qui nous a permis de comprendre le comportement des variantes des algorithmes et de comparer la méthode avec d'autres mesures de similarité. Lorsque la classification initiale n'est pas connue, les indices IS et IC ne peuvent pas être utilisés pour répondre aux questions qui se posent alors : quel est le nombre de familles à retrouver ? Sur quels critères ? Quelle variante de l'algorithme utiliser et surtout comment la choisir *a priori* ?

Conclusion et perspectives

Depuis plusieurs années, la recherche de molécules bio-actives est un enjeu majeur pour la biologie et la recherche thérapeutique. L'objectif est d'identifier, dans de grandes collections de molécules (chimiothèques), des molécules actives sur une cible biologique donnée puis de rechercher (et/ou synthétiser) des analogues structuraux de manière à optimiser leur spécificité et leur sensibilité. De telles molécules peuvent ensuite servir d'outil chimique pour la recherche en biologie ou devenir des candidats médicaments. L'utilisation de plates-formes automatisées permet de tester un très grand nombre de molécules mais qui reste petit devant la taille de l'espace chimique à explorer et la conception rationnelle des chimiothèques devient donc un enjeu crucial. Dans ce contexte, les chimistes ont besoin d'outils automatiques pour évaluer leur similarité et étudier la diversité des chimiothèques et plus fondamentalement, comparer des molécules qui sont des objets structurés

Dans ce contexte, les techniques d'analyse automatique comme l'apprentissage artificiel peuvent apporter des solutions méthodologiques. D'une part, ces méthodes peuvent utiliser des langages capables de capturer l'information structurelle d'objets complexes comme des molécules et d'autre part, elles se basent sur des opérateurs de comparaison comme la similarité ou l'inclusion qui sont tout à fait appropriés aux besoins des analyses en chimie.

Le travail de thèse présenté dans ce mémoire s'inscrit dans le cadre de la comparaison d'objets structurés et plus particulièrement la recherche et la quantification, au moyen d'une valeur réelle, de l'inclusion de deux objets structurés ; ce qui correspond à la notion de subsomption en logique du premier ordre et d'isomorphisme de sous-graphe partiel pour les graphes. La première partie propose un algorithme qui quantifie un degré d'inclusion entre deux objets structurés. Dans la seconde partie, une mesure de similarité entre molécules, définie à partir de l'algorithme développé dans ce travail, est utilisée pour classer automatiquement des molécules et est comparée à d'autres mesures de similarité actuelles.

Degré d'inclusion entre objets structurés

Le problème de la quantification d'un degré d'inclusion entre deux objets structurés a été abordé ici de façon générique. Dans la description de l'algorithme, les objets sont représentés sous forme de clauses Datalog mais il s'adapte naturellement à d'autres formalismes comme les graphes et hypergraphes ou encore les problèmes de satisfaction de contraintes.

L'approche consiste à rechercher le meilleur appariement entre les variables de deux clauses h et ex en identifiant une sous-structure commune à ces deux clauses puis en quantifiant le degré de recouvrement de cette sous-structure commune par rapport à la clause h. Les heuristiques implémentées dans l'algorithme (matrice d'inclusion entre les variables de chaque clause, stratégies d'exploration des solutions) permettent d'une part de réduire la taille de l'espace de recherche (*i.e.* en réduisant le nombre de variables candidates à l'appariement) et d'autre part de guider efficacement la recherche dans ce même espace.

En fonction du domaine d'application et des informations disponibles sur les données, la représentation des objets peut être facilement enrichie par des descripteurs numériques ou symboliques. Il peut également être nécessaire d'utiliser différentes stratégies d'appariement des variables ainsi que le type de sous-structures communes qui peuvent être connexes ou non connexes. Tous ces aspects sont intégrés à l'algorithme qui est basé sur un ensemble de fonctions génériques, définies suivant l'application.

L'algorithme étant basé sur des heuristiques, il recherche la meilleure solution possible mais sans garantir de trouver une solution complète si elle existe (*i.e.* si h est totalement incluse dans la clause ex) : l'algorithme est dit incomplet. La capacité de l'algorithme à identifier une solution complète a été évaluée dans le cadre de la transition de phase où les problèmes sont connus pour être difficiles. Dans le cas de la logique du premier ordre, les résultats ont montré l'intérêt d'utiliser la matrice INC pour guider la recherche car cela permet de retrouver la solution complète sur les instances les plus difficiles beaucoup plus rapidement que les méthodes non guidées par la matrice. Pour compléter l'étude, il faudrait positionner l'algorithme I_π par rapport à d'autres algorithmes incomplets.

Application à la chimie

L'algorithme a été appliqué à la chimie en adaptant les différentes fonctions génériques de l'algorithme à la spécificité des graphes moléculaires. Une mesure de similarité a été dérivée des degrés d'inclusion entre deux graphes moléculaires et a été utilisée dans une tâche de classification automatique de composés. Les expérimentations ont été menées sur quatre chimiothèques publiques de diversité différente et les résultats

différents d'une base à l'autre ont permis d'étudier de façon plus approfondie le comportement de l'algorithme.

Nous avons notamment observé que l'utilisation de la plus grande sous-structure commune pour définir une mesure de similarité moléculaire n'était pas toujours un gage de performance surtout dans le cas où elles sont plus grandes que le châssis moléculaire ; de meilleures classifications sont obtenues en se basant sur des sous-structures communes plus petites que le châssis moléculaire. C'est un résultat intéressant dans la mesure où de nombreuses mesures de similarité moléculaire sont basées sur la plus grande sous-structure commune entre molécules.

La mesure de similarité a également été comparée à d'autres mesures de similarité moléculaire pour classer des composés et se révèle l'une des plus performantes pour retrouver les classifications des experts. La comparaison avec deux fonctions noyau très proches de I_π (le noyau ISOA [Rupp *et al.*, 2007] et le noyau AO [Fröhlich *et al.*, 2005a]) a mis en évidence l'intérêt de la recherche d'APC pour l'appariement global des molécules par rapport à l'algorithme hongrois utilisé dans les deux fonctions noyau. De plus, dans le cas de graphes simples comme des graphes moléculaires, il n'est pas nécessaire d'utiliser la matrice de ressemblance entre atomes pour apparier correctement les molécules car les descripteurs utilisés ainsi que la structure des graphes elle-même rendent le problème d'appariement suffisamment contraint. On débouche même vers une approche stochastique du problème.

Perspectives

Améliorations de l'algorithme

Comme il a déjà été évoqué dans le chapitre 3, le problème de la sur-évaluation des valeurs d'inclusion dans la matrice pourrait être géré par la mise à jour de la matrice à chaque itération de l'appariement global. Par exemple, chaque fois qu'une variable v de la clause h est appariée avec une variable v' de la clause ex, il faudrait revenir au calcul de la matrice en fixant l'appariement de v avec v' de façon à ce que les choix d'appariement réalisés dans l'étape globale soient cohérents avec les appariements locaux du calcul de INC.

Le deuxième point à travailler concerne la différenciation des cycles lors du calcul de la matrice INC. Nous avons vu que le problème était directement lié au nombre de variables voisines pour une variable donnée. La manière d'aborder ce problème reste assez floue. Il serait également utile de recenser les cas où ce problème apparaît en pratique, suivant les domaines et les langages de représentation utilisés de manière à évaluer l'impact réel sur le degré d'inclusion.

En ce qui concerne l'implémentation de l'algorithme, une réflexion devrait également être menée au sujet de la généricité de l'algorithme en ce qui concerne les langages utilisés pour représenter les objets. Dans sa définition, l'algorithme permet l'utilisation de langages très expressifs mais en pratique, on se retrouve malgré tout dans le compromis classique d'une implémentation ad-hoc rapide ou d'une implémentation très générique mais qui peut considérablement augmenter les temps de calcul.

Analyse de chimiothèques

L'étude expérimentale réalisée en chimie a porté sur des chimiothèques publiques dans lesquelles les familles chimiques sont déjà connues. L'étape suivante serait d'opérer un passage à l'échelle sur de plus grandes chimiothèques en réduisant les temps de calcul. Une première étape de parallélisation pourrait facilement être envisagée : sur un ordinateur parallèle, on pourrait facilement répartir le calcul de la similarité d'une paire de molécules sur des processeurs différents.

La plate-forme de criblage du CMBA au CEA-Grenoble dispose de collections de plusieurs dizaines de milliers de composés. Dans le cadre des criblages à haut débit réalisés sur cette plate-forme, l'algorithme I_π pourrait alors répondre aux besoins suivants :

- analyser la diversité des chimiothèques fournies au CMBA ;
- sélectionner dans les chimiothèques, des molécules de façon à construire des sous-chimiothèques très diverses ou au contraire, ciblées sur une structure chimique particulière ;
- rechercher dans les chimiothèques des analogues structuraux aux touches identifiées lors d'un criblage, en s'appuyant sur une ou plusieurs sous-structures communes.

Prédiction de bio-activité

A partir d'un ensemble de molécules actives et inactives sur une cible biologique, les techniques d'apprentissage artificiel supervisé construisent des modèles qui permettent de discriminer les molécules actives de celles qui sont inactives. Ces modèles permettent ensuite de prédire l'activité d'une nouvelle molécule. Cela peut également contribuer à l'acquisition de nouvelles connaissances comme les mécanismes qui rendent une molécule active.

Depuis plusieurs années, de nombreux travaux ont montré es très bonnes performances des machines à vecteurs supports (basées sur des fonctions noyau) sur des problèmes de prédiction de bio-activité [], [], []. A l'instar des fonctions noyau, il serait intéressant d'utiliser la mesure de similarité issue de l'algorithme I_π dans ce type

d'approche pour évaluer ses performances sur ce problème d'apprentissage supervisé et la comparer à d'autres fonctions noyau (dont celles décrites dans ce travail). Dans ce type d'approche, la démarche expérimentale est la suivante. A partir d'une chimiothèque contenant les informations de bio-activité de chaque molécule, deux ensembles de molécules sont construits : un ensemble d'apprentissage pour apprendre le modèle de bio-activité et un ensemble de test pour vérifier la validité de ce modèle. La matrice de distance entre composés, calculée par l'algorithme I_π serait utilisée par une machine à vecteurs supports (à la place de celle générée par une fonction noyau) pour construire un modèle de bio-activité à partir de l'ensemble d'apprentissage. La performance de l'approche est ensuite évaluée en prédisant l'activité des molécules de l'ensemble de test. Pour quantifier cette performance, on calcule en général le pourcentage de molécules correctement prédites. On peut également s'appuyer sur les notions de *spécificité* et de *sélectivité* par l'intermédiaire de la courbe ROC.

L'utilisation de l'algorithme I_π soulève cependant une question théorique importante. En effet, une des caractéristiques importantes des fonctions noyau utilisées dans une machine à vecteurs support est qu'elles doivent être semi-positives définies, c'est-à-dire que la matrice de similarité entre les composés est xxxx. Si ce n'est pas le cas (comme c'est effectivement vérifié pour le noyau AO [Vert, 2008]), cela soulève un problème théorique important : comment utiliser des fonctions non semi-positives définies dans leur utilisation dans une machine à vecteurs support ?

Troisième partie

Annexes

Annexe A

Paramètres optimaux pour le noyau spectral et le noyau de Tanimoto

Pour le noyau spectral et le noyau de Tanimoto, nous avons retenu les valeurs de l et u qui donnaient les meilleurs résultats. Elles sont résumées dans le tableau A.1.

Chimiothèque	noyau de Tanimoto	noyau spectral
BZR	u=10	u=10
COX2	u=10	l=10
DHFR	u=10	l=5
ER	l=20	l=5

Tableau A.1 – Valeurs optimales pour les paramètres l et u du noyau spectral et du noyau de Tanimoto, pour le nombre de classes attendues.

Les figures A.1 et A.2 présentent les résultats des indices IC pour les 4 bases et pour des valeurs de l et u comprises entre 1 et 20.

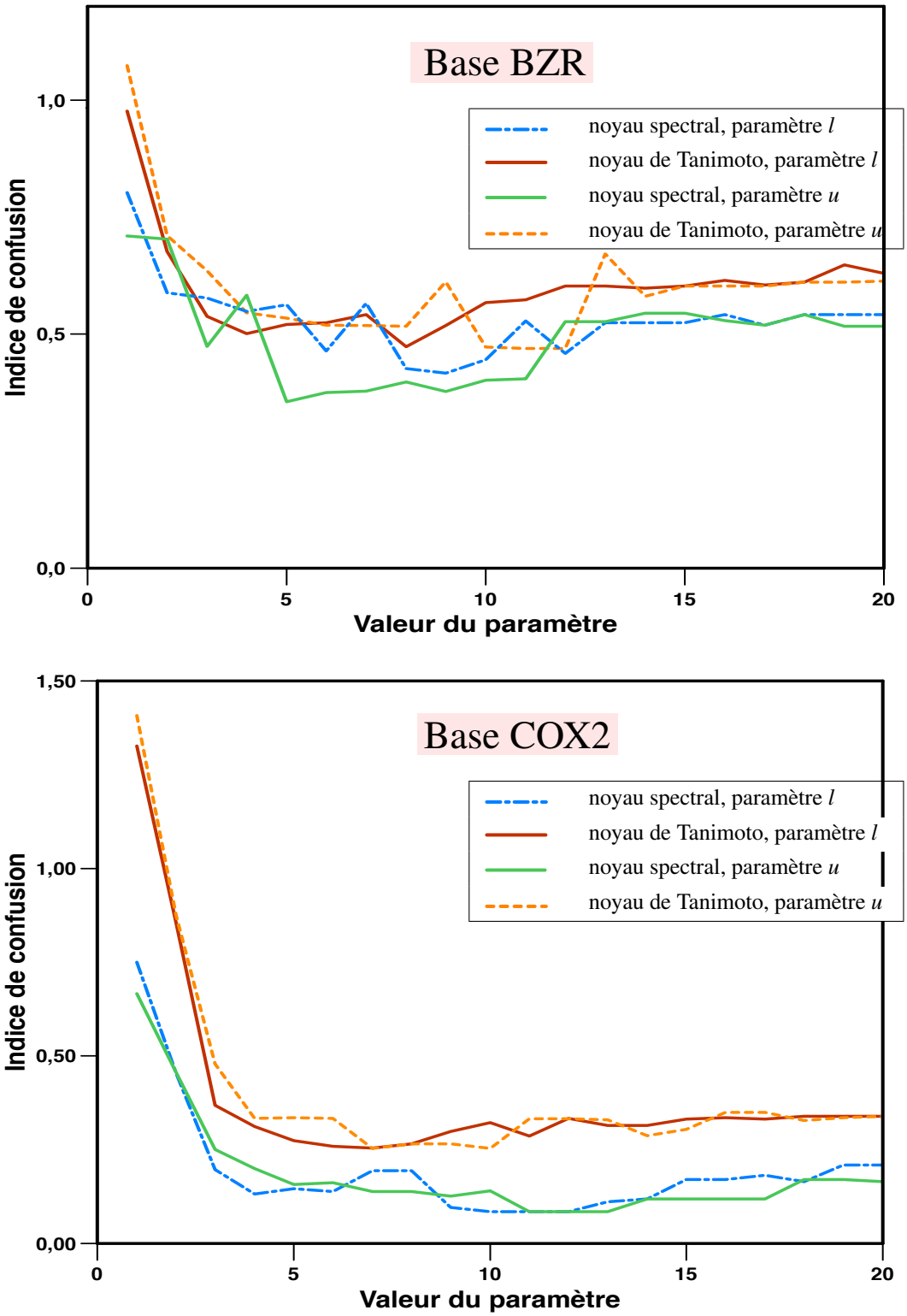


FIGURE A.1 – Indices IC pour différentes valeurs des paramètres l et u du noyau de Tanimoto et du noyau spectral, pour les bases BZR et COX2.

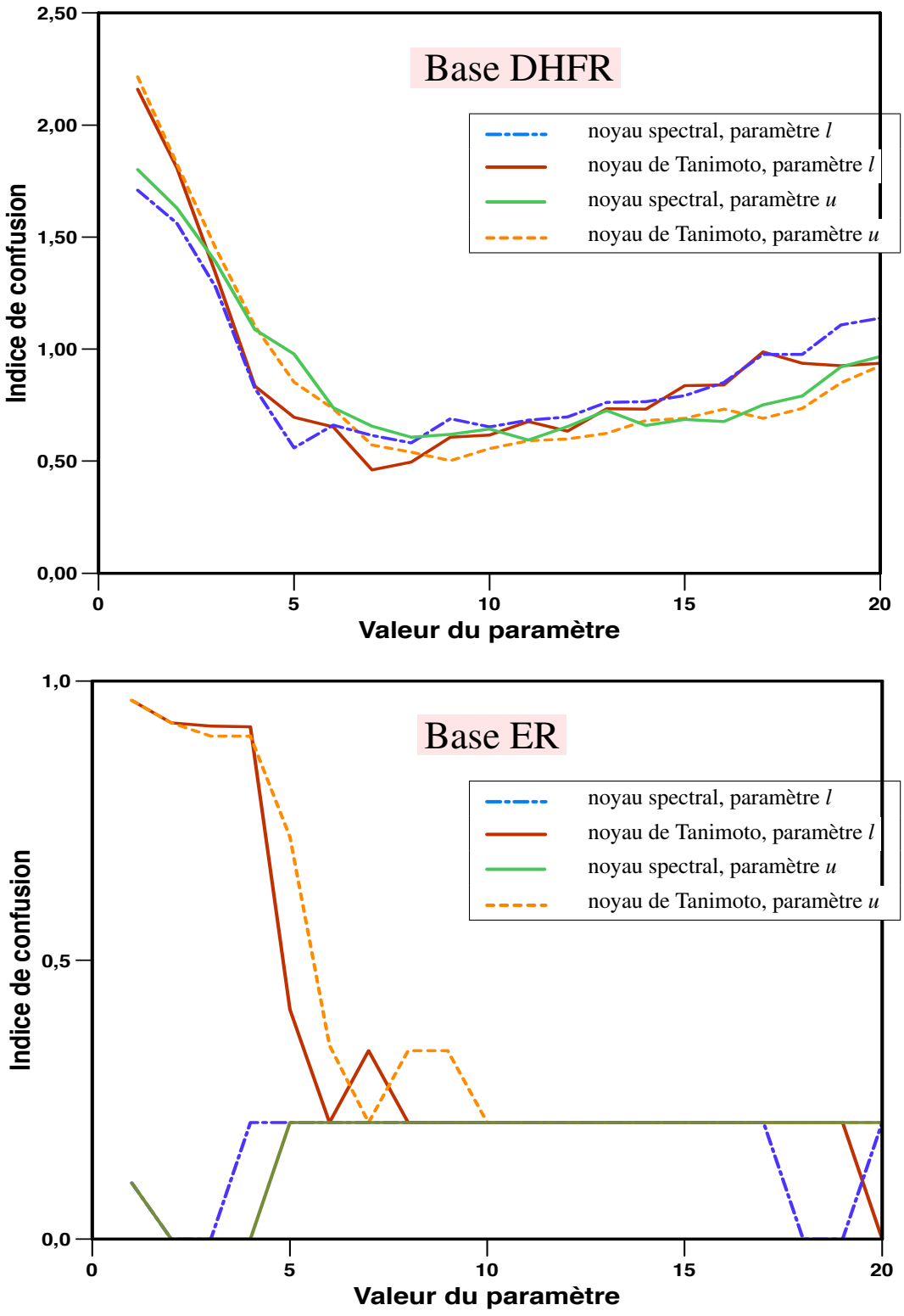


FIGURE A.2 – Indices IC pour différentes valeurs des paramètres l et u du noyau de Tanimoto et du noyau spectral, pour les bases DHFR et ER.

Annexe B

Influence du nombre de graines

B.1 Variantes stochastiques de l'algorithme I_π

Les figures B.1 et B.2 présentent les résultats des indices IC pour les 4 bases, pour les variantes APC_Stochast_25, APC_Stochast_50, APC_Stochast_75, APC_Stochast_Filtre et APC_Stochast_SansArret.

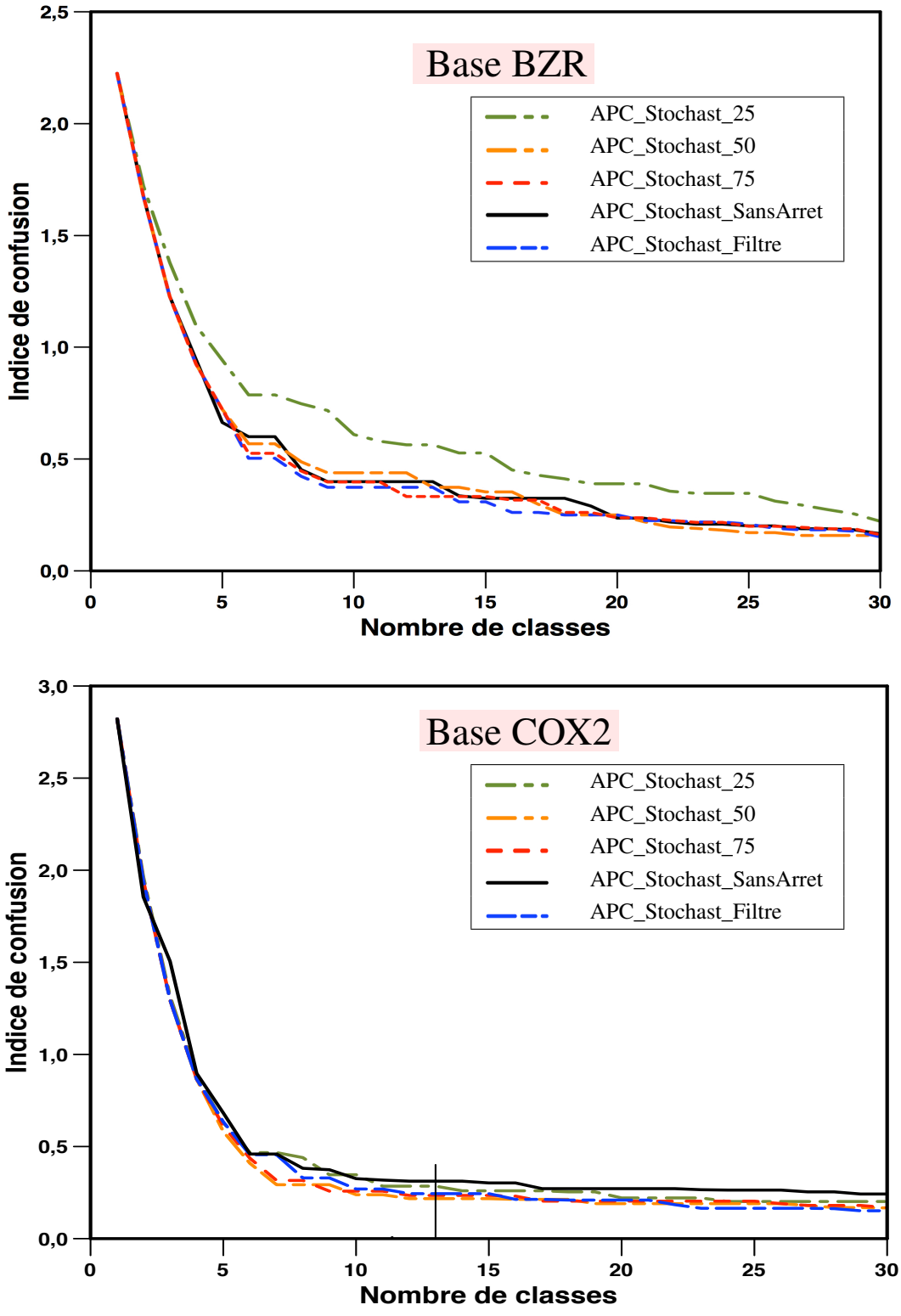


FIGURE B.1 – Indices ICpour des nombres de graines différents avec la variante APC_Stochast_*, sur les bases BZR et COX2.

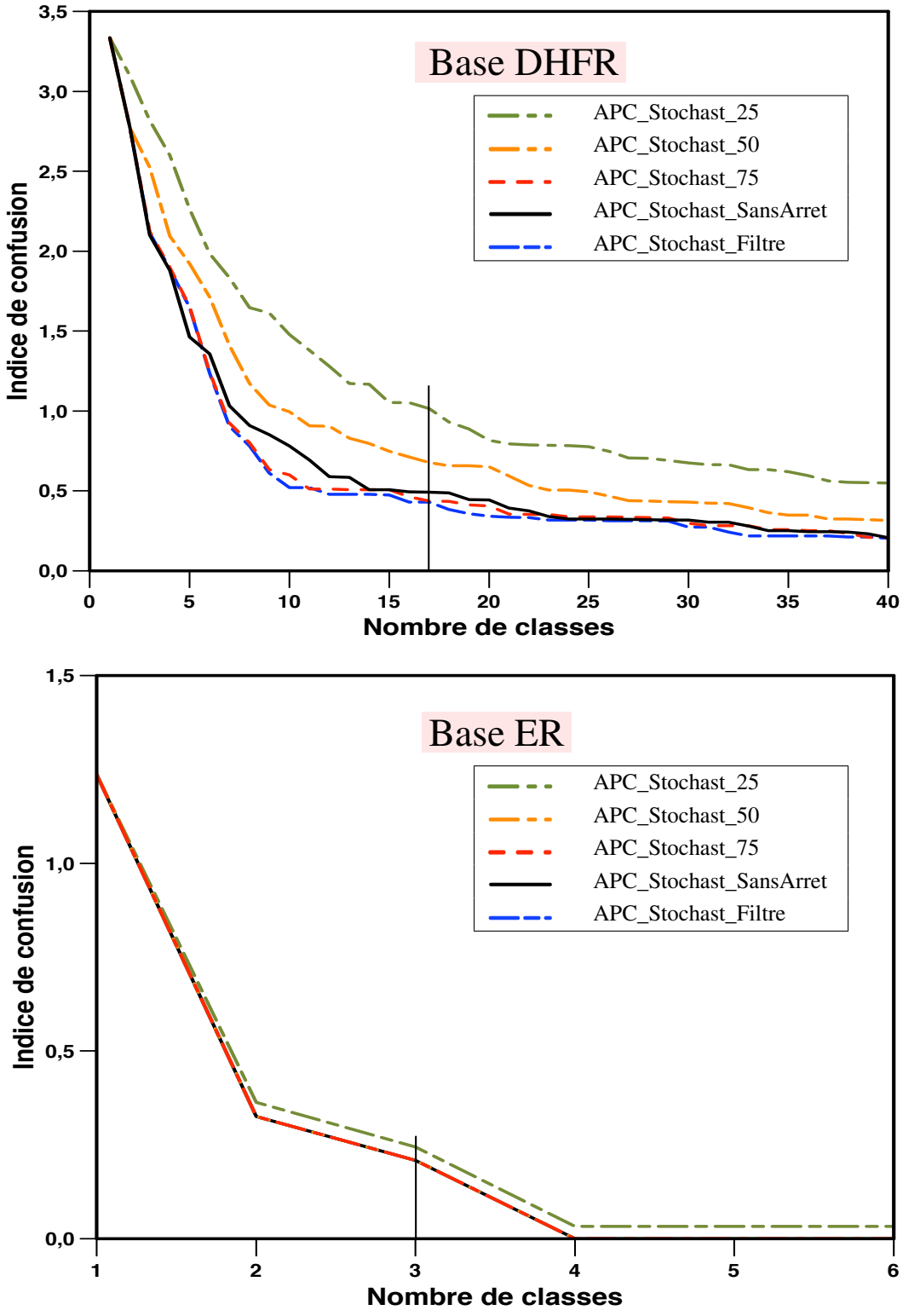


FIGURE B.2 – Courbes des indices IC pour des nombres de graines différents avec la variante APC_Stochast_*, sur les bases DHFR et ER.

B.2 Variantes de l'algorithme I_π avec l'algorithme hongrois

Les figures B.3 et B.4 présentent les résultats des indices IC, pour les 4 bases, pour les variantes APC_Kuhn_25, APC_Kuhn_50, APC_Kuhn_75, APC_Kuhn_Filtre et APC_Kuhn_SansArret qui utilisent l'algorithme hongrois.

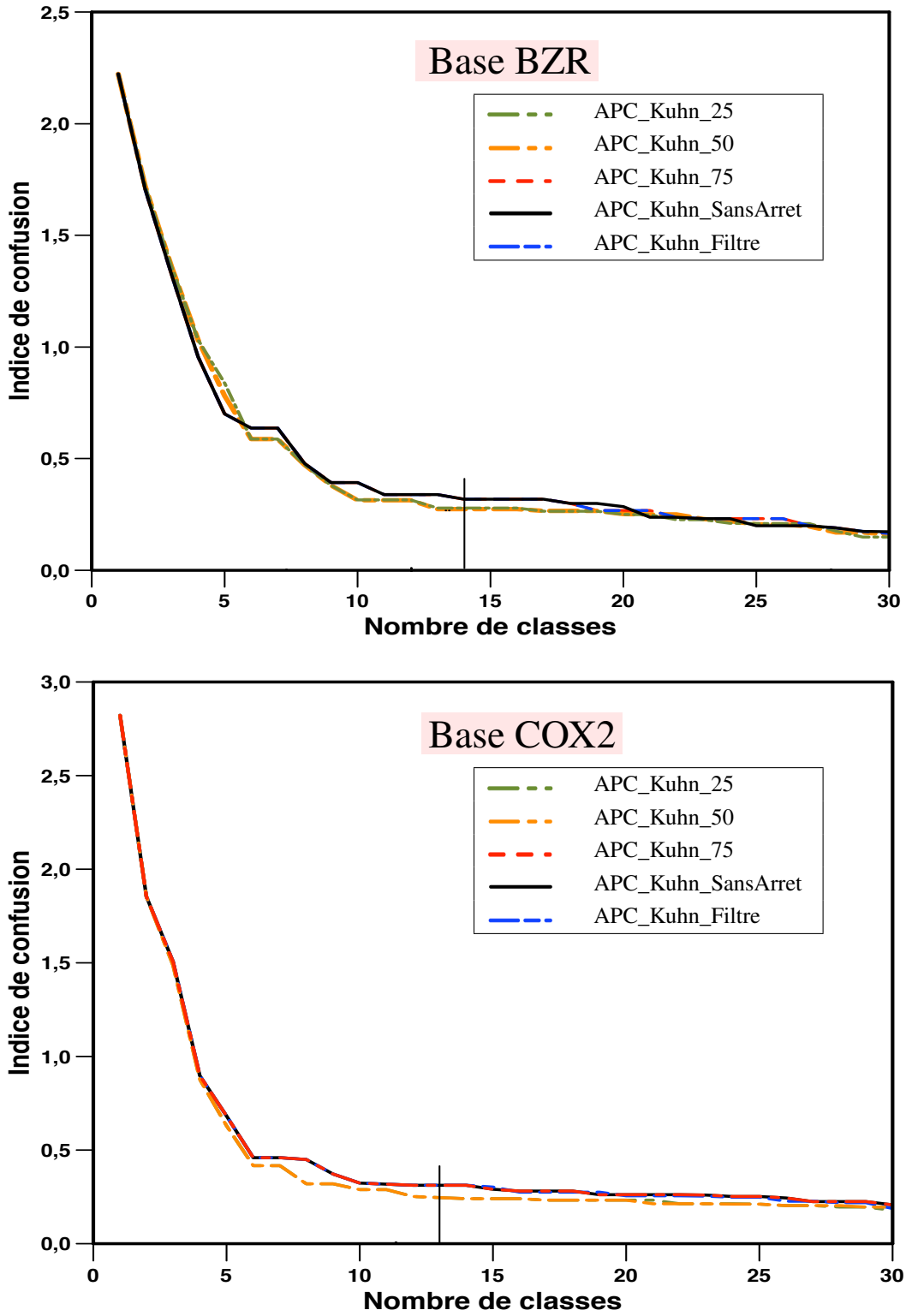


FIGURE B.3 – Courbes des indices IC pour des nombres de graines différents avec la variante Ipi_*_APC, sur les bases BZR et COX2.

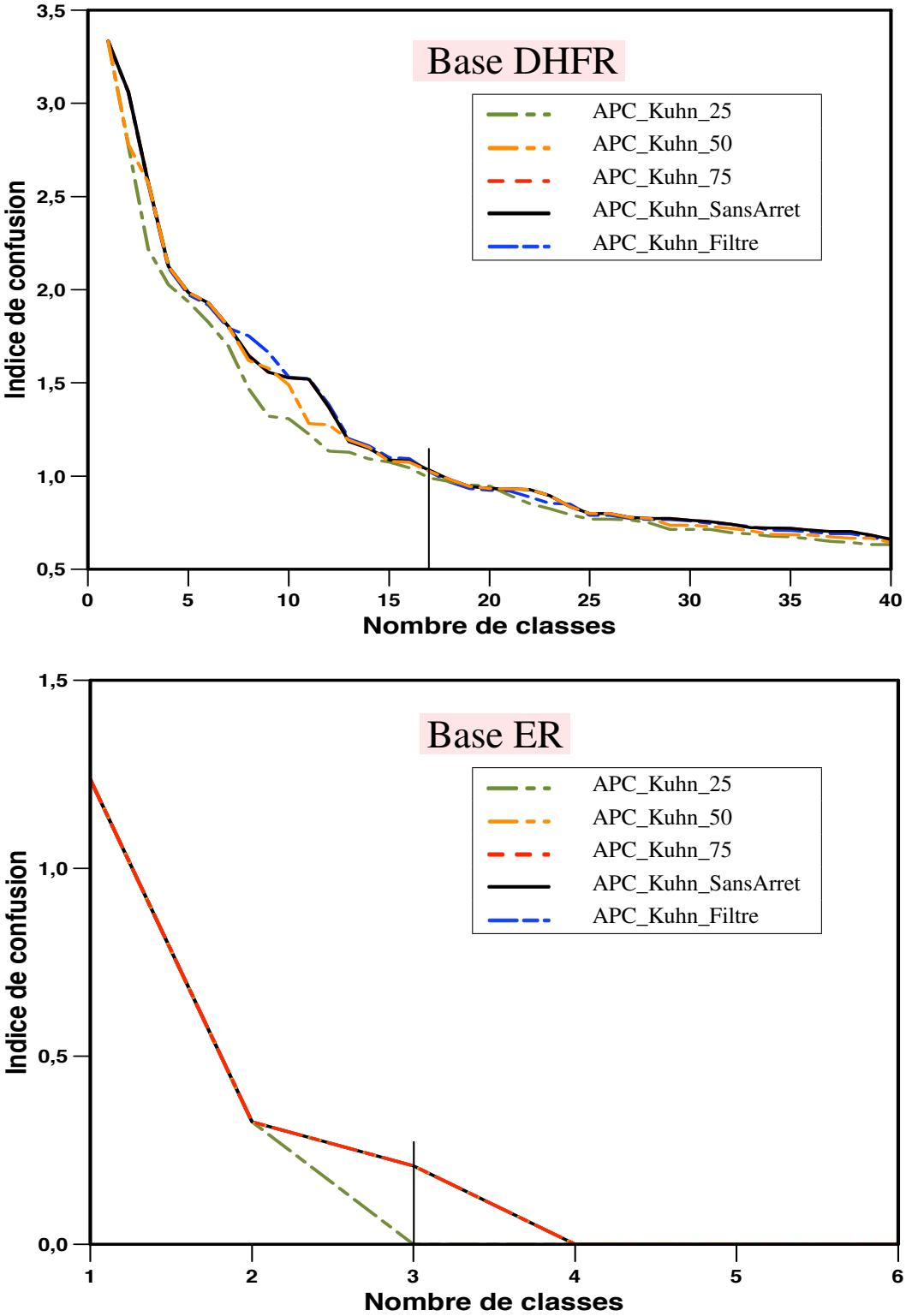


FIGURE B.4 – Courbes des indices IC pour des nombres de graines différents avec la variante Ipi_*_APC, sur les bases DHFR et ER.

Annexe C

Comparaison des variantes de I_π pour différentes fonctions Di_{lit}

Le tableau C.1 indique les valeurs numériques des indices IC pour les variantes I_π_Inc_APC et I_π_Inc_APC_P qui utilisent la recherche d'APC pour la phase d'appariement global et où la fonction Di_{lit} est définie par la moyenne (resp. le produit) dans I_π_Inc_APC (resp. I_π_Inc_APC_P) pour un nombre de classes correspondant à la classification initiale dans les chimiothèques.

Méthode	BZR $h=14$	COX2 $h=13$	DHFR $h=17$	ER $h=3$
I_π_Inc_APC	0.35	0.35	1.03	0.21
I_π_Inc_APC_P	0.32	0.31	1.02	0.21

Tableau C.1 – Valeur de l'indice IC pour les variantes I_π_Inc_APC et I_π_Inc_APC_P de l'algorithme I_π, au point du nombre de classes attendu.

Les courbes correspondant aux indices IC sont affichées dans les figures C.1 et C.2.

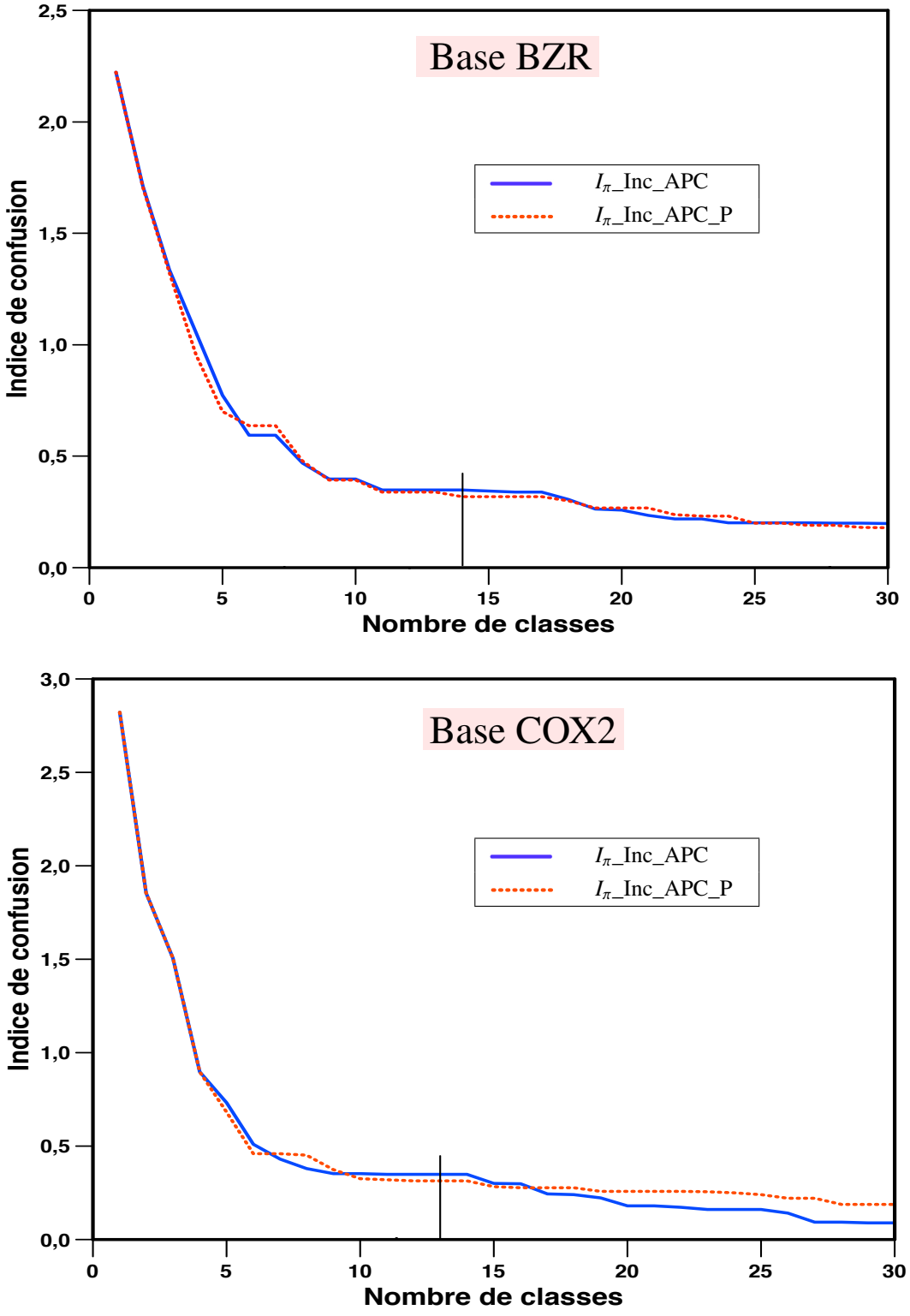


FIGURE C.1 – Comparaison des indices IC pour les variantes I_π_Inc_APC et I_π_Inc_APC_P pour les bases BZR et COX2.

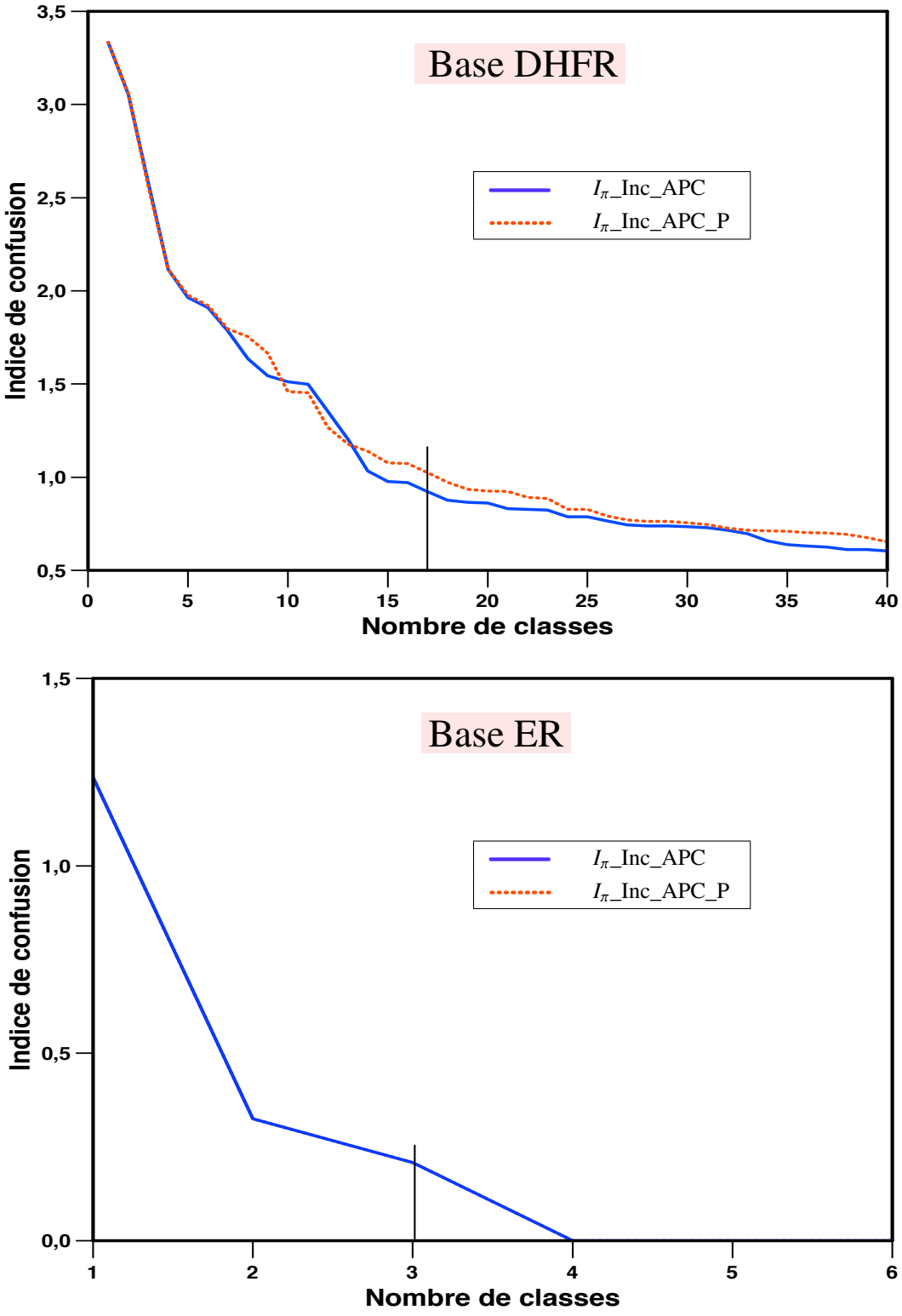


FIGURE C.2 – Comparaison des indices IC des variantes I_π_Inc_APC et I_π_Inc_APC_P pour les bases DHFR et ER.

Bibliographie

[Achlioptas *et al.*, 2000] Dimitris Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. *Proc. of AAAI'00*, pages 256–301, 2000.

[Aci *et al.*, 2007] S. Aci, G. Bisson, S. Roy, and S. Wieczorek. Classification automatique de molécules. In *Proc. CAP*, 2007.

[Adamson and Bawden, 1981] George W. Adamson and David Bawden. Comparison of hierarchical cluster analysis techniques for automatic classification of chemical structures. *J. Chem. Inf. Comput. Sci.*, 21(4) :204–209, 1981.

[Adamson and Bush, 1975] George W. Adamson and Judith A. Bush. A comparison of the performance of some similarity and dissimilarity measures in the automatic classification of chemical structures. *J. Chem. Inf. Comput. Sci*, 15(1) :55–58, 1975.

[Adamson, 1973] J. A. Adamson, G. W. et Bush. A method for the automatic classification of chemical structures. *The entity from which ERIC acquires the content, including journal, organization, and conference names, or by means of online submission from the author. Information Storage and Retrieval*, 9(10) :561–568, 1973.

[Alphonse and Osmani, 2008] E. Alphonse and A. Osmani. On the connection between the phase transition of the covering test and the learning success rate in ilp. *Machine Learning*, 2008.

[Alphonse and Rouveirol, 2006] E. Alphonse and C. Rouveirol. Extension of the top-down data-driven strategy to ilp. In *Proc. of Int. Conf. On Inductive Logic Programming*. Springer Verlag, 2006.

[Arias *et al.*, 2005] Marta Arias, Roni Khardon, and Jérôme Maloberti. Learning horn expressions with logan-h. Technical Report 4, Department of Computer Science Tufts University, 2005.

[Azencott *et al.*, 2007] Chloé-Agathe Azencott, Alexandre Ksikes, S. Joshua Swamidass, Jonathan H. Chen, Liva Ralaivola, and Pierre Baldi. One- to four-dimensional kernels for virtual screening and the prediction of physical, chemical, and biological properties. *J. Chem. Inf. Model.*, 47(3) :965–974, 2007.

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook : Theory, Implementation, Applications*. Cambridge University Press, 2003.

[Barker *et al.*, 2006] E. J. Barker, D. Buttar, A. Cosgrove, E. J. Gardiner, P. Kitts, P. Willett, and V. J. Gillet. Scaffold hopping using clique detection applied to reduces graphs. *Journal of Chemical Information and Modeling*, 46 :503–511, 2006.

[Barnard and Downs, 1992] J. M. Barnard and G. M. Downs. Clustering of chemical structures on the basis of two-dimensional similarity measures. *J. Chem. Inf. Comput. Sci*, 32 :644–649, 1992.

[Bayada *et al.*, 1992] D. Bayada, R. W. Simpson, and A. P. Johnson. An algorithm for the multiple common subgraph problem. *J. Chem. Inf. Comput. Sci.*, 32 :680–685, 1992.

[Ben-Hur and Noble, 2005] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21 :38–46, 2005.

[Bender *et al.*, 2004] A. Bender, H.Y. Mussa, R.C. Glen, and S. Reiling. Molecular similarity searching using atom environments, information-based feature selection, and a native bayesian classifier. *Journal of Chemical Information and Computer Sciences*, 44(1) :170–178, 2004.

[Berge, 1958] C. Berge. *Théorie des Graphes et ses Applications*. 1958.

[Berge, 1969] C. Berge. *Graphes et Hypergraphes*. 1969.

[Berge, 1987] C. Berge. *Hypergraphes. Combinatoires des ensembles finis*. Gauthier-Villars, 1987.

[Bisson, 1992] G. Bisson. Learning in fol with a similarity measure. In *Proceeding of 10th AAAI Conference*, San-Jose 1992.

[Bisson, 1995] G. Bisson. Why and how to define a similarity measure for object-based representation systems. In IOS press, editor, *Proceedings of 2nd International Conference on Building and Sharing Very Large-scale knowledge bases (KBKS)*, pages 236–246, 1995.

[Borgwardt, 2007] Karsten Michael Borgwardt. *Graph kernels*. PhD thesis, Fakultät für Mathematik, Informatik und Statistik - Universität München, 2007.

[Botta *et al.*, 2000] M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning : Hard problems and phase transitions. *Lecture notes in computer science*, 1792 :178–189, 2000.

[Botta *et al.*, 2003] Marco Botta, Lorenza Saitta, and Michèle Sebag. Relational learning as search in a critical region. *Journal of Machine Learning research*, 4 :431–463, 2003.

[Bratchell, 1989] N. Bratchell. Clsuter analysis. *Chemom. Intell. Lab. Systems*, 6 :105, 1989.

[Buhot and Gordon, 1998] A. Buhot and M. B. Gordon. Phase transitions in optimal unsupervised learning. *Phys. Rev. E*, 57 :3326–3333, 1998.

[Bunke and Shearer, 1998] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19 :255–259, 1998.

[Candellier *et al.*, 2006] L. Candellier, I. Tellier, F. Torre, and O. Bousquet. Cascade evaluation of clustering algorithms. In *Proceedings of ECML*, 2006.

[Cao *et al.*, 2008] Y. Cao, T. Jiang, and T. Girke. A maximum common substrucure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24 :i366–i374, 2008.

[Ceri *et al.*, 1989] S Ceri, G Gottlob, and L Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1) :146–166, 1989.

[Ceroni *et al.*, 2007] A. Ceroni, F. Costa, and P. Frasconi. Classification of small molecules by two- and three-dimensional decomposition kernels. *Bioinformatics*, 23(16) :2038–2045, 2007.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. M. Taylor. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 331–337, 1991.

[Cone *et al.*, 1977] M. M. Cone, Rengachari Venkataraghven, and F. W. McLafferty. Molecular structure comparison program for the identification of maximal common substructures. *Journal of American Chemical Society*, 99(23) :7668–7671, 1977.

[Conte *et al.*, 2004] D Conte, P Foggia, C Sansone, and M Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.

[Corter and Gluck, 1992] James E. Corter and Mark A. Gluck. Explaining basic categories : Feature predictability and information. *Psychological Bulletin*, 111(2) :291–303, 1992.

[Cristiani, 2000] Cristiani. *Introduction to Support Vector Machines*. Cambridge University press, 2000.

[Denoeud and Guénoche, 2006] Lucile Denoeud and Alain Guénoche. *Comparison of Distance Indices Between Partitions*, pages 21–28. Studies in Classification, Data Analysis, and Knowledge Organization. Vladimir Batagelj AND Hans-Hermann Bock AND Anuška Ferligoj and Aleš Žiberna, 2006.

[Dixon and Koehler, 1999] S. L. Dixon and R. T. Koehler. The hidden component of size in two-dimensional fragment descriptors : Side effects on sampling in bioactive libraries. *Journal of Medicinal Chemistry*, 42(15) :2887–2900, 1999.

[Downs and Barnard, 2002] G. M. Downs and J. M. Barnard. Clustering methods and their uses in computational chemistry. *Rev. in Comput. Chem.*, 18(1-40), 2002.

[Downs *et al.*, 1994] G. M. Downs, P. Willett, and W. Fisanick. Similarity searching and clustering of chemical-structure databases using molecular property data. *J. Chem. Inf. Comput. Sci.*, 34(5) :1094, 1994.

[Dubois *et al.*, 2008] J. Dubois, S. Bourg, C. Vrain, and L. Morin-Allory. Collections of compounds - how to deal with them ? *Current Computer-Aided Drug Design*, 4, 2008.

[Eckert and Bajorath, 2007] H. Eckert and J. Bajorath. Molecular similarity analysis in virtual screening : Foundations, limitations and novel approaches. *Drug Discovery Today*, 12 :225–233, 2007.

[El-Sonbaty and Ismail, 2000] Y. El-Sonbaty and M. A. Ismail. *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, chapter A New Error-Correcting Distance for Attributed Relational Graph Problems, pages 266–276. 2000.

[Engels, 2000] M. F. M. Engels. Cerberus : A system supporting the sequential screening process. *J. Chem. Inf. Comput. Sci*, 40 :241–245, 2000.

[Faulon *et al.*, 2004] J.-L. Faulon, M. Collins, and R. D. Carr. Canonizing molecules using extended valence sequence. *J. Chem. Inf. Comput. Sci.*, 44 :427–436, 2004.

[Faulon *et al.*, 2008] Jean-Loup Faulon, Milind Misra, Shawn Martin, Ken Sale, and Rajat Sapra. Genome scale enzyme–metabolite and drug–target interaction predictions using the signature molecular descriptor. *Bioinformatics*, 24(2) :225–233, 2008.

[Ferilli *et al.*, 2002] Stefano Ferilli, Nicola Fanizzi, Nicola Di Mauro, and Teresa M. A. Basile, editors. *Efficient Theta-subsumption under Object Identity*. del Workshop AI*IA 2002 su Apprendimento Automatico : Metodi e Applicazioni dell'Ottavo Convegno della Associazione Italiana per l'Intelligenza Artificiale, 2002.

[Flower, 1998] D. R. Flower. On the properties of bit string-based measures of chemical similarity. *J. Chem. Inf. Comput.*, 38 :378–386, 1998.

[Fox *et al.*, 2006] S. Fox, Farr-Jones S, Sopchak L, Boggs A, Nicely HW, Khoury R, and Biros M. High-throughput screening : update on practices and success. *J. Biomolecular Screening*, 11 :864–849, 2006.

[Freuder and Wallace, 1992] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70, 1992.

[Fröhlich *et al.*, 2005a] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

[Fröhlich *et al.*, 2005b] H. Fröhlich, J. K. Wegner, and A. Zell. Assignment kernels for chemical compounds. In *International Joint Conference on Neural Networks 2005 (IJCNN'05)*, pages 913–918, 2005.

[Fürnkranz and Flach, 2005] J. Fürnkranz and P. Flach. Roc 'n' rule learning-towards a better understanding of covering algorithms. *Machine Learning*, pages 39–77, 2005.

[Gärtner *et al.*, 2003] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels : Hardness results and efficient alternatives. In *Proc. 16th Ann. Conf. Comp. Learning Theory and 7th Ann. Workshop on Kernel Machines*, 2003.

[Gärtner, 2003a] T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS workshop on Unreal Data : Principles of Modeling Nonvectorial Data*, 2003.

[Gärtner, 2003b] Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1) :49–58, 2003.

[Giordana *et al.*, 2000] Attilio Giordana, Lorenza Saitta, Michèle Sebag, and Marco Botta. Can relational learning scale up ? In *ISMIS '00 Lecture Notes In Computer Science ; Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*, pages 31–39. Springer-Verlag London, UK, 2000.

[Gluck and Corter, 1985] Mark A. Gluck and James E. Corter. Information, uncertainty, and the utility of categories. In *Program of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287, 1985.

[Gusfield, 2002] D. Gusfield. Partition-distance a problem and class of perfect graphs arising in clustering. *Inf. Proc. Lett.*, 82 :159–164, 2002.

[Hagadone, 1992] T. R. Hagadone. Molecular substructure similarity searching : efficient retrieval in two-dimensional structure databases. *J. Chem. Inf. Comput. Sci*, (32) :515–521, 1992.

[Haranczyk and Holliday, 2008] Maciej Haranczyk and John Holliday. Comparison of similarity coefficients for clustering and compound selection. *J. Chem. Inf. Model.*, 48(3) :498–508, 2008.

[Haussler, 1999] David Haussler. Convolution kernels on discrete structures. 1999.

[Hogg *et al.*, 1996] T. Hogg, B. A. Huberman, and C. P. Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81 :1–15, 1996.

[Hölldobler and Skvortsova, 2006] S. Hölldobler and O. Skvortsova. Yet another context-based theta-subsumption for ai planning. 2006.

[Holliday *et al.*, 2004] J. D. Holliday, S. L. Rodgers, P. Willett, M.-Y. Chen, M. Mahfouf, K. Lawson, and G. Mullier. Clustering files of chemical structures using the fuzzy k-means clustering method. *Journal of Chemical Information and Computer Sciences*, 44(3) :894–902, 2004.

[http ://www.chemaxon.com/product/jc_base.html,] http ://www.chemaxon.com/product/jc_base.html. http ://www.chemaxon.com/product/jc_base.html.

[Inglese *et al.*, 2007] James Inglese, Ronald L Johnson, Anton Simeonov, Menghang Xia, Wei Zheng, Christopher P Austin, and Douglas S Auld. High-throughput screening assays for the identification of chemical probes. *Nature Chemical Biology*, 3 :466–479, 2007.

[Jacob and Vert, 2008] L. Jacob and J.-P. Vert. Protein-ligand interaction prediction : an improved chemogenomics approach. *Bioinformatics*, 24(19) :2149–2156, 2008.

[Jarvis and Patrick, 1973] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C22 :1025–1034, 1973.

[JKlustor,] JKlustor. http ://www.chemaxon.com/product/jklustor.html.

[Johnson and Maggiora, 1990] A. M. Johnson and G. M. Maggiora. *Concepts and Applications of Molecular Similarity*. Wiley, New York, 1990.

[Karabaev *et al.*, 2006] E. Karabaev, G. Rammé, and O. Skvortsova. Efficient symbolic reasoning for first-order mdps. In *Workshop on "Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds", at the 17th European Conference on Artificial Intelligence*, 2006.

[Kashima *et al.*, 2003] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning, Washington DC*, 2003.

[Kashima *et al.*, 2004] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. *Kernels Methods in Computational Biology*, chapter Kernels for Graphs. MIT PRess, 2004.

[Kondor and Lafferty, 2002] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the ICML*, 2002.

[Kozak *et al.*, 2007] K. Kozak, M. Kozak, and K. Stapor. *Kernels for Chemical Compounds in Biological Screening*, volume 4432 of *Lecture Notes in Computer Science*, chapter Kernels for Chemical Compounds in Biological Screening, pages 327–337. 2007.

[Kuhn, 1955] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Quarterly*, 2 :83, 1955.

[Landwehr *et al.*, 2006] Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. kfoil : Learning simple relational kernels. In *Proc. of AAAI06*, 2006.

[Leslie *et al.*, 2002] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel : a string kernel for svm protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[Lin, 1998] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[Mahé *et al.*, 2005] P. Mahé, N. Ueda, T. Akutsu, and J.-P. Vert. Extensions of marginalized graph kernels. In *Proc. of ICML*, 2005.

[Mahé *et al.*, 2006] P. Mahé, L. Ralaivola, V. Stoven, and J.-P. Vert. The pharmacophore kernel for virtual screening with support vector. *Journal of Chemical Information and Modeling*, 46(5) :2003–2014, 2006.

[Maloberti and Sebag, 2004] Jérôme Maloberti and Michèle Sebag. Fast theta-subsumption with constraint satisfaction algorithms. 2004.

[Maloberti, 2005] Jérôme Maloberti. *Improving Inductive Logic Programming with Constraint Satisfaction Techniques : Applications to Frequent Query Discovery*. PhD thesis, Laboratoire de Recherche en Informatique, Université Paris-Sud - Orsay, 2005.

[Maréchal, 2008] Eric Maréchal. Chemongenomics : A discipline at the crossroad of high throughput technologies, biomarker research, combinatorial chemistry, genomics, cheminformatics, bioinformatics and artificial intelligence. *Combinatorial Chemistry and High Throughput Screening*, 11 :583–586, 2008.

[Mayer, 2003] T. U. Mayer. Chemical genetics : tailoring tools for cell biology. *Trends Cell Biology*, 13 :270–277, 2003.

[McGregor and Willett, 1981] J. J. McGregor and P. Willett. Use of a maximum common subgraph algorithm in the automatic identification of ostensible bond changes occurring in chemical reactions. *Journal of Chemical Information and Computer Sciences*, 21(3) :137–140, 1981.

[MDL, 2009] MDL. http ://www.mdl.com/downloads/public/ctfile/ctfile.pdf, 2009.

[Menchetti *et al.*, 2005] S. Menchetti, F. Costa, and P. Frasconi. Weighted decomposition kernels. In *ICML*, 2005.

[Milligan, 1980] G. W. Milligan. A monte carlo study of thirty internal criterion measures for cluster analysis. *Psychometrika*, 46 :187–199, 1980.

[Minton *et al.*, 1993] Steven Minton, Andy Philips, Mark D. Johnston, and Philip Laird. Minimizing conflicts : A heuristic repair method for constraint-satisfaction and scheduling problems. *Journal of Artificial Intelligence Research*, 1 :1–15, 1993.

[Mitchell, 1982] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18 :203–226, 1982.

[Mohr *et al.*, 2008] Johannes A. Mohr, Brijnesh J. Jain, and Klaus Obermayer. Molecule kernels : A descriptor- and alignment-free quantitative structure - activity relationship approach. *Journal of Chemical Information and Modeling*, 48(9) :1868–1881, 2008.

[Morgan, 1965] H. Morgan. The generation of unique machine description for chemical structures - a techniqe developped at chemical abstracts service. *Journal of Chemical Documentation*, 5(2) :107–113, 1965.

[Muggleton and Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19 :629–679, 1994.

[Muggleton, 1992] S. Muggleton. *Inductive Logic Programming*. The APIC Series. Academic Press, 1992.

[Nicolaou *et al.*, 2002] C. A. Nicolaou, S. Y. Tamura, B. P. Kelley, S. I. Bassett, and R. F. Nutt. Analysis of large screening data sets via adaptively grown phylogenetic-like trees. *J. Chem. Inf. Comput. Sci*, 42(5) :1069–1079, 2002.

[Nikolova and Jaworska, 2003] N. Nikolova and J. Jaworska. Approaches to measure chemical similarity - a review. *QSAR Comb. Sci*, pages 1006–1026, 2003.

[Ning *et al.*, 2008] Xia Ning, Huzefa Rangwala, and George Karypis. Improved sar models - exploiting the target-ligand relationships. 2008.

[Ott *et al.*, 2004] T. Ott, A. Kern, A. Schuffenhauer, M. Popov, P. Acklin., E. Jacoby, and R. Stoop. Sequential superparamagnetic clustering for unbiased classification of high-dimensional chemical data. *Journal of Chemical Information and Computer Sciences*, 44(4) :1358–1364, 2004.

[Palmer, 1985] E. M. Palmer. *Graphical Evolution*. Wiley, 1985.

[Parker and Schreyer, 2004] C. N. Parker and S. K. Schreyer. Application of chemoinformatics to high-throughput screening. In J. Bajorath, editor, *Chemoinformatics : Concepts, Methods and Tools for Drug Discovery*, volume 275 of *Methods in Molecular Biology*. Humana Press Inc, 2004.

[Pernot *et al.*, 2005] N. Pernot, A. Cornuéjols, and M. Sebag. Phase transition within grammatical inference. In *Proceedings of Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland, 2005.

[Petit-Zeman, 2003] S. Petit-Zeman. Charting chemical space : finding new tools to explore biology. In *4th Horizon Symposium*, 2003.

[Plotkin, 1970] G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5, 1970.

[Prosser, 1996] Patrick Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81 :81–109, 1996.

[Qiu *et al.*, 2007] J. Qiu, M. Hue, A. Ben-Hur, J.-P. Vert, and W. S. Noble. A structural alignment kernel for protein structures. *Bioinformatics*, 23 :1090–1098, 2007.

[Ralaivola *et al.*, 2005] Liva Ralaivola, Sanjay J. Swamidass, Hiroti Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks, special issue on Neural Networks and Kernel Methods for Structured Domains,*, 18(8) :1093–1110, 2005.

[Raymond and Willett, 2002] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16 :521–533, 2002.

[Raymond *et al.*, 2002a] J. W. Raymond, E. J. Gardiner, and P. Willett. Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *J. Chem. Inf. Comput. Sci.*, 42(2) :305–316, 2002.

[Raymond *et al.*, 2002b] J. W. Raymond, E. J. Gardiner, and P. Willett. Rascal : Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6), 2002.

[Raymond *et al.*, 2003] J. W. Raymond, C. J. Blankley, and P. Willett. Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures. *Journal of Molecular Graphics and Modelling*, 21(5) :421–433, 2003.

[Rensink, 2004] Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335. Springer Verlag, 2004.

[Reynolds *et al.*, 1998] C. H. Reynolds, R. Druker, and L. B. Pfahler. Lead discovery using stochastic cluster analysis (sca) : A new method for clustering structurally similar compounds. *J. Chem. Inf. Comput. Sci*, 38(2) :305–312, 1998.

[Rubin and Willett, 1983] V. Rubin and P. Willett. A comparison of some hierarchical monothetic divisive clustering algorithms for structure-property correlation. *Anal. Chim. Acta*, 151 :161, 1983.

[Ruiz *et al.*, 2005] I. L. Ruiz, G. C.Garcia, and M. A. Gomez-Nieto. Clustering chemical databases using adaptable projection cells and mcs similarity values. *J. Chem. Inf. Comput. Sci*, 45 :1178–1194, 2005.

[Rupp *et al.*, 2007] Matthias Rupp, Ewgenij Proschak, and Gisbert Schneider. Kernel approach to molecular similarity based on iterative graph similarity. *Journal of Chemical Information and Modeling*, 47(6) :2280–2286, 2007.

[Scheffer *et al.*, 1996] T. Scheffer, R. Herbrich, and F. Wysotzki. Efficient theta-subsumption based on graph algorithms. In *ILP workshop*, 1996.

[Schölkopf and Smola, 2002] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT PRess, Cambridge, MA, 2002.

[Scholkopf *et al.*, 2004] B. Scholkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology*. The MIT Press, 2004.

[Schuffenhauer *et al.*, 2000] A. Schuffenhauer, V. J. Gillet, and P. Willett. Similarity searching in files of three-dimensional chemical structures : Analysis of the bioster database using two-dimensional fingerprints and molecular field descriptors. *Journal of Chemical Information and Computer Sciences*, 40(2) :295–307, 2000.

[Sheridan, 2002] R.P. Sheridan. The most common chemical replacements in drug-like compounds. *Journal of Chemical Information and Computer Sciences*, 42(1) :103–108, 2002.

[Singh *et al.*, 2008] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105(35) :12763–12768, 2008.

[Smith, 1994] B. M. Smith. Locating the phase transition in binary constraint satisfaction problems. Technical Report 94.16, University of Leeds, 1994.

[Sorlin and Solnon, 2004] S. Sorlin and C. Solnon. A global constraint for graph isomorphism problems. In *LNCS*, volume 3011, pages 287–302, 2004.

[Sorlin *et al.*, 2003] S. Sorlin, P.-A.Champin, and C. Solnon. Mesurer la similarité de graphes étiquetés. In *9èmes Journées Nationales sur la résolution pratique de problèmes NP-Complets (JNPC 2003) - pages 325-339 Juin 2003*, pages 91–107. Hermès, 2003.

[Sorlin, 2006] Sébastien Sorlin. *Mesurer la similarité de graphes*. PhD thesis, Université Claude Bernard Lyon I, 2006.

[Sowa, 1984] Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Sowa, 1984.

[Stahl and Mauser, 2005] M. Stahl and H. Mauser. Database clustering with a combination of fingerprint and maximum common substructure methods. *Journal of Chemical Information and Modeling*, 45(3) :542–548, 2005.

[Stahl *et al.*, 2005] M. Stahl, H. Mauser, M. Tsui, and N. R. Taylor. A robust clustering method for chemical structures. *J. Med. Chem.*, 48 :4358–4366, 2005.

[Stein, 2003] R. L. Stein. High-throughput screening in academia : the harvard experience. *J. Biomolecular Screening*, 8 :615–619, 2003.

[Stockwell, 2000] B. R. Stockwell. Frontiers in chemical genetics. *Trends Biotechnologies*, 18 :449–455, 2000.

[Sutherland *et al.*, 2003] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver. Spline-fitting with a genetic algorithm : A method for developing classification structure-activity relationships. *J. Chem. Inf. Comput. Sci.*, 43 :1906–1915, 2003.

[Swamidass *et al.*, 2005] S. Joshua Swamidass, Jonathan Chen, Jocelyne Bruand, Peter Phung, Liva Ralaivola, and Pierre Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21, Suupl 1 :359–368, 2005.

[Takahashi *et al.*, 1987] H. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki. Recognition of largest common structural fragment among a variety of chemical structures. *Analytical sciences*, 3(1) :23–28, 1987.

[Todeschini and Consonni, 2000] R. Todeschini and V. Consonni. *Handbook of Molecular Descriptors*, volume 11 of *Methods and Principles in Medicinal Chemistry*. Wiley-VCH, 2000.

[Tsang, 1993] Edward Tsang. *Foundations of Constraint Satisfaction*. Computation in Cognitive Science. Academic Press, 1993.

[Tversky, 1977] A. Tversky. Features of similarity. *Psychological Reviews*, 84(4) :327–352, 1977.

[Vert, 2008] J.-P. Vert. The optimal assignment kernel is not positive definite. *CoRR*, 2008.

[Wieczorek *et al.*, 2006a] S. Wieczorek, S. Aci, S. Roy, G. Bisson, and M.B. Gordon. Partial subsumption test and phase transition. In S. Muggleton and R. Otero, editors, *Proc. of the 16th Int. Conf. On Inductive Logic Programming*, pages 222–224, 2006.

[Wieczorek *et al.*, 2006b] S. Wieczorek, G. Bisson, and M.B. Gordon. Guiding ilp search in the sparse solutions region with a partial subsumption test. In J. Fürnkranz, T. Scheffer, and M. SPiliopoulou, editors, *European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Artificial intelligence*, pages 817–824, Berlin, Heidelberg, 2006. Springer Verlag.

[Willett *et al.*, 1986] P. Willett, V. Winterman, and D. Bawden. Implementation of nonhierarchic cluster analysis methods in chemical information systems : selection of compounds for biological testing and clustering of substructure search output. *J. Chem. Inf. Comput. Sci*, 26 :109–118, 1986.

[Willett, 1984] Peter Willett. Evaluation of relocation clustering algorithms for the automatic classification of chemical structures. *J. Chem. Inf. Comput. Sci*, 24(1) :29–33, 1984.

[Willett, 1987] P. Willett. *Similarity and Clustering in Chemical Information Systems*. Research Studies Press, 1987.

[Willett, 1998] P. Willett. Chemical similarity searching. *J. Chem. Inf. Comput. Sci.*, 38 :983–996, 1998.

[Willett, 1999] P. Willett. Matching of chemical and biological structures using subgraph and maximal common subgraph isomorphism algorithms. *IMA Volume Math. and Its Applications*, 108 :11–38, 1999.

[Willett, 2006] P. Willett. Similarity-based virtual screening using 2d fingerprints. *Drug Discovery Today*, 11 :1046–1053, 2006.

[Winston, 1975] P. H. Winston. *Learning Structural descriptions from examples.* P. H. Winston, 1975.

[Wysotzki *et al.*, 1981] F. Wysotzki, J. Selbig, and W. Kolbe. Concept learning by structured examples - an algebraic approach. In *In Proc. of the 7th International Joint Conference on Artificial Intelligence*, 1981.

[Xu and Li, 2006] Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355 :291–302, 2006.

[Xu *et al.*, 2005] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. In *Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 337–342, 2005.

[Xue *et al.*, 2003] L. Xue, J. W. Godden, F. L. Stahura, and J. Bajorath. Profile scaling increases the similarity search performance of molecular fingerprints containing numercila descriptors and structural keys. *J. Chem. Inf. Comput. Sci*, 43 :1218–1225, 2003.

[Youness, 2004] Genane Youness. *Contributions à une méthodologie de comparaison de partitions*. PhD thesis, Université Paris 6, 2004.

[Yuan and Bajorath, 2008] W. Yuan and J. Bajorath. Balancing the influence of molecular complexity on fingerprint similarity searching. *Journal of chemical information and modeling*, 48(1) :75–84, 2008.

[Zampelli *et al.*, 2007] S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, and P. Dupont. *Filtering for Subgraph Isomorphism*, volume 4741 of *Lecture Notes in Computer Science*. 2007.

Résumé

L'identification de molécules bio-actives est un problème majeur pour la recherche thérapeutique et la recherche en biologie. La découverte de ces molécules repose largement sur le criblage de très grandes collections de molécules mais qui restent petites devant la taille de l'espace chimique. Dans ce contexte, les scientifiques sont demandeurs d'outils d'analyse automatique de chimiothèques et de molécules. L'objectif de cette thèse est de fournir un outil de comparaison des molécules et plus généralement d'objets structurés. Nous proposons dans ce travail un algorithme générique qui identifie plusieurs sous-structures communes à entre deux objets, représentés par des graphes ou des formules logiques et évalue un *degré d'inclusion* entre ces objets.

Ce degré d'inclusion correspond à un test de subsomption à valeur réelle entre formules logiques qui pourrait compléter le test de θ-subsomption classique dans les algorithmes d'apprentissage relationnel. Dans le domaine de la chimie, une mesure de similarité moléculaire a été définie à partir de deux degrés d'inclusion pour classer des molécules. L'algorithme se révèle être plus performant que les mesures de similarité et fonctions noyau auxquelles il a été comparé. Il pourra être envisagé de l'utiliser dans des problèmes de prédiction de bio-activité.

Mots-clés: inclusion, graphes, logique des prédicats, similarité, classification

Abstract

The identification of bioactive molecules is a major problem in biology and medicinal chemistry. The discovery of such molecules is mainly based on the screening of large chemical libraries, that are small regarding the size of the chemical space. In this context, scientists need automatic tools to analyze and design rational chemical libraries. The subject of this thesis is to provide a tool that is able to compare molecules or, more generally, structured objects. We propose a generic algorithm which identifies several common substructures between two structured objects (such as graphs or logical formulae), and evaluates an inclusion index between theses objects.

This inclusion index corresponds to a real value subsumption test, and should complete the theta subsumption test which is classically used in relational learning algorithms. In the field of chemistry, a molecular similarity measure, defined with two inclusion indexes, allows to classify compounds with respect to their structures. The algorithm is more efficient than the molecular similarity measures or the kernel functions it was compared to. The algorithm may be used to predict the bioactivity of chemical compounds.

Keywords: inclusion, graphs, first order logic, molecular similarity, clustering